aws

Panduan Pengguna untuk Aurora

# Amazon Aurora

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

# Amazon Aurora: Panduan Pengguna untuk Aurora

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan properti dari masing-masing pemilik, yang mungkin berafiliasi, terkait dengan, atau disponsori oleh Amazon, atau tidak.

# Table of Contents

Apa itu Aurora? ..... 1
Model tanggung jawab bersama Amazon RDS ..... 2
Cara kerja Amazon Aurora dengan Amazon RDS ..... 2
Klaster DB Aurora ..... 4
Versi Aurora ..... 6
Mesin basis data yang didukung ..... 6
Menentukan versi Aurora ..... 7
Versi Aurora ..... 7
Peningkatan cluster Aurora DB ..... 13
Dukungan versi Aurora ..... 15
Wilayah dan Zona Ketersediaan ..... 18
AWS Daerah ..... 19
Zona Ketersediaan ..... 28
Zona waktu lokal untuk klaster DB ..... 28
Fitur Aurora yang didukung berdasarkan Wilayah dan mesin ..... 35
Konvensi tabel ..... 36
Deployment Blue/Green ..... 36
Konfigurasi klaster Aurora ..... 37
Aliran aktivitas basis data ..... 37
Mengekspor data klaster ke Amazon S3 ..... 46
Mengekspor data snapshot ke Amazon S3 ..... 48
Basis data global Aurora ..... 48
IAMotentikasi basis data ..... 60
Autentikasi Kerberos ..... 61
Machine Learning Aurora ..... 67
Wawasan Performa ..... 77
Integrasi nol-ETL ..... 87
Proksi RDS ..... 91
Integrasi Secrets Manager ..... 104
Aurora Serverless v2 ..... 104
Aurora Serverless v1 ..... 110
API Data RDS ..... 114
Penambalan nol-downtime () ZDP ..... 124
Fitur asli mesin ..... 125

Koneksi titik akhir ..... 125
Jenis titik akhir Aurora ..... 126
Melihat endpoint ..... 128
Titik akhir dan ketersediaan tinggi ..... 128
Titik akhir cluster ..... 129
Titik akhir pembaca ..... 130
Endpoint instans ..... 131
Titik akhir kustom ..... 132
Kelas instans DB ..... 152
Jenis kelas instans DB ..... 152
Mesin DB yang didukung ..... 155
Menentukan dukungan kelas instans DB di Wilayah AWS ..... 166
Spesifikasi perangkat keras ..... 170
Penyimpanan ..... 183
Gambaran umum penyimpanan Aurora ..... 183
Konten volume klaster ..... 184
Konfigurasi penyimpanan klaster Aurora ..... 184
Bagaimana penyimpanan berubah ukuran ..... 185
Penagihan data ..... 186
Keandalan ..... 187
Perbaikan penyimpanan otomatis ..... 187
Cache halaman yang dapat bertahan ..... 187
Pemulihan dari pengaktifan ulang yang tidak direncanakan ..... 188
Keamanan Aurora ..... 189
Menggunakan SSL dengan cluster Aurora DB ..... 191
Ketersediaan tinggi ..... 191
Ketersediaan data Aurora yang tinggi ..... 191
Ketersediaan yang tinggi untuk instans DB Aurora ..... 192
Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora ..... 193
Toleransi kesalahan ..... 193
Ketersediaan yang tinggi dengan Proksi Amazon RDS ..... 195
Replikasi ..... 195
Replika Aurora ..... 196
Aurora MySQL ..... 198
Aurora PostgreSQL ..... 199
Penagihan instans DB untuk Aurora ..... 199

Instans DB Sesuai Permintaan ... 202
Instans DB terpesan ... 204
Menyiapkan lingkungan Anda ... 220
Mendaftar untuk Akun AWS ... 220
Buat pengguna dengan akses administratif ... 221
Memberikan akses programatis ... 222
Menentukan persyaratan ... 223
Memberikan akses ke klaster DB ... 225
Memulai ... 228
Membuat dan terhubung ke klaster DB Aurora MySQL ... 228
Prasyarat ... 230
Langkah 1: Buat sebuah EC2 instance ... 230
Langkah 2: Membuat klaster DB Aurora MySQL ... 236
(Opsional) Buat VPC, EC2 instance, dan cluster MySQL Aurora menggunakan AWS CloudFormation ... 241
Langkah 3: Membuat klaster DB Aurora MySQL ... 243
Langkah 4: Hapus EC2 instance dan cluster DB ... 246
(Opsional) Hapus EC2 instance dan cluster DB yang dibuat dengan CloudFormation ... 247
(Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda ... 248
Membuat dan terhubung ke klaster DB Aurora PostgreSQL ... 248
Prasyarat ... 250
Langkah 1: Buat sebuah EC2 instance ... 250
Langkah 2: Membuat klaster DB Aurora PostgreSQL ... 256
(Opsional) Buat VPC, EC2 instance, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation ... 261
Langkah 3: Membuat klaster DB Aurora PostgreSQL ... 263
Langkah 4: Hapus EC2 instance dan cluster DB ... 266
(Opsional) Hapus EC2 instance dan cluster DB yang dibuat dengan CloudFormation ... 267
(Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda ... 267
Tutorial: Membuat server web dan klaster DB Amazon Aurora ... 269
Luncurkan EC2 instance untuk terhubung dengan Anda ... 270
Membuat klaster DB ... 276
Menginstal server web ... 287
Tutorial dan kode sampel ... 300
Tutorial dalam panduan ini ... 300
Tutorial dalam AWS panduan lain ... 301

AWS lokakarya dan portal konten lab untuk Aurora PostgreSQL ..... 302
AWS lokakarya dan portal konten lab untuk ..... 303
Tutorial dan kode sampel di GitHub ..... 305
Bekerja dengan AWS SDKs ..... 305
Akses terprogram ke Aurora ..... 307
Konsol-ke-Kode ..... 308
Menyiapkan cluster Aurora DB ..... 309
Membuat klaster DB ..... 310
Prasyarat ..... 311
Membuat klaster DB ..... 317
Pengaturan yang tersedia ..... 327
Pengaturan yang tidak berlaku untuk klaster DB Aurora ..... 349
Pengaturan yang tidak berlaku untuk instans DB Aurora ..... 350
Menciptakan sumber daya dengan AWS CloudFormation ..... 353
Aurora dan template AWS CloudFormation ..... 353
Pelajari lebih lanjut tentang AWS CloudFormation ..... 353
Menghubungkan ke klaster DB ..... 354
Menghubungkan ke cluster Aurora DB dengan driver AWS ..... 355
Menghubungkan ke Aurora MySQL ..... 357
Menghubungkan ke Aurora PostgreSQL ..... 363
Pemecahan masalah koneksi ..... 366
Grup parameter ..... 367
Ikhtisar grup parameter ..... 367
Grup parameter klaster DB ..... 371
Grup parameter DB ..... 389
Membandingkan grup parameter DB ..... 406
Menentukan parameter DB ..... 407
Memigrasikan data ke klaster DB ..... 412
Aurora Saya SQL ..... 412
Aurora Postgre SQL ..... 412
Membuat ElastiCache cache dari Amazon RDS ..... 413
Ikhtisar pembuatan ElastiCache cache dengan pengaturan instans DB cluster Aurora RDS ..... 413
..... 414
Database migrasi otomatis EC2 ..... 417
Gambaran Umum ..... 417

Prasyarat ... 418
Batasan ... 419
Membuat sumber daya IAM ... 420
Mengatur migrasi data ... 427
Mengelola migrasi ... 429
Pemantauan ... 432
Tutorial: Membuat cluster dengan kustom dan grup opsi baru ... 434
Prasyarat ... 434
Buat grup parameter kustom ... 435
Ubah parameter nilai ... 435
Buat cluster DB DB Anda ... 436
Mengelola klaster DB Aurora ... 438
Menghentikan dan memulai klaster ... 439
Ikhtisar menghentikan dan memulai klaster ... 439
Pembatasan ... 440
Menghentikan klaster DB ... 441
Saat klaster DB dihentikan ... 442
Memulai klaster DB ... 442
Menghubungkan sebuah EC2 instance ... 444
Gambaran Umum ... 445
Menghubungkan sebuah EC2 instance ... 450
Melihat sumber daya komputasi terhubung ... 453
Menghubungi instans basis data yang menjalankan mesin basis data tertentu ... 454
Menghubungkan fungsi Lambda ... 455
Ikhtisar ... 457
Menghubungkan fungsi Lambda ... 468
Melihat sumber daya komputasi terhubung ... 470
Memodifikasi klaster DB Aurora ... 472
Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API ... 472
Memodifikasi instans DB dalam klaster DB ... 474
Mengubah kata sandi pengguna utama ... 477
Pengaturan yang tersedia ... 479
Pengaturan yang tidak berlaku untuk klaster DB Aurora ... 515
Pengaturan yang tidak berlaku untuk instans DB Aurora ... 516
Menambahkan Replika Aurora ... 518
Penskalaan Otomatis dengan Replika Aurora ... 525

Menambahkan kebijakan penskalaan otomatis ..... 531
Mengedit kebijakan penskalaan otomatis ..... 544
Menghapus kebijakan penskalaan otomatis ..... 546
Mengelola performa dan penskalaan ..... 549
Penskalaan penyimpanan ..... 549
Penskalaan instans ..... 556
Penskalaan baca ..... 556
Mengelola koneksi ..... 557
Mengelola rencana eksekusi kueri ..... 557
Mengkloning volume untuk klaster DB Aurora ..... 558
Gambaran umum kloning Aurora ..... 558
Batasan kloning Aurora ..... 559
Cara kerja kloning Aurora ..... 560
Membuat klon Aurora ..... 564
VPCKloning silang ..... 575
Kloning lintas akun ..... 593
Integrasi dengan layanan AWS ..... 611
Aurora Saya SQL ..... 611
Aurora Postgre SQL ..... 611
Memelihara klaster DB Aurora ..... 613
Ikhtisar pembaruan pemeliharaan cluster DB ..... 613
Melihat pemeliharaan Tertunda ..... 614
Memilih frekuensi pembaruan pemeliharaan Aurora MySQL ..... 617
Periode pemeliharaan ..... 618
Menerapkan pembaruan ..... 623
Peningkatan versi minor otomatis untuk klaster DB Aurora ..... 625
Pembaruan sistem operasi ..... 630
Mem-boot ulang instans atau klaster DB Aurora ..... 639
Mem-boot ulang instans DB dalam klaster Aurora ..... 640
Mem-boot ulang klaster Aurora dengan ketersediaan baca ..... 641
Mem-boot ulang klaster Aurora tanpa ketersediaan baca ..... 643
Memeriksa waktu aktif untuk klaster dan instans Aurora ..... 644
Contoh operasi boot ulang Aurora ..... 647
Gagal atas cluster Aurora DB ..... 663
Menghapus klaster dan instans Aurora ..... 665
Menghapus klaster DB Aurora ..... 665

Perlindungan penghapusan untuk klaster Aurora ..... 673
Menghapus klaster Aurora yang berhenti ..... 674
Menghapus Aurora Cluster SQL saya yang dibaca replika ..... 674
Snapshot akhir saat menghapus klaster ..... 674
Menghapus instans dari klaster DB Aurora ..... 674
Menandai RDSAurora dan sumber daya ..... 677
Mengapa menggunakan RDS tag? ..... 678
Bagaimana RDS tag bekerja ..... 679
Praktik terbaik ..... 682
Menyalin tag ke snapshot klaster DB ..... 683
Menambahkan dan menghapus tag di Amazon RDS ..... 683
Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan. ..... 688
ARNsdi Amazon RDS ..... 692
Membuat konsep ARN ..... 692
Mendapatkan yang sudah ada ARN ..... 699
Pembaruan Aurora ..... 703
Mengidentifikasi versi Amazon Aurora Anda ..... 703
Dukungan yang Diperluas RDS ..... 705
Ikhtisar RDS Extended Support ..... 706
RDSBiaya Extended Support ..... 707
Menghindari biaya untuk RDS Extended Support ..... 707
Versi dengan RDS Extended Support ..... 708
Tanggung Jawab dengan RDS Extended Support ..... 708
Amazon Aurora Amazon ..... 708
Tanggung jawab Anda ..... 709
Aurora DB atau cluster global ..... 709
RDSPerilaku Support yang Diperpanjang ..... 710
Pertimbangan untuk RDS Extended Support ..... 710
Aurora DB atau cluster global dengan Extended Support RDS ..... 711
Melihat pendaftaran RDS Extended Support ..... 712
Melihat tanggal dukungan ..... 715
Aurora DB atau cluster global ..... 717
RDSPerilaku Support yang Diperpanjang ..... 718
Pertimbangan untuk RDS Extended Support ..... 718
Kembalikan Aurora DB atau cluster global dengan Extended Support RDS ..... 719
Menggunakan Deployment Blue/Green untuk pembaruan basis data ..... 721

Ikhtisar Penerapan Biru/Hijau ..... 722
Wilayah dan ketersediaan versi ..... 723
Manfaat ..... 723
Alur kerja ..... 723
Mengizinkan akses ..... 728
Pertimbangan dan batasan ..... 730
Praktik terbaik ..... 737
Membuat deployment blue/green ..... 741
Mempersiapkan deployment blue/green ..... 741
Menentukan perubahan ..... 743
Membuat deployment blue/green ..... 744
Setelan tersedia ..... 746
Melihat deployment blue/green ..... 748
Mengganti deployment blue/green ..... 751
Waktu habis switchover ..... 752
Pagar pembatas switchover ..... 752
Tindakan switchover ..... 753
Praktik terbaik switchover ..... 754
Memverifikasi CloudWatch metrik sebelum peralihan ..... 755
Memantau kelambatan replika sebelum peralihan ..... 756
Melakukan switchover pada deployment blue/green ..... 757
Setelah switchover ..... 759
Menghapus deployment blue/green ..... 761
Mencadangkan dan memulihkan klaster DB Aurora ..... 765
Gambaran umum pencadangan dan pemulihan ..... 766
Cadangan ..... 766
Jendela cadangan ..... 767
Memulihkan data ..... 770
Kloning basis data ..... 771
Backtrack ..... 771
Mempertahankan cadangan otomatis ..... 772
Periode penahanan ..... 772
Biaya retensi ..... 773
Mencegah penghapusan cadangan otomatis ..... 773
Batasan ..... 773
Melihat cadangan otomatis yang dipertahankan ..... 774

Menghapus cadangan otomatis yang dipertahankan ..... 774
Penyimpanan cadangan ..... 777
Penyimpanan cadangan otomatis ..... 777
Penyimpanan snapshot ..... 777
CloudWatch metrik untuk penyimpanan cadangan ..... 778
Menghitung penggunaan penyimpanan cadangan ..... 779
FAQs ..... 780
Membuat snapshot klaster DB ..... 783
Menentukan apakah snapshot tersedia ..... 785
Memulihkan dari snapshot klaster DB ..... 786
Grup parameter ..... 787
Grup keamanan ..... 787
Pertimbangan Aurora ..... 787
Memulihkan dari snapshot ..... 788
Penyalinan snapshot cluster DB ..... 791
Menyalin snapshot cluster DB dengan AWS Management Console ..... 791
Batasan ..... 793
Pertimbangan ..... 794
Menyalin snapshot cluster DB yang tidak terenkripsi ..... 796
Menyalin snapshot cluster DB terenkripsi ..... 799
Menyalin snapshot klaster DB di seluruh akun ..... 802
Berbagi snapshot klaster DB ..... 807
Berbagi snapshot ..... 808
Berbagi snapshot publik ..... 811
Berbagi snapshot terenkripsi ..... 813
Menghentikan berbagi snapshot ..... 817
Mengekspor data klaster DB ke Amazon S3 ..... 819
Pertimbangan ..... 821
Menyiapkan akses ke bucket S3 ..... 823
Membuat tugas ekspor cluster DB ..... 826
Memantau ekspor klaster DB ..... 830
Membatalkan ekspor klaster DB ..... 832
Pemecahan Masalah ..... 834
Mengekspor data snapshot klaster DB ke Amazon S3 ..... 837
Pertimbangan ..... 839
Menyiapkan akses ke bucket S3 ..... 851

Membuat tugas ekspor snapshot ..... 857
Memantau ekspor snapshot ..... 861
Membatalkan ekspor snapshot ..... 863
Ekspor kinerja di Aurora My SQL ..... 864
Pemecahan Masalah ..... 865
Point-in-time pemulihan ..... 868
Memulihkan cluster DB ke titik waktu ..... 869
oint-in-time Pemulihan P dari cadangan otomatis yang dipertahankan ..... 872
P oint-in-time pemulihan menggunakan AWS Backup ..... 875
Menghapus snapshot klaster DB ..... 881
Menghapus snapshot klaster DB ..... 881
Tutorial: Memulihkan klaster DB dari snapshot ..... 883
Memulihkan klaster DB menggunakan konsol ..... 883
Memulihkan cluster DB menggunakan AWS CLI ..... 888
Memantau metrik di klaster DB Aurora ..... 895
Rencana pemantauan ..... 895
Acuan dasar performa ..... 896
Pedoman performa ..... 896
Alat pemantauan ..... 898
Alat-alat pemantauan otomatis ..... 898
Alat pemantauan manual ..... 900
Melihat status cluster ..... 902
Melihat klaster DB ..... 903
Melihat status klaster DB ..... 909
Melihat status instans DB di klaster Aurora ..... 913
..... 920
Melihat rekomendasi ..... 922
Menerapkan rekomendasi ..... 930
Mengabaikan rekomendasi ..... 936
Memodifikasi rekomendasi yang diberhentikan menjadi aktif ..... 938
Rekomendasi referensi ..... 939
Melihat metrik di konsol Amazon RDS ..... 959
Melihat dasbor Performance Insights ..... 960
Memilih tampilan pemantauan baru dari tab Monitoring ..... 961
Memilih tampilan pemantauan baru dari halaman Performance Insights ..... 963
Membuat dasbor khusus ..... 964

Memilih dasbor yang telah dikonfigurasi sebelumnya ..... 968
Memantau Aurora dengan CloudWatch ..... 970
Ikhtisar Amazon Aurora dan Amazon CloudWatch ..... 971
Melihat CloudWatch metrik ..... 973
Mengekspor metrik Performance Insights ke CloudWatch ..... 978
Membuat CloudWatch alarm ..... 984
Pemantauan dengan Wawasan Database ..... 986
Harga ..... 986
Dukungan kelas Engine, Region, dan instance ..... 986
Mengaktifkan mode Lanjutan ..... 990
Menghidupkan mode Standar ..... 994
Pantau kueri lambat ..... 999
Pertimbangan ..... 1001
Memantau muatan DB dengan Wawasan Performa ..... 1002
Ikhtisar Wawasan Performa ..... 1003
Mengaktifkan dan menonaktifkan Wawasan Performa ..... 1013
Skema Kinerja untuk Aurora MySQL ..... 1020
Kebijakan Wawasan Performa ..... 1025
Menganalisis metrik dengan dasbor Wawasan Performa ..... 1039
Melihat rekomendasi proaktif Performance Insights ..... 1073
Mengambil metrik dengan Performance Insights API ..... 1076
Pencatatan panggilan Performance Insights menggunakan AWS CloudTrail ..... 1101
VPCtitik akhir ()AWS PrivateLink ..... 1103
Menganalisis kinerja dengan DevOps Guru untuk RDS ..... 1107
Manfaat DevOps Guru untuk RDS ..... 1107
Bagaimana DevOps Guru RDS bekerja ..... 1109
Menyiapkan DevOps Guru untuk RDS ..... 1110
Memantau OS dengan Pemantauan yang Ditingkatkan ..... 1119
Gambaran Umum Pemantauan Ditingkatkan ..... 1119
Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan ..... 1121
Melihat metrik OS di konsol RDS ..... 1128
Melihat metrik OS menggunakan CloudWatch Log ..... 1130
Referensi metrik Aurora ..... 1132
CloudWatch metrik untuk Aurora ..... 1132
CloudWatch dimensi untuk Aurora ..... 1173
Ketersediaan metrik Aurora di konsol Amazon RDS ..... 1174

CloudWatch metrik untuk Performance Insights ..... 1177
Metrik penghitung untuk Wawasan Performa ..... 1180
Statistik SQL untuk Wawasan Performa ..... 1211
Metrik OS dalam Pemantauan yang Ditingkatkan ..... 1220
Memantau peristiwa, log, dan aliran aktivitas basis data ..... 1229
Melihat log, peristiwa, dan streaming di konsol Amazon RDS ..... 1230
Memantau peristiwa Aurora ..... 1234
..... 1234
Melihat RDS acara Amazon ..... 1236
Bekerja dengan pemberitahuan RDS acara Amazon ..... 1240
Membuat aturan yang dipicu pada peristiwa Amazon Aurora ..... 1266
Kategori acara Amazon RDS dan pesan acara untuk Aurora ..... 1270
Memantau log Aurora ..... 1301
Melihat dan mencantumkan file log basis data ..... 1301
Mengunduh file log basis data ..... 1303
Melihat file log basis data ..... 1304
Menerbitkan ke CloudWatch Log ..... 1306
Membaca isi file log menggunakan REST ..... 1308
File log basis data MySQL ..... 1310
File log basis data PostgreSQL ..... 1320
Memantau panggilan Aurora masuk API CloudTrail ..... 1331
CloudTrail ..... 1331
Entri file RDS log Amazon Aurora ..... 1332
Memantau Aurora dengan Aliran Aktivitas Basis Data ..... 1336
Ikhtisar ..... 1336
Prasyarat jaringan Aurora MySQL ..... 1340
Memulai aliran aktivitas basis data ..... 1342
Mendapatkan status aliran aktivitas ..... 1345
Menghentikan aliran aktivitas basis data ..... 1347
Memantau aliran aktivitas ..... 1348
Contoh kebijakan IAM untuk aliran aktivitas ..... 1385
Memantau ancaman dengan GuardDuty Perlindungan RDS ..... 1389
Menggunakan Aurora MySQL ..... 1391
Gambaran umum Aurora MySQL ..... 1392
Peningkatan performa Amazon Aurora MySQL ..... 1392
Aurora MySQL dan data spasial ..... 1393

Aurora SQL Versi saya 3 kompatibel dengan My 8.0 SQL ..... 1394
Aurora SQL Versi saya 2 kompatibel dengan My 5.7 SQL ..... 1428
Keamanan dengan Aurora MySQL ..... 1431
Hak akses pengguna master Aurora MySQL ..... 1433
Koneksi TLS ..... 1433
Memperbarui aplikasi untuk menggunakan sertifikat TLS baru ..... 1441
Menentukan apakah ada aplikasi yang tersambung ke klaster DB Aurora MySQL menggunakan TLS ..... 1442
Menentukan apakah klien memerlukan verifikasi sertifikat untuk terhubung ..... 1442
Memperbarui penyimpanan kepercayaan aplikasi Anda ..... 1443
Contoh kode Java untuk membangun koneksi TLS ..... 1444
Menggunakan otentikasi Kerberos untuk Aurora My SQL ..... 1446
Ikhtisar otentikasi Kerberos untuk Aurora My SQL ..... 1447
Batasan ..... 1448
Menyiapkan autentikasi Kerberos untuk Aurora MySQL ..... 1449
Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos ..... 1460
Mengelola klaster DB di dalam domain ..... 1464
Migrasi data ke Aurora My SQL ..... 1466
Bermigrasi dari basis data MySQL eksternal ke Aurora MySQL ..... 1471
Bermigrasi dari instans DB MySQL ke Aurora MySQL ..... 1498
Mengelola Aurora MySQL ..... 1523
Mengelola performa dan penskalaan untuk Amazon Aurora MySQL ..... 1523
Melakukan backtracking klaster DB ..... 1534
Menguji Amazon Aurora My SQL menggunakan kueri injeksi kesalahan ..... 1556
Mengubah tabel di Amazon Aurora menggunakan DDL Cepat ..... 1560
Menampilkan status volume untuk klaster DB Aurora ..... 1567
Menyesuaikan Aurora MySQL ..... 1569
Konsep penting untuk penyesuaian Aurora MySQL ..... 1569
Menyesuaikan Aurora MySQL dengan peristiwa tunggu ..... 1572
Menyesuaikan Aurora MySQL dengan status thread ..... 1625
Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru ..... 1633
Kueri paralel untuk Aurora My SQL ..... 1639
Ikhtisar kueri paralel ..... 1639
Membuat klaster kueri paralel ..... 1643
Mengaktifkan dan menonaktifkan kueri paralel ..... 1648
Mengoptimalkan query paralel ..... 1651

Memverifikasi penggunaan kueri paralel ... 1656
Memantau kueri paralel ... 1660
SQLkonstruksi untuk query paralel ... 1667
Audit Lanjutan dengan Aurora My SQL ... 1689
Mengaktifkan Audit Lanjutan ... 1689
Melihat log audit ... 1693
Detail log audit ... 1693
Replikasi dengan Aurora MySQL ... 1695
Replika Aurora ... 1695
Opsi replikasi ... 1697
Performa replikasi ... 1698
Restart nol-downtime () ZDR ... 1698
Filter replikasi ... 1701
Memantau replikasi ... 1708
Replikasi lintas Wilayah ... 1709
Replikasi log biner (binlog) ... 1725
GTIDreplikasi berbasis ... 1768
Penerusan tulis lokal ... 1775
Mengaktifkan penerusan tulis lokal ... 1776
Memeriksa apakah klaster DB memiliki penerusan tulis yang diaktifkan ... 1778
Kompatibilitas aplikasi dan SQL dengan penerusan tulis ... 1779
Tingkat isolasi untuk penerusan tulis ... 1781
Konsistensi baca ... 1781
Menjalankan pernyataan multibagian dengan penerusan tulis ... 1785
Transaksi dengan penerusan tulis ... 1786
Parameter konfigurasi untuk penerusan tulis ... 1786
Metrik untuk penerusan tulis ... 1788
Mengidentifikasi transaksi dan kueri yang diteruskan ... 1792
Mengintegrasikan Aurora My SQL dengan layanan AWS ... 1794
Mengotorisasi Aurora MySQL untuk mengakses layanan AWS ... 1794
Memuat data dari file teks di Amazon S3 ... 1813
Menyimpan data ke dalam file teks di Amazon S3 ... 1828
Memanggil fungsi Lambda dari Aurora My SQL ... 1839
Menerbitkan log Aurora MySQL ke Log CloudWatch ... 1851
Mode lab Aurora MySQL ... 1857
Fitur mode lab Aurora ... 1857

Praktik terbaik dengan Aurora My SQL ..... 1859
Menentukan ke instans DB mana Anda terhubung ..... 1860
Praktik terbaik untuk kinerja dan penskalaan ..... 1860
Praktik terbaik untuk ketersediaan tinggi ..... 1870
Rekomendasi untuk SQL fitur Saya ..... 1872
Mengevaluasi penggunaan instans DB untuk Aurora My SQL dengan metrik Amazon CloudWatch ..... 1880
Memecahkan masalah Aurora Kinerja saya SQL ..... 1882
AWS opsi pemantauan ..... 1882
Alasan paling umum untuk masalah kinerja DB ..... 1883
Memecahkan masalah beban kerja ..... 1884
Logging untuk Aurora MySQL ..... 1915
Memecahkan masalah koneksi database ..... 1917
Memecahkan masalah kinerja kueri ..... 1932
Aurora Referensi saya SQL ..... 1937
Parameter konfigurasi ..... 1937
Variabel status global ..... 2011
Peristiwa tunggu ..... 2027
Status thread ..... 2033
Tingkat isolasi ..... 2037
Petunjuk ..... 2044
Referensi prosedur tersimpan ..... 2048
Tabel information_schema ..... 2096
Aurora Pembaruan saya SQL ..... 2104
Memeriksa nomor versi ..... 2104
Dukungan jangka panjang dan rilis beta ..... 2107
Mempersiapkan Aurora SQL Versi saya 2 akhir hidup ..... 2109
Mempersiapkan Aurora SQL Versi saya 1 akhir hidup ..... 2114
Memutakhirkan Amazon Aurora SQL My DB cluster ..... 2117
Pembaruan dan perbaikan mesin database untuk Amazon Aurora MySQL ..... 2245
Menggunakan Aurora PostgreSQL ..... 2246
Lingkungan Pratinjau Basis Data ..... 2248
Jenis kelas instans DB yang didukung ..... 2248
Fitur yang tidak didukung di Lingkungan Pratinjau ..... 2249
Membuat klaster DB baru di Preview Environment ..... 2250
Postgre SQL versi 17 di lingkungan Pratinjau Database ..... 2252

Keamanan dengan Aurora PostgreSQL ..... 2253
Memahami peran dan izin PostgreSQL ..... 2254
Mengamankan data Aurora PostgreSQL dengan SSL/TLS ..... 2270
Memperbarui aplikasi untuk sertifikat SSL/TLS baru ..... 2284
Menentukan apakah aplikasi terhubung ke klaster DB Aurora PostgreSQL menggunakan SSL ..... 2285
Menentukan apakah klien memerlukan verifikasi sertifikat agar dapat terhubung ..... 2285
Memperbarui penyimpanan kepercayaan aplikasi Anda ..... 2286
Menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi ..... 2287
Menggunakan autentikasi Kerberos ..... 2288
Wilayah dan ketersediaan versi ..... 2289
Ikhtisar autentikasi Kerberos ..... 2289
Pengaturan ..... 2290
Mengelola Aurora PostgreSQL DB cluster instance PostgreSQL DB dalam domain Active Directory ..... 2305
Menghubungkan dengan autentikasi Kerberos ..... 2307
Menggunakan grup keamanan AD untuk kontrol akses Aurora Postgre SQL ..... 2310
Migrasi data ke Aurora Postgre SQL ..... 2322
Memigrasi instance RDS untuk Postgre SQL DB menggunakan snapshot ..... 2323
Memigrasikan instans DB RDS for PostgreSQL menggunakan replika baca Aurora ..... 2330
Mengoptimalkan kinerja kueri di Aurora PostgreSQL ..... 2344
Meningkatkan performa kueri dengan Aurora Optimized Reads ..... 2345
Mengoptimalkan subkueri berkorelasi di Aurora PostgreSQL ..... 2351
Meningkatkan kinerja kueri menggunakan gabungan adaptif ..... 2358
Bekerja dengan tabel yang tidak tercatat di Aurora PostgreSQL ..... 2360
Membuat tabel yang tidak tercatat ..... 2361
Mengonversi tabel yang tidak tercatat ke tabel yang dicatat ..... 2362
Tabel yang tidak tersumbat dan replikasi logis ..... 2362
Bekerja dengan autovacuum PostgreSQL ..... 2363
Mengalokasikan memori untuk autovacuum ..... 2364
Mengurangi kemungkinan penyelesaian ID transaksi ..... 2365
Menentukan apakah tabel di basis data Anda perlu divakum atau tidak ..... 2366
Menentukan tabel mana yang saat ini memenuhi syarat untuk autovacuum ..... 2368
Menentukan apakah autovacuum saat ini sedang berjalan atau tidak, dan berapa lama durasinya ..... 2369
Melakukan pembekuan vakum manual ..... 2370

Mengindeks ulang tabel saat autovacuum berjalan ... 2373
Mengelola autovacuum dengan indeks berukuran besar ... 2374
Parameter lain yang memengaruhi autovacuum ... 2377
Mengatur parameter autovacuum tingkat tabel ... 2378
Melakukan log aktivitas autovacuum dan vakum ... 2379
Memahami perilaku autovacuum dengan database yang tidak valid ... 2380
Mengidentifikasi penghambat vakum ... 2382
Menggunakan Babelfish for Aurora PostgreSQL ... 2407
Batasan Babelfish ... 2409
Memahami konfigurasi dan arsitektur Babelfish ... 2410
Membuat klaster DB Babelfish for Aurora PostgreSQL ... 2455
Memigrasi basis data SQL Server ke Babelfish ... 2465
Autentikasi basis data dengan Babelfish for Aurora PostgreSQL ... 2475
Menghubungkan ke klaster DB Babelfish ... 2493
Menggunakan Babelfish ... 2506
Memecahkan Masalah Babelfish ... 2585
Menonaktifkan Babelfish ... 2587
Mengelola pembaruan versi Babelfish ... 2588
Referensi Babelfish ... 2610
Kinerja dan penskalaan untuk Aurora PostgreSQL ... 2666
Penskalaan instans DB Aurora PostgreSQL ... 2667
Koneksi maksimum ... 2667
Batas penyimpanan sementara ... 2669
Halaman besar untuk Aurora PostgreSQL ... 2673
Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan ... 2674
Menampilkan status volume untuk klaster DB Aurora ... 2679
Menentukan disk RAM untuk stats_temp_directory ... 2681
Mengelola file sementara dengan PostgreSQL ... 2682
Menyetel dengan peristiwa tunggu di Aurora PostgreSQL ... 2688
Konsep penting untuk penyetelan Aurora PostgreSQL ... 2689
Peristiwa tunggu Aurora PostgreSQL ... 2694
Klien: ClientRead ... 2697
Klien: ClientWrite ... 2700
CPU ... 2702
IO:BufFileRead dan IO:BufFileWrite ... 2708
IO: DataFileRead ... 2717

IO: XactSync ..... 2728
IPC:DamRecordTxAck ..... 2730
IPC: Acara tunggu paralel ..... 2731
Lock:advisory ..... 2738
Lock:extend ..... 2741
Lock:Relation ..... 2744
Lock:transactionid ..... 2751
Lock:tuple ..... 2754
LWLock:buffer_content (BufferContent) ..... 2759
LWLock:buffer_mapping ..... 2761
LWLock:BufferIO (IPC: Bufferio) ..... 2763
LWLock:lock_manager ..... 2765
LWLock:MultiXact ..... 2770
LWLock:pg_stat_statement ..... 2774
Timeout:PgSleep ..... 2778
Menyetel dengan wawasan proaktif Amazon Guru DevOps ..... 2779
Basis data telah lama berjalan idle dalam koneksi transaksi ..... 2780
Praktik terbaik dengan Aurora PostgreSQL ..... 2783
Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans DB Aurora PostgreSQL ..... 2783
Mendiagnosis bloat tabel dan indeks ..... 2784
Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL ..... 2788
Failover cepat ..... 2790
Pemulihan cepat setelah failover ..... 2802
Mengelola churn koneksi ..... 2809
Penanganan koneksi mati di PostgreSQL ..... 2817
..... 2820
Menganalisis penggunaan sumber daya dengan CloudWatch metrik ..... 2829
Menggunakan replikasi logis untuk upgrade versi mayor ..... 2833
Memecahkan masalah penyimpanan di Aurora PostgreSQL ..... 2842
Replikasi dengan Aurora PostgreSQL ..... 2843
Aurora Replica ..... 2844
Meningkatkan ketersediaan baca Aurora Replica ..... 2845
Memantau replikasi ..... 2847
Ikhtisar replikasi logis ..... 2847
Menyiapkan replikasi logis ..... 2848

Menggunakan replikasi logis ..... 2851
Memantau cache write-through dan slot logis ..... 2851
Contoh: Menggunakan replikasi logis ..... 2854
Contoh: Replikasi logis menggunakan SQL Aurora Postgre dan AWS DMS ..... 2856
Penerusan tulis lokal di Aurora PostgreSQL ..... 2858
Keterbatasan dan pertimbangan penerusan tulis lokal di Aurora PostgreSQL ..... 2859
Mengkonfigurasi Aurora SQL Postgre untuk penerusan tulis Lokal ..... 2861
Bekerja dengan penerusan tulis lokal untuk Aurora PostgreSQL ..... 2864
Memantau penerusan penulisan lokal di Aurora Postgre SQL ..... 2868
Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock ..... 2874
Prasyarat ..... 2875
Mempersiapkan Aurora PostgreSQL menjadi Basis Pengetahuan ..... 2875
Membuat Basis Pengetahuan di konsol Bedrock ..... 2879
Cepat membuat SQL Basis Pengetahuan Aurora Postgre untuk Amazon Bedrock ..... 2880
Mengintegrasikan Aurora SQL Postgre dengan layanan AWS ..... 2883
Mengimpor data dari Amazon S3 ke Aurora PostgreSQL ..... 2884
Mengekspor data PostgreSQL ke Amazon S3 ..... 2904
Menginvokasi fungsi Lambda dari Aurora PostgreSQL ..... 2921
Menerbitkan log PostgreSQL Aurora ke Log CloudWatch ..... 2937
Memantau rencana eksekusi kueri dan memori puncak untuk Aurora Postgre SQL ..... 2949
Mengakses rencana eksekusi kueri dan memori puncak menggunakan fungsi Aurora ..... 2949
Referensi parameter untuk rencana eksekusi kueri Aurora Postgre SQL ..... 2950
Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL ..... 2955
Gambaran umum manajemen rencana kueri Aurora PostgreSQL ..... 2956
Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL ..... 2964
Manajemen rencana kueri ..... 2967
Mengambil rencana eksekusi Aurora PostgreSQL ..... 2969
Menggunakan rencana terkelola Aurora PostgreSQL ..... 2972
Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans ..... 2977
Meningkatkan rencana ..... 2978
Menghapus rencana ..... 2982
Mengekspor dan mengimpor rencana terkelola ..... 2984
Referensi parameter ..... 2986
Referensi fungsi ..... 2993
Referensi untuk tampilan apg_plan_mgmt.dba_plans ..... 3004
Fitur lanjutan dalam Manajemen Rencana Kueri ..... 3009

Menangani ekstensi dan pembungkus data asing ..... 3024
Menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk Postgre SQL 3025
Mengelola objek besar secara lebih efisien dengan modul lo ..... 3039
Mengelola data spasial dengan Post GIS ..... 3043
Mengelola partisi dengan ekstensi pg_partman ..... 3053
Menjadwalkan pemeliharaan dengan ekstensi pg_cron ..... 3059
Menggunakan pgAudit untuk mencatat aktivitas database ..... 3068
Menggunakan pglogical untuk menyinkronkan data ..... 3082
..... 3096
Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL ..... 3112
Terminologi ..... 3113
Persyaratan untuk menggunakan Ekstensi Bahasa Tepercaya ..... 3114
Menyiapkan Ekstensi Bahasa Tepercaya ..... 3117
Ikhtisar Ekstensi Bahasa Tepercaya ..... 3121
Membuat TLE ekstensi ..... 3122
Menjatuhkan TLE ekstensi Anda dari database ..... 3127
Menghapus instalan Ekstensi Bahasa Tepercaya ..... 3129
Menggunakan hook PostgreSQL dengan ekstensi TLE ..... 3130
Referensi fungsi untuk Ekstensi Bahasa Tepercaya ..... 3136
Referensi hook untuk Trusted Language Extensions ..... 3149
Referensi Aurora PostgreSQL ..... 3153
Kolasi Aurora PostgreSQL untuk EBCDIC dan migrasi mainframe lainnya ..... 3153
Koleksi yang didukung di Aurora Postgre SQL ..... 3155
Referensi fungsi Aurora PostgreSQL ..... 3155
Parameter Aurora PostgreSQL ..... 3214
Peristiwa tunggu Aurora PostgreSQL ..... 3275
Pembaruan Aurora Postgre SQL ..... 3304
Mengidentifikasi versi Amazon Aurora Postgre SQL ..... 3304
Aurora Postgre rilis SQL ..... 3306
Versi ekstensi untuk Aurora Postgre SQL ..... 3307
Meningkatkan klaster DB Amazon Aurora PostgreSQL ..... 3307
Menggunakan rilis dukungan jangka panjang (LTS) ..... 3347
Menggunakan Basis Data Tanpa Batas Aurora Postgre SQL ..... 3349
Arsitektur Database Tanpa Batas ..... 3350
Istilah kunci ..... 3352
Jenis tabel ..... 3353

Penagihan ..... 3353
Memulai dengan Limitless Database ..... 3354
Persyaratan dan pertimbangan Database Tanpa Batas ..... 3355
Persyaratan ..... 3355
Pertimbangan ..... 3356
Fitur-fitur tidak didukung ..... 3359
Prasyarat untuk menggunakan Limitless Database ..... 3360
Mengaktifkan operasi grup shard DB ..... 3360
Membuat cluster DB yang menggunakan Limitless Database ..... 3362
Menghubungkan kapasitas maksimum dengan router dan pecahan ..... 3362
Membuat cluster DB Anda ..... 3365
Bekerja dengan grup pecahan DB ..... 3374
Menghubungkan ke cluster DB Database PostgreSQL Limitless Aurora Anda ..... 3375
Menemukan jumlah router dan pecahan dalam grup pecahan DB ..... 3376
Menggambarkan kelompok pecahan DB ..... 3376
Mem-boot ulang grup pecahan DB ..... 3377
Mengubah kapasitas grup pecahan DB ..... 3378
Memisahkan pecahan ..... 3380
Menambahkan router ..... 3386
Menghapus grup pecahan DB ..... 3390
Menambahkan grup pecahan DB ke cluster DB Limitless Database yang ada ..... 3391
Membuat tabel Database Tanpa Batas ..... 3396
Membuat tabel tanpa batas dengan menggunakan variabel ..... 3397
Mengonversi tabel standar ke tabel tanpa batas ..... 3402
Skema sampel ..... 3406
Memuat data ke dalam Database Tanpa Batas ..... 3408
Menggunakan COPY perintah dengan Limitless Database ..... 3410
Menggunakan utilitas pemuatan data Limitless Database ..... 3412
Meminta Database Tanpa Batas ..... 3441
Kueri pecahan tunggal ..... 3442
Kueri terdistribusi ..... 3452
Penelusuran kueri terdistribusi ..... 3453
Kebuntuan yang didistribusikan ..... 3461
Mengelola Database Tanpa Batas ..... 3464
Database dan pertimbangan ukuran tabel ..... 3464
Merebut kembali ruang penyimpanan dengan menyedot debu ..... 3465

Memantau Database Tanpa Batas ..... 3469
Memantau Database Tanpa Batas dengan CloudWatch ..... 3470
Memantau Database Tanpa Batas dengan Wawasan Database ..... 3475
Memantau Database Tanpa Batas dengan CloudWatch Log ..... 3487
Memantau Database Tanpa Batas dengan Pemantauan yang Ditingkatkan ..... 3488
Memantau Database Tanpa Batas dengan Performance Insights ..... 3489
Memantau Database Tanpa Batas dengan Perlindungan GuardDuty RDS ..... 3497
Fungsi dan tampilan untuk Limitless Database ..... 3498
Tunggu acara untuk Limitless Database ..... 3524
Mencadangkan dan memulihkan Limitless Database ..... 3537
Mencadangkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database .. 3537
Memulihkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database ..... 3540
Postgre SQL backup dan restore utilitas tidak didukung ..... 3543
Meningkatkan Database Tanpa Batas ..... 3544
Upgrade cluster DB yang menggunakan Aurora PostgreSQL Limitless Database ..... 3544
Referensi Database Tanpa Batas ..... 3546
DDLperintah ..... 3546
Keterbatasan dan informasi DDL ..... 3555
Perintah DML ..... 3593
DMLketerbatasan dan informasi ..... 3596
Variabel ..... 3600
Parameter cluster DB ..... 3601
Menggunakan Aurora Global Database ..... 3602
Ikhtisar Database Global Aurora ..... 3602
Keuntungan dari Amazon Aurora Global Database ..... 3604
Wilayah dan ketersediaan versi ..... 3605
Keterbatasan Database Global Aurora ..... 3605
Memulai dengan Aurora Global Database ..... 3607
Persyaratan konfigurasi ..... 3608
Membuat global database ..... 3610
Menambahkan cluster sekunder ..... 3624
Membuat cluster sekunder tanpa kepala ..... 3628
Membuat database global dari snapshot ..... 3631
Mengelola basis data global Aurora ..... 3633
Memodifikasi basis data global Aurora ..... 3633
Memodifikasi parameter global basis data ..... 3635

Menghapus klaster dari basis data global Aurora ..... 3636
Menghapus basis data global Aurora ..... 3639
Penandaan untuk Aurora Global Database ..... 3641
Menghubungkan ke Aurora Global Database ..... 3646
Memilih titik akhir yang memenuhi kebutuhan aplikasi Anda ..... 3646
Melihat titik akhir dari database global ..... 3648
Pertimbangan dengan menggunakan titik akhir penulis Global ..... 3651
Menggunakan penerusan menulis dalam basis data global Aurora ..... 3652
Menggunakan penerusan tulis di dalam Aurora MySQL ..... 3652
Menggunakan penerusan tulis di Aurora PostgreSQL ..... 3673
Peralihan atau failover untuk Aurora Global Database ..... 3688
Perencanaan untuk BCDR ..... 3688
Melakukan switchover ..... 3690
Memulihkan dari pemadaman yang tidak direncanakan ..... 3696
Mengelola RPOs (Aurora PostgreSQL) ..... 3707
Ketahanan Lintas Wilayah untuk klaster sekunder ..... 3713
Memantau basis data global Aurora ..... 3713
Memantau basis data global Aurora dengan Wawasan Performa ..... 3715
Memantau basis data global Aurora dengan Aliran Aktivitas Basis Data ..... 3716
Memantau Aurora SQL Database global berbasis saya ..... 3716
Memantau basis data global berbasis Aurora Postgre SQL ..... 3720
Menggunakan database global Aurora dengan layanan lain AWS ..... 3723
Meningkatkan basis data global Amazon Aurora ..... 3724
Peningkatan versi utama ..... 3725
Peningkatan versi kecil. ..... 3726
Proksi Amazon RDS ..... 3729
Wilayah dan ketersediaan versi ..... 3730
Kuota dan Batasan ..... 3730
Batasan MySQL ..... 3732
Batasan PostgreSQL ..... 3733
Merencanakan lokasi penggunaan Proksi RDS ..... 3734
RDSKonsep dan terminologi proxy ..... 3735
Ikhtisar konsep RDS Proxy ..... 3736
Pengumpulan koneksi ..... 3737
Keamanan ..... 3738
Failover ..... 3740

Transaksi ..... 3741
Memulai dengan RDS Proxy ..... 3742
Siapkan jaringan proxy ..... 3742
Menyiapkan kredensial basis data ..... 3746
Mengkonfigurasi otentikasi IAM ..... 3750
Membuat proxy ..... 3754
Melihat proxy ..... 3763
Terhubung melalui Proksi RDS ..... 3765
Mengelola RDS Proxy ..... 3768
Mengubah Proksi RDS ..... 3769
Menambahkan pengguna basis data ..... 3776
Pertimbangan koneksi RDS Proxy ..... 3777
Hindari menyematkan Proxy RDS ..... 3781
Menghapus Proxy RDS ..... 3786
Bekerja dengan titik akhir Proksi RDS ..... 3787
Ikhtisar titik akhir proksi ..... 3788
Batasan untuk titik akhir proksi ..... 3789
Menggunakan titik akhir pembaca dengan klaster Aurora ..... 3790
Mengakses database  RDS di seluruh VPCs ..... 3794
Membuat titik akhir proksi ..... 3796
Melihat titik akhir proksi ..... 3798
Mengubah titik akhir proksi ..... 3800
Menghapus titik akhir proksi ..... 3801
Memantau Proxy RDS dengan CloudWatch ..... 3802
Bekerja dengan acara RDS Proxy ..... 3811
RDSAcara proxy ..... 3811
RDSContoh proxy ..... 3814
Memecahkan masalah Proksi RDS ..... 3817
Memverifikasi konektivitas untuk proksi ..... 3817
Masalah dan solusi umum ..... 3819
Pemecahan masalah untuk RDS untuk MySQL ..... 3820
Pemecahan masalah untuk RDS untuk PostgreSQL ..... 3822
Penggunaan Proksi RDS dengan AWS CloudFormation ..... 3827
Menggunakan Proksi RDS dengan basis data global Aurora ..... 3828
Batasan untuk Proksi RDS dengan basis data global ..... 3829
Cara kerja titik akhir Proksi RDS dengan basis data global ..... 3829

Integrasi nol-ETL ..... 3831
Manfaat ..... 3833
Konsep utama ..... 3833
Batasan ..... 3834
Batasan umum ..... 3834
Batasan Aurora MySQL ..... 3835
Keterbatasan Aurora PostgreSQL ..... 3836
Batasan Amazon Redshift ..... 3836
Amazon SageMakerketerbatasan lakehouse ..... 3836
Kuota ..... 3837
Wilayah yang Didukung ..... 3837
Mulai menggunakan integrasi nol-ETL ..... 3837
Langkah 1: Buat grup parameter klaster DB kustom ..... 3838
Langkah 2: Pilih atau buat cluster DB sumber ..... 3839
Langkah 3a: Buat gudang data target ..... 3840
Siapkan integrasi menggunakan AWS SDKs ..... 3841
Langkah 3b: Buat AWS Glue katalog untuk integrasi Amazon SageMaker nol-ETL ..... 3851
Langkah selanjutnya ..... 3854
Membuat integrasi Nol-ETL dengan Amazon Redshift ..... 3854
Prasyarat ..... 3855
Izin yang diperlukan ..... 3855
Membuat integrasi nol-ETL ..... 3858
Integrasi enkripsi ..... 3862
Langkah selanjutnya ..... 3864
Membuat integrasi nol-ETL dengan rumah danau Amazon SageMaker ..... 3865
Prasyarat ..... 3865
Izin yang diperlukan ..... 3865
Membuat integrasi nol-ETL dengan rumah danau Amazon SageMaker ..... 3868
Integrasi enkripsi ..... 3873
Langkah selanjutnya ..... 3875
Pemfilteran data untuk integrasi nol-ETL ..... 3875
Format filter data ..... 3876
Filter logika ..... 3878
Filter prioritas ..... 3879
..... 3879
Contoh PostgreSQL Aurora ..... 3880

Menambahkan filter data ... 3881
Menghapus filter data ... 3883
Menambahkan dan mengueri data ... 3883
Membuat database target ... 3884
Menambahkan data ke sumber DB cluster ... 3884
Menanyakan data Anda di Amazon Redshift ... 3886
Perbedaan jenis data ... 3887
Operasi DDL untuk Aurora PostgreSQL ... 3900
Melihat dan memantau integrasi nol-ETL ... 3903
Melihat integrasi ... 3903
Pemantauan menggunakan tabel sistem ... 3905
Pemantauan EventBridge dengan ... 3906
Memodifikasi integrasi nol-ETL ... 3906
Menghapus integrasi nol-ETL ... 3907
Memecahkan masalah integrasi nol-ETL ... 3908
Saya tidak dapat membuat integrasi nol-ETL ... 3909
Integrasi saya terjebak dalam keadaan `Syncing` ... 3910
Tabel saya tidak bereplikasi ke Amazon Redshift ... 3910
Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang ... 3911
Masalah gagal integrasi untuk integrasi Amazon SageMaker lakehouse zero-ETL ... 3914
Operasi DDL yang tidak lengkap muncul di Amazon Redshift ... 3915
Menggunakan Aurora Serverless v2 ... 3916
Aurora Serverless v2 kasus penggunaan ... 3916
Mengonversi beban kerja terprovisi ... 3919
Keuntungan dari Aurora Serverless v2 ... 3919
Bagaimana Aurora Serverless v2 cara kerja ... 3920
Gambaran umum ... 3921
Konfigurasi klaster ... 3923
Kapasitas ... 3923
Penskalaan ... 3926
Ketersediaan tinggi ... 3928
Penyimpanan ... 3929
Parameter konfigurasi ... 3930
Persyaratan dan batasan untuk Aurora Serverless v2 ... 3930
Wilayah dan ketersediaan versi ... 3931

Cluster yang menggunakan Aurora Serverless v2 harus memiliki rentang kapasitas yang ditentukan ..... 3931
Beberapa fitur yang disediakan tidak didukung di Aurora Serverless v2 ..... 3932
Beberapa Aurora Serverless v2 Aspek berbeda dari Aurora Serverless v1 ..... 3933
Membuat Aurora Serverless v2 klaster DB. ..... 3933
Pengaturan ..... 3933
Membuat Aurora Serverless v2 klaster DB. ..... 3935
Membuat Aurora Serverless v2 penulis ..... 3940
Mengelola Aurora Serverless v2 ..... 3941
Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster ..... 3941
Memeriksa Aurora Serverless v2 Rentang kapasitas ..... 3948
Menambahkan Aurora Serverless v2 pembaca ..... 3950
Konversi dari disediakan ke Aurora Serverless v2 ..... 3951
Konversi dari Aurora Serverless v2 untuk disediakan ..... 3952
Menjeda Aurora Serverless v2 Instans DB ..... 3953
Memilih tingkat promosi untuk Aurora Serverless v2 pembaca ..... 3953
Menggunakan TLS/SSL dengan Aurora Serverless v2 ..... 3955
Melihat Aurora Serverless v2 penulis dan pembaca ..... 3956
Penebangan untuk Aurora Serverless v2 ..... 3957
Performa dan penskalaan untuk Aurora Serverless v2 ..... 3964
Memilih rentang kapasitas ..... 3965
Menggunakan grup parameter untuk Aurora Serverless v2 ..... 3979
Menghindari out-of-memory kesalahan ..... 3985
CloudWatch Metrik penting ..... 3986
Memantau performa Aurora Serverless v2 dengan Wawasan Performa ..... 3991
Memecahkan masalah kapasitas Aurora Serverless v2 ..... 3991
Penskalaan ke 0 ACUs dengan jeda otomatis dan lanjutkan ..... 3993
Ikhtisar jeda otomatis ..... 3993
Prasyarat dan Batasan ..... 3995
Menghidupkan dan menonaktifkan jeda otomatis ..... 3995
Cara kerja jeda otomatis ..... 4000
Konfigurasi kluster jeda otomatis dan Aurora ..... 4004
Memantau jeda otomatis ..... 4007
Pemecahan masalah untuk jeda otomatis ..... 4011
Desain aplikasi untuk jeda otomatis ..... 4013
Migrasi ke Aurora Serverless v2 ..... 4014

Penggunaan Aurora Serverless v2 dengan cluster yang ada ..... 4015
Beralih dari klaster terprovisi ..... 4018
Perbandingan Aurora Serverless v2 and Aurora Serverless v1 ..... 4024
Upgrade dari Aurora Serverless v1 kepada Aurora Serverless v2 ..... 4035
Migrasi dari database lokal ke Aurora Serverless v2 ..... 4038
Penggunaan Aurora Serverless v1 ..... 4039
Ketersediaan wilayah dan versi untuk Aurora Serverless v1 ..... 4040
Keuntungan dari Aurora Serverless v1 ..... 4040
Kasus penggunaan untuk Aurora Serverless v1 ..... 4041
Pembatasan Aurora Serverless v1 ..... 4041
Persyaratan konfigurasi untuk Aurora Serverless v1 ..... 4044
Menggunakan TLS/SSL dengan Aurora Serverless v1 ..... 4044
Suite cipher yang didukung untuk koneksi ke Aurora Serverless v1 Klaster DB ..... 4048
Bagaimana Aurora Serverless v1 cara kerja ..... 4048
Aurora Serverless v1 arsitektur ..... 4049
Penskalaan otomatis ..... 4050
Tindakan batas waktu habis ..... 4051
Jeda dan lanjutkan ..... 4053
Menentukan max_connections ..... 4053
Grup parameter ..... 4056
Pencatatan log ..... 4059
Pemeliharaan ..... 4063
Failover ..... 4064
Snapshot ..... 4064
Membuat Aurora Serverless v1 klaster DB ..... 4065
Memulihkan Aurora Serverless v1 Klaster DB ..... 4069
Memodifikasi Aurora Serverless v1 Klaster DB ..... 4072
Memodifikasi konfigurasi penskalaan ..... 4073
Meningkatkan versi mayor ..... 4075
Konversi dari Aurora Serverless v1 untuk disediakan ..... 4077
Pertimbangan saat mengonversi dari Aurora Serverless v1 Cluster DB ke cluster yang disediakan ..... 4079
Penskalaan Aurora Serverless v1 kapasitas klaster DB secara manual ..... 4081
Melihat Aurora Serverless v1 klaster DB ..... 4083
Pemantauan Aurora Serverless v1 Cluster DB dengan CloudWatch ..... 4086
Menghapus sebuah Aurora Serverless v1 Klaster DB ..... 4087

Aurora Serverless v1 dan versi mesin basis data Aurora ..... 4090
Aurora MySQL Nirserver ..... 4091
Aurora PostgreSQL Nirserver ..... 4091
Peningkatan versi minor otomatis ..... 4092
Menggunakan Amazon RDS Data API ..... 4093
Ketersediaan wilayah dan versi untuk Amazon RDS Data API ..... 4094
Batasan ..... 4094
Data API Aurora Serverless v2 dibandingkan dengan Aurora Serverless v1 ..... 4095
Jumlah maksimum permintaan per detik ..... 4095
Mengaktifkan atau menonaktifkan Amazon RDS Data API pada database yang ada ..... 4096
CloudTrail acara ..... 4096
Dukungan multistatement ..... 4097
Permintaan bersamaan untuk ID transaksi yang sama ..... 4098
BatchExecuteStatement perilaku ..... 4099
Eksekusi perilaku QL ..... 4099
ExecuteStatement perilaku ..... 4100
Perilaku parameter skema ..... 4100
Autentikasi dan otorisasi ..... 4101
Otorisasi berbasis tag ..... 4102
Menyimpan kredensi dalam rahasia Secrets Manager ..... 4105
Mengaktifkan API Data ..... 4105
Mengaktifkan RDS Data API saat Anda membuat database ..... 4106
Mengaktifkan API Data RDS pada database yang ada ..... 4107
Membuat titik akhir Amazon VPC () PrivateLink ..... 4110
Memanggil Data API ..... 4113
Referensi operasi ..... 4114
Menelepon dari AWS CLI ..... 4118
Menelepon dari aplikasi Python ..... 4129
Menelepon dari aplikasi Java ..... 4133
Mengontrol perilaku batas waktu ..... 4138
Pustaka klien Java ..... 4140
Mengunduh pustaka klien Java untuk API Data ..... 4140
Contoh pustaka klien Java ..... 4140
Memproses hasil kueri di JSON ..... 4142
Mengambil hasil kueri dalam format JSON ..... 4143
Pemetaan Tipe Data ..... 4143

Pemecahan Masalah ..... 4144
Contoh ..... 4145
Pemecahan Masalah Data API ..... 4150
Transaksi <transaction_ID>tidak ditemukan ..... 4150
Paket untuk kueri terlalu besar ..... 4151
Respons basis data melebihi batas ukuran ..... 4151
HttpEndpointtidak diaktifkan untuk cluster <cluster_ID> ..... 4151
DatabaseErrorException: Transaksi masih menjalankan kueri ..... 4151
Pengecualian hasil yang tidak didukung ..... 4152
Multi-pernyataan tidak didukung ..... 4152
Parameter skema tidak didukung ..... 4152
Pencatatan panggilan API Data ..... 4153
Bekerja dengan informasi API Data di CloudTrail ..... 4153
Menyertakan dan mengecualikan peristiwa API Data dari jejak CloudTrail ..... 4154
Memahami entri file log API Data ..... 4157
Memantau kueri API Data ..... 4159
Bagaimana kueri API Data RDS direpresentasikan dalam Performance Insights ..... 4159
Menggunakan editor kueri ..... 4160
Ketersediaan editor kueri ..... 4160
Mengotorisasi akses ..... 4160
Menjalankan kueri ..... 4162
DBQMSAPIreferensi ..... 4166
CreateFavoriteQuery ..... 4167
CreateQueryHistory ..... 4167
CreateTab ..... 4167
DeleteFavoriteQueries ..... 4167
DeleteQueryHistory ..... 4167
DeleteTab ..... 4167
DescribeFavoriteQueries ..... 4167
DescribeQueryHistory ..... 4167
DescribeTabs ..... 4168
GetQueryString ..... 4168
UpdateFavoriteQuery ..... 4168
UpdateQueryHistory ..... 4168
UpdateTab ..... 4168
Menggunakan machine learning Aurora ..... 4169

Menggunakan pembelajaran mesin Aurora dengan Aurora My SQL ..... 4170
Persyaratan untuk menggunakan machine learning Aurora ..... 4171
Wilayah dan ketersediaan versi ..... 4172
Fitur yang didukung dan batasan ..... 4173
Menyiapkan klaster Aurora Anda untuk machine learning Aurora ..... 4174
Menggunakan Amazon Bedrock dengan cluster Aurora My DB SQL Anda ..... 4188
Menggunakan Amazon Comprehend dengan cluster Aurora My DB Anda SQL ..... 4190
Menggunakan SageMaker AI dengan cluster Aurora My SQL DB Anda ..... 4192
Pertimbangan performa ..... 4196
Pemantauan ..... 4198
Menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL ..... 4200
Persyaratan untuk menggunakan machine learning Aurora ..... 4200
Fitur yang didukung dan batasan ..... 4201
Menyiapkan klaster DB Aurora untuk menggunakan machine learning Aurora ..... 4202
Menggunakan Amazon Bedrock dengan cluster DB Aurora Postgre SQL Anda ..... 4215
Menggunakan Amazon Comprehend dengan cluster DB Aurora Postgre Anda SQL ..... 4218
Menggunakan SageMaker AI dengan cluster Aurora SQL Postgre DB Anda ..... 4220
Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan) ..... 4224
Pertimbangan performa ..... 4225
Pemantauan ..... 4231
Contoh kode ..... 4233
Hal-hal mendasar ..... 4244
Halo Aurora ..... 4245
Pelajari dasar-dasarnya ..... 4255
Tindakan ..... 4425
Skenario ..... 4629
Membuat API REST pustaka peminjaman ..... 4629
Buat pelacak butir kerja Aurora Nirserver ..... 4630
Praktik terbaik dengan Aurora ..... 4636
Pedoman operasional dasar untuk Amazon Aurora ..... 4636
Rekomendasi RAM instans DB ..... 4637
AWS driver basis data ..... 4638
Memantau Amazon Aurora ..... 4638
Menggunakan grup parameter DB dan grup parameter klaster DB ..... 4639
Video praktik terbaik Amazon Aurora ..... 4639
Melakukan pembuktian konsep Aurora ..... 4640

Gambaran umum bukti konsep Aurora ..... 4640
1. Identifikasi tujuan Anda ..... 4641
2. Pahami karakteristik beban kerja Anda ..... 4642
3. Berlatih dengan konsol atau CLI ..... 4643
Berlatih dengan konsol ..... 4643
Berlatihlah dengan AWS CLI ..... 4644
4. Buat klaster Aurora Anda ..... 4645
5. Siapkan skema Anda ..... 4646
6. Impor data Anda ..... 4647
7. Port SQL kode Anda ..... 4648
8. Tentukan pengaturan konfigurasi ..... 4649
9. Hubungkan ke Aurora ..... 4649
10. Jalankan beban kerja Anda ..... 4651
11. Ukur performa ..... 4652
12. Coba ketersediaan tinggi Aurora ..... 4655
13. Apa yang harus dilakukan selanjutnya ..... 4657
Keamanan ..... 4659
Autentikasi basis data ..... 4661
Autentikasi kata sandi ..... 4662
Autentikasi basis data IAM ..... 4663
Autentikasi Kerberos ..... 4663
Manajemen kata sandi dengan Aurora dan Secrets Manager ..... 4665
Wilayah dan ketersediaan versi ..... 4665
Batasan: ..... 4665
Gambaran Umum ..... 4666
Manfaat ..... 4666
Izin ..... 4667
Menegakkan manajemen ..... 4668
Mengelola kata sandi untuk ..... 4669
Rahasia berputar untuk cluster DB ..... 4673
Melihat detail rahasia untuk cluster DB ..... 4675
Perlindungan data ..... 4678
Enkripsi data ..... 4679
Privasi lalu lintas jaringan internet ..... 4709
Manajemen identitas dan akses ..... 4711
Audiens ..... 4711

Mengautentikasi dengan identitas ..... 4712
Mengelola akses menggunakan kebijakan ..... 4716
Bagaimana Amazon Aurora bekerja dengan IAM ..... 4718
Contoh kebijakan berbasis identitas ..... 4726
AWS kebijakan terkelola ..... 4748
Pembaruan kebijakan ..... 4755
Pencegahan "confused deputy" lintas layanan ..... 4766
Autentikasi basis data IAM ..... 4768
Pemecahan Masalah ..... 4819
Pencatatan log dan pemantauan ..... 4821
Validasi kepatuhan ..... 4824
Ketahanan ..... 4825
Pencadangan dan pemulihan ..... 4825
Replikasi ..... 4826
Failover ..... 4826
Keamanan infrastruktur ..... 4827
Grup keamanan ..... 4827
Aksesibilitas publik ..... 4827
Titik akhir VPC (AWS PrivateLink) ..... 4829
Pertimbangan ..... 1104
Ketersediaan ..... 1104
Membuat titik akhir VPC antarmuka ..... 1104
Membuat kebijakan titik akhir VPC ..... 1104
Praktik terbaik keamanan ..... 4832
Mengontrol akses dengan grup keamanan ..... 4834
Ikhtisar grup keamanan VPC ..... 4834
Skenario grup keamanan ..... 4835
Membuat grup keamanan VPC ..... 4836
Mengaitkan dengan klaster DB ..... 4837
Hak akses akun pengguna master ..... 4837
Peran terkait layanan ..... 4840
Izin peran terkait layanan untuk Amazon Aurora ..... 4840
Izin peran terkait layanan untuk Amazon RDS Beta ..... 4843
Peran terkait layanan untuk Amazon RDS Preview ..... 4844
Menggunakan Amazon Aurora dengan Amazon VPC ..... 4846
Bekerja dengan klaster DB dalam VPC ..... 4846

Skenario untuk mengakses cluster DB di VPC ..... 4862
Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 ..... 4869
Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda) ..... 4876
Kuota dan batasan ..... 4887
Kuota dalam Amazon Aurora ..... 4887
Batasan penamaan dalam Amazon Aurora ..... 4893
Batas ukuran Amazon Aurora ..... 4894
Pemecahan Masalah ..... 4896
Tidak dapat terhubung ke instans DB ..... 4896
Menguji koneksi instans DB ..... 4898
Memecahkan masalah autentikasi koneksi ..... 4899
Masalah keamanan ..... 4899
Pesan kesalahan “gagal mengambil atribut akun, fungsi konsol tertentu mungkin terganggu.” ..... 4900
Mengatur ulang kata sandi pemilik instans DB ..... 4900
Penghentian atau boot ulang instans DB ..... 4900
Perubahan parameter tidak diberlakukan ..... 4901
Masalah memori Aurora yang dapat dikosongkan ..... 4901
Masalah replikasi Aurora MySQL ..... 4903
Mendiagnosis dan mengatasi jeda di antara replika baca ..... 4903
Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL ..... 4905
Kesalahan replikasi terhenti ..... 4907
Replikasi replika baca gagal menginisialisasi struktur metadata ..... 4907
Referensi Amazon RDS API ..... 4909
Menggunakan API Kueri ..... 4909
Parameter Kueri ..... 4909
Autentikasi permintaan Kueri ..... 4910
Memecahkan masalah aplikasi ..... 4910
Mengambil kesalahan ..... 4910
Tips pemecahan masalah ..... 4911
Riwayat dokumen ..... 4912
AWS Glosarium ..... 5015
..... 5016

# Apa itu Amazon Aurora?

Amazon Aurora (Aurora) adalah mesin basis data relasional yang dikelola sepenuhnya dan kompatibel dengan MySQL dan PostgreSQL. Anda sudah tahu bagaimana MySQL dan PostgreSQL menggabungkan kecepatan dan keandalan basis data komersial kelas atas dengan kesederhanaan dan efektivitas biaya basis data sumber terbuka. Kode, alat, dan aplikasi yang Anda gunakan saat ini dengan basis data MySQL dan PostgreSQL Anda yang sudah ada dapat digunakan dengan Aurora. Dengan sejumlah beban kerja, Aurora dapat memberikan hingga lima kali throughput MySQL dan hingga tiga kali throughput PostgreSQL tanpa memerlukan perubahan pada sebagian besar aplikasi Anda saat ini.

Aurora mencakup subsistem penyimpanan performa tinggi. Mesin basis data-nya yang kompatibel dengan MySQL dan PostgreSQL disesuaikan untuk memanfaatkan penyimpanan terdistribusi cepat tersebut. Penyimpanan dasarnya meningkat secara otomatis sesuai kebutuhan. Volume klaster Aurora dapat bertambah ke ukuran maksimum 128 tebibyte (TiB). Aurora juga mengotomatiskan dan membakukan pengklasteran dan replikasi basis data, yang biasanya termasuk dalam aspek konfigurasi dan administrasi basis data yang paling menantang.

Aurora adalah bagian dari layanan basis data terkelola Amazon Relational Database Service (Amazon RDS). Amazon RDS memudahkan untuk mengatur, mengoperasikan, dan menskalakan database relasional di cloud. Jika Anda belum memahami tentang Amazon RDS, lihat Panduan pengguna Amazon Relational Database Service. Untuk mempelajari lebih lanjut tentang berbagai opsi basis data yang tersedia di Amazon Web Services, lihat Memilih basis data yang tepat untuk organisasi Anda pada AWS.

Topik


- Model tanggung jawab bersama Amazon RDS
- Cara kerja Amazon Aurora dengan Amazon RDS
- Klaster DB Amazon Aurora
- Versi Amazon Aurora
- Wilayah dan Zona Ketersediaan
- Fitur yang didukung di Amazon Aurora oleh dan mesin Wilayah AWS Aurora DB
- Koneksi titik akhir Amazon Aurora
- Kelas instans Amazon Aurora DB
- Penyimpanan Amazon Aurora

- Keandalan Amazon Aurora
- Keamanan Amazon Aurora
- Ketersediaan yang tinggi untuk Amazon Aurora
- Replikasi dengan Amazon Aurora
- DB untuk Aurora

## Model tanggung jawab bersama Amazon RDS

Amazon RDS bertanggung jawab untuk meng-host komponen perangkat lunak dan infrastruktur instans DB dan klaster DB. Anda bertanggung jawab untuk penyetelan kueri, yang merupakan proses menyesuaikan kueri SQL untuk meningkatkan performa. Performa kueri sangat bergantung pada desain basis data, ukuran data, distribusi data, beban kerja aplikasi, dan pola kueri, yang dapat sangat bervariasi. Pemantauan dan penyetelan adalah proses sangat khusus yang Anda miliki untuk basis data RDS Anda. Anda dapat menggunakan Wawasan Performa Amazon RDS dan alat lainnya untuk mengidentifikasi kueri bermasalah.

## Cara kerja Amazon Aurora dengan Amazon RDS

Poin-poin berikut mengilustrasikan bagaimana Amazon Aurora berkaitan dengan mesin standar MySQL dan PostgreSQL yang tersedia di Amazon RDS:

- Anda memilih Aurora MySQL atau Aurora PostgreSQL sebagai opsi mesin DB saat mengatur server basis data baru melalui Amazon RDS.
- Aurora memanfaatkan fitur Amazon Relational Database Service (Amazon RDS) yang sudah Anda ketahui untuk manajemen dan administrasi. Aurora menggunakan AWS Management Console antarmuka Amazon RDS, AWS CLI perintah, dan operasi API untuk menangani tugas database rutin seperti penyediaan, penambalan, pencadangan, pemulihan, deteksi kegagalan, dan perbaikan.
- Operasi manajemen Aurora biasanya melibatkan seluruh klaster server basis data yang disinkronkan melalui replikasi, bukan instans basis data secara individual. Pengklasteran, replikasi, dan alokasi penyimpanan otomatis membuatnya sederhana dan hemat biaya untuk mengatur, mengoperasikan, dan menskalakan deployment MySQL dan PostgreSQL terbesar Anda.
- Anda dapat membawa data dari Amazon RDS for MySQL dan Amazon RDS for PostgreSQL ke Aurora dengan membuat dan memulihkan snapshot, atau dengan mengatur replikasi satu arah.

Anda dapat menggunakan alat migrasi tombol-tekan untuk mengonversi aplikasi RDS for MySQL dan RDS for PostgreSQL Anda saat ini ke Aurora.

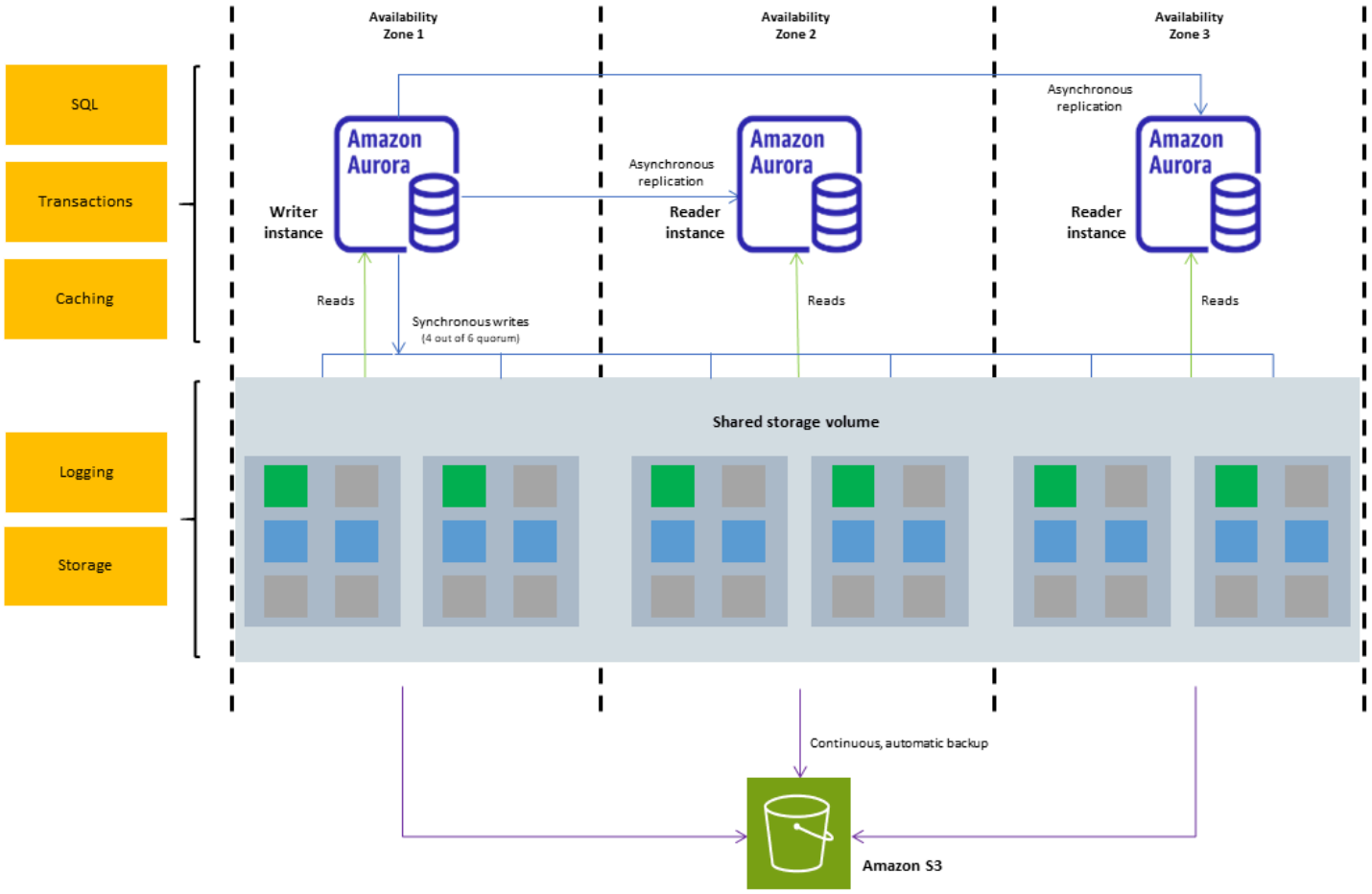
Sebelum menggunakan Amazon Aurora, selesaikan langkah-langkah dalam Menyiapkan lingkungan Anda untuk Amazon Aurora, lalu tinjau konsep dan fitur Aurora di Klaster DB Amazon Aurora.

# Klaster DB Amazon Aurora

Klaster DB Amazon Aurora terdiri dari satu atau beberapa instans DB dan sebuah volume klaster yang mengelola data untuk instans DB tersebut. Volume klaster Aurora adalah volume penyimpanan basis data virtual yang mencakup beberapa Zona Ketersediaan, dengan masing-masing Zona Ketersediaan memiliki salinan data klaster DB. Ada dua jenis instans DB yang membentuk klaster DB Aurora:

- Instans DB primer (penulis) — Mendukung operasi baca dan tulis, dan melakukan semua modifikasi data pada volume cluster. Setiap klaster DB Aurora memiliki satu instans DB primer.
- Aurora Replica (instans DB pembaca) - Menghubungkan ke volume penyimpanan yang sama dengan instans DB utama tetapi hanya mendukung operasi baca. Setiap klaster DB Aurora dapat memiliki hingga 15 Replika Aurora di samping instans DB primer. Ketersediaan tinggi dipertahankan dengan menempatkan Replika Aurora di Zona Ketersediaan terpisah. Aurora secara otomatis melakukan failover ke sebuah Replika Aurora jika instans DB primer tidak tersedia. Anda dapat menentukan prioritas failover untuk Replika Aurora. Replika Aurora juga dapat mengalihkan beban kerja baca dari instans DB primer.

Diagram berikut menggambarkan hubungan antara volume cluster, instance DB penulis, dan instance DB pembaca dalam cluster Aurora DB.




> **Note**
>
> Informasi sebelumnya berlaku untuk semua cluster Aurora DB — yang disediakan, query paralel, Aurora Global Database, Aurora MySQL kompatibel, dan Aurora PostgreSQL kompatibel. Aurora Serverless

Cluster Aurora DB menggambarkan pemisahan kapasitas komputasi dan penyimpanan. Misalnya, konfigurasi Aurora dengan hanya satu instance DB masih merupakan cluster, karena volume penyimpanan yang mendasarinya melibatkan beberapa node penyimpanan yang didistribusikan di beberapa Availability Zones ()AZs.

Input/output (I/O) operasi di cluster Aurora DB dihitung dengan cara yang sama, terlepas dari apakah mereka menggunakan instans DB penulis atau pembaca. Lihat informasi yang lebih lengkap di Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

# Versi Amazon Aurora

Dengan Amazon Aurora, Anda dapat memilih mesin database relasional yang didukung yang paling sesuai dengan kebutuhan aplikasi Anda sambil mempertahankan kompatibilitas dengan mesin yang mendasarinya. Aurora menggunakan kembali kode mesin database dari mesin yang didukung, sehingga Anda dapat memanfaatkan keterampilan, alat, dan perpustakaan yang ada untuk mesin tersebut. Saat membuat cluster, Anda menentukan versi mesin database Amazon Aurora yang ingin Anda gunakan. Versi yang Anda pilih menentukan kompatibilitas dan fitur yang tersedia.

Dokumentasi ini menjelaskan poin umum dan perbedaan antara Amazon Aurora dan mesin database yang sesuai. Dengan informasi ini, Anda dapat menentukan versi perangkat lunak yang harus Anda pilih dan cara memverifikasi fitur dan perbaikan bug yang tersedia di setiap versi. Anda juga dapat menggunakan referensi ini untuk menentukan irama dan rencana peningkatan yang sesuai untuk peningkatan Anda.

## Mesin basis data yang didukung untuk kluster basis data Amazon Aurora

Database relasional berikut tersedia di Amazon Aurora. Aurora menggunakan kembali kode dan mempertahankan kompatibilitas dengan mesin DB yang mendasarinya. Namun, Aurora memiliki nomor versi unik, siklus rilis, dan garis waktu untuk penghentian versi. Setiap versi Aurora baru dilengkapi dengan catatan rilis yang mencantumkan fitur baru, perbaikan, dan perubahan serta penyempurnaan lainnya yang berlaku untuk setiap versi.

| Basis data Aurora | Panduan pengguna | Versi yang tersedia | Catatan rilis |
|---|---|---|---|
| Edisi yang Kompatibel dengan Amazon Aurora MySQL | Menggunakan Amazon Aurora MySQL | Pembaruan mesin database untuk Amazon Aurora My SQL | Catatan Rilis untuk Aurora MySQL |
| Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL | Menggunakan Amazon Aurora PostgreSQL | Pembaruan mesin database untuk Amazon Aurora Postgre SQL | Catatan Rilis untuk Aurora PostgreSQL |

# Menentukan versi basis data Amazon Aurora untuk klaster basis data Anda

Saat Anda membuat cluster DB baru dengan operasi Buat database di atau AWS CLI, AWS Management Console atau dengan operasi `CreateDBCluster` API, Anda dapat menentukan versi database Aurora yang tersedia saat ini (mayor atau minor).

Untuk mempelajari cara membuat klaster Aurora, lihat Membuat klaster DB Amazon Aurora. Untuk mempelajari cara mengubah versi klaster Aurora yang ada, lihat Memodifikasi klaster DB Amazon Aurora.

> **Note**
>
> Tidak semua versi database Aurora tersedia di setiap. Wilayah AWS Untuk mempelajari lebih lanjut tentang Wilayah dan versi yang tersedia di masing-masing wilayah Wilayah AWS, lihat Wilayah dan Zona Ketersediaan danDaerah yang Didukung dan mesin DB untuk database global Aurora.

## Versi Amazon Aurora

Versi Amazon Aurora berbeda dari database komunitas hulu yang kompatibel dengannya. Untuk membantu Anda menjaga kompatibilitas aplikasi dan memanfaatkan fitur mesin DB terbaru, bagian berikut menjelaskan konvensi versi Aurora dan bagaimana versi Aurora memetakan ke basis data komunitas masing-masing.

Untuk daftar database relasional yang tersedia di Amazon Aurora, lihat. Mesin basis data yang didukung untuk kluster basis data Amazon Aurora

### Perbedaan nomor versi antara basis data komunitas dan Aurora

Setiap versi Amazon Aurora kompatibel dengan versi spesifik dari basis data komunitas yang sesuai. Anda dapat menemukan versi komunitas database Anda dengan `version` fungsi, dan versi Aurora dengan fungsi tersebut`aurora_version`.

Contoh berikut menunjukkan cara menemukan versi komunitas database Anda untuk Aurora MySQL dan Aurora PostgreSQL.

Aurora MySQL

`version`Fungsi mengembalikan versi komunitas database Anda untuk Aurora MySQL.

```
mysql> select version();
```

Contoh keluaran:

```
+------------------+
|   version()      |
+------------------+
|  8.0.32          |
+------------------+
```

Dan `aurora_version` fungsi mengembalikan versi Aurora:

```
mysql> select aurora_version(), @@aurora_version;
```

Contoh keluaran:

```
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Aurora PostgreSQL

`version`Fungsi mengembalikan versi komunitas database Anda untuk Aurora PostgreSQL.

```
postgres=> select version();
```

Contoh keluaran:

```
-----------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)
```

Dan `aurora_version` fungsi mengembalikan versi Aurora:

```
postgres=> select aurora_version();
```

Contoh keluaran:

```
aurora_version
----------------
3.2.2
```

Untuk informasi selengkapnya, lihat Memeriksa Aurora Versi saya menggunakan SQL SQL dan Mengidentifikasi versi Amazon Aurora Postgre SQL.

## Versi Amazon Aurora default

Versi default adalah versi yang dipilih Aurora secara otomatis untuk pembuatan atau peningkatan basis data ketika Anda tidak menentukan versi mesin target secara manual. Misalnya, perintah berikut menunjukkan versi mesin default untuk Aurora PostgreSQL (sampel output termasuk).

```
aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text

16.4
```

Setiap versi utama memiliki versi minor default yang sesuai. Dengan demikian, versi minor default adalah 16. n untuk Aurora PostgreSQL 16, dengan nomor versi n berubah ketika Aurora merilis versi minor default baru. Biasanya, Aurora merilis dua versi minor default untuk setiap versi utama per tahun. Skrip bash shell berikut menunjukkan versi minor default untuk satu set Aurora PostgreSQL versi utama (contoh output termasuk).

```
for major in 16 15 14 13 12 11; do
  echo -n "Default for Aurora PostgreSQL major version $major: "
  aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --engine-version "$major" \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text
done

Default for Aurora PostgreSQL major version 16: 16.4
```

```
Default for Aurora PostgreSQL major version 15: 15.8
Default for Aurora PostgreSQL major version 14: 14.13
Default for Aurora PostgreSQL major version 13: 13.16
Default for Aurora PostgreSQL major version 12: 12.20
Default for Aurora PostgreSQL major version 11: 11.21
```

Jika Anda mengaktifkan upgrade versi minor otomatis untuk cluster Aurora DB Anda, Aurora menggunakan versi minor default atau versi minor yang lebih baru untuk versi mayor tertentu. Misalnya, jika versi minor default untuk Aurora PostgreSQL 15 adalah 15.8, dan versi 15.10 yang lebih baru juga tersedia, Aurora dapat secara otomatis meningkatkan ke 15.8 atau 15.10.

## Versi mayor Amazon Aurora

Versi Aurora menggunakan skema *major*.*minor*.*patch*. Versi mayor Aurora mengacu pada versi mayor komunitas MySQL atau PostgreSQL yang kompatibel dengan Aurora. Versi mayor Aurora MySQL dan Aurora PostgreSQL tersedia berdasarkan dukungan standar setidaknya sampai akhir masa penggunaan komunitas untuk versi komunitas yang sesuai. Anda dapat terus menjalankan versi utama melewati akhir Aurora dari tanggal dukungan standar dengan biaya tertentu. Untuk informasi selengkapnya, lihat Amazon RDS Extended Support dengan  dan Harga Amazon Aurora.

Untuk informasi selengkapnya tentang versi utama dan kalender rilis untuk Aurora MySQL dan Aurora PostgreSQL, lihat halaman berikut di Catatan Rilis masing-masing:

- Kalender rilis untuk Aurora MySQL versi utama
- Kalender rilis untuk Aurora PostgreSQL versi utama

Anda juga dapat melihat informasi tentang tanggal dukungan untuk versi mesin utama dengan menjalankan AWS CLI perintah describe-db-major-engine-versions atau dengan menggunakan operasi Describe DBMajor EngineVersions RDS API.

> **Note**
>
> Amazon RDS Extended Support untuk Aurora MySQL versi 2 dimulai pada 1 November 2024, tetapi Anda tidak akan dikenakan biaya hingga 1 Desember 2024. Antara 1 November dan 30 November 2024, semua cluster DB Aurora MySQL versi 2 tercakup dalam Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas Amazon RDS untuk versi Aurora tertentu.

### Berapa lama versi mayor Amazon Aurora tetap tersedia

Amazon Aurora versi mayor tetap tersedia setidaknya sampai akhir masa penggunaan komunitas untuk versi komunitas yang sesuai. Anda dapat menggunakan tanggal akhir dukungan standar Aurora untuk merencanakan siklus pengujian dan peningkatan Anda. Tanggal ini merepresentasikan tanggal paling awal saat peningkatan ke versi yang lebih baru mungkin diperlukan. Untuk informasi selengkapnya tentang tanggal, lihat Versi mayor Amazon Aurora.

Sebelum Aurora meminta Anda untuk meningkatkan ke versi utama yang lebih baru dan untuk membantu Anda merencanakan, Anda menerima pengingat setidaknya 12 bulan sebelumnya. Pengingat mengkomunikasikan hal berikut tentang proses peningkatan.

- Waktu tonggak tertentu
- Dampaknya pada cluster DB Anda
- Tindakan yang disarankan

Kami menyarankan Anda menguji aplikasi Anda secara menyeluruh dengan versi database baru sebelum memutakhirkan klaster Anda ke versi utama yang baru.

Setelah versi utama mencapai akhir Aurora dari dukungan standar, cluster DB apa pun yang masih menjalankan versi sebelumnya secara otomatis ditingkatkan ke versi Extended Support selama jendela pemeliharaan terjadwal. Biaya Extended Support mungkin berlaku. Untuk informasi selengkapnya tentang Amazon RDS Extended Support, lihat Menggunakan dukungan Amazon RDS Extended.

## Versi minor Amazon Aurora

Versi Aurora menggunakan skema *major*.*minor*.*patch*. Versi minor Aurora menyediakan peningkatan inkremental khusus komunitas dan Aurora pada layanan ini, misalnya fitur dan perbaikan baru.

Untuk informasi selengkapnya tentang versi minor dan kalender rilis untuk Aurora MySQL dan Aurora PostgreSQL, lihat halaman berikut di Catatan Rilis masing-masing:

- Kalender rilis untuk Aurora MySQL versi minor
- Kalender rilis untuk Aurora PostgreSQL versi minor

Bagian berikut menjelaskan detail tentang irama dan masa pakai yang dapat Anda harapkan untuk versi minor Aurora.

Topik


- Seberapa sering Amazon Aurora versi minor dirilis
- Berapa lama versi minor Amazon Aurora tetap tersedia


### Seberapa sering Amazon Aurora versi minor dirilis

Secara umum, Amazon Aurora versi minor dirilis setiap tiga bulan. Jadwal rilis mungkin bervariasi untuk menyertakan fitur atau perbaikan tambahan.

### Berapa lama versi minor Amazon Aurora tetap tersedia

Biasanya, Amazon Aurora membuat setiap versi minor dari versi utama tertentu tersedia setidaknya selama 12 bulan. Pada akhir periode ini, Aurora mungkin secara otomatis meningkatkan database Anda ke versi minor default atau ke versi yang lebih baru. Aurora memulai upgrade selama jendela pemeliharaan terjadwal untuk setiap cluster DB yang menjalankan versi minor sebelumnya.

Dalam beberapa kasus, Aurora mungkin menggantikan versi minor dari versi mayor tertentu lebih cepat dari periode 12 bulan biasa. Alasan dapat mencakup masalah keamanan kritis atau end-of-support tanggal untuk versi utama.

Sebelum memulai upgrade otomatis versi minor yang mendekati akhir masa pakai, Aurora biasanya memberikan pengingat tiga bulan sebelumnya. Aurora merinci hal-hal berikut tentang proses peningkatan.

- Waktu tonggak tertentu
- Dampaknya pada cluster DB Anda
- Tindakan yang disarankan

Pemberitahuan dengan pemberitahuan kurang dari tiga bulan menjelaskan hal-hal penting, seperti masalah keamanan, yang memerlukan tindakan lebih cepat.

Jika pengaturan pemutakhiran versi minor otomatis diaktifkan, Anda mendapatkan pengingat tetapi tidak ada pemberitahuan peristiwa RDS. Aurora meningkatkan database Anda dalam jendela pemeliharaan setelah batas waktu upgrade wajib telah berlalu.

Jika pengaturan pemutakhiran versi minor otomatis tidak diaktifkan, Anda mendapatkan pengingat dan peristiwa klaster Amazon RDS DB dengan kategori `maintenance` dan ID. `RDS-EVENT-0156` Aurora meningkatkan database Anda dalam jendela pemeliharaan berikutnya.

Perhatikan bahwa setelah versi minor mencapai akhir Aurora dari dukungan standar, tidak ada versi patch lebih lanjut yang akan dirilis untuk versi minor tersebut. Untuk menerima perbaikan bug kritis atau CVEs, Anda harus meningkatkan ke versi minor dengan dukungan standar.

Untuk informasi selengkapnya tentang upgrade versi minor otomatis, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

### Versi patch Amazon Aurora

Versi Aurora menggunakan skema *`major`*`.`*`minor`*`.`*`patch`*. Versi patch Aurora mencakup perbaikan penting yang ditambahkan ke versi minor setelah rilis awal (misalnya, Aurora MySQL 3.04.0, 3.04.1,..., 3.04.3). Sementara setiap versi minor baru menyediakan fitur Aurora baru, versi patch baru dalam versi minor tertentu terutama digunakan untuk menyelesaikan masalah penting.

Untuk informasi selengkapnya tentang patching, lihat Memelihara klaster DB Amazon Aurora.

## Meningkatkan kluster Amazon Aurora DB

Dengan Amazon Aurora, Anda dapat mengontrol dan menguji peningkatan ke cluster DB Anda. Amazon Aurora menyediakan opsi untuk upgrade versi minor otomatis, kontrol upgrade manual, upgrade yang diperlukan, dan pengujian pra-upgrade. Anda dapat menyimpan klaster Anda up-to-date dengan versi minor terbaru, menunda peningkatan non-kritis, memaksa peningkatan untuk masalah serius, dan memvalidasi perilaku pemutakhiran di lingkungan non-produksi. Bagian berikut merinci cara mengelola dan menguji upgrade cluster Aurora DB menggunakan kemampuan ini.

### Upgrade versi minor otomatis untuk Aurora

Upgrade versi minor otomatis memperbarui database Anda secara berkala ke versi mesin database terbaru. Namun, upgrade mungkin tidak selalu menyertakan versi mesin database terbaru. Jika Anda perlu menyimpan database Anda pada versi tertentu pada waktu tertentu, kami sarankan Anda meningkatkan secara manual ke versi database yang Anda butuhkan sesuai dengan jadwal yang Anda butuhkan. Dalam kasus masalah keamanan kritis atau ketika versi mencapai end-of-support tanggalnya, Aurora mungkin menerapkan pemutakhiran versi minor meskipun Anda belum mengaktifkan opsi pemutakhiran versi minor Otomatis. Untuk informasi selengkapnya, lihat dokumentasi pemutakhiran untuk mesin database spesifik Anda.

Lihat Memutakhirkan versi minor atau level patch cluster Aurora SQL My DB dan Melakukan upgrade versi minor.

Anda dapat terus menggunakan versi minor Aurora terbaru dengan mengaktifkan Peningkatan versi minor otomatis untuk setiap instans DB di klaster Aurora. Aurora melakukan pemutakhiran otomatis hanya jika semua instans DB di cluster Anda mengaktifkan pengaturan ini.

Jika pemutakhiran versi minor otomatis adalah Ya untuk cluster DB Anda, Aurora memutakhirkan secara otomatis ke versi minor default atau ke versi minor yang lebih baru. Misalnya, jika versi minor default adalah 15.8 untuk Aurora PostgreSQL 15, dan versi 15.10 ada, target untuk upgrade otomatis dapat berupa 15.8 atau 15.10.

Aurora biasanya menjadwalkan upgrade otomatis dua kali setahun untuk cluster DB yang memiliki upgrade versi minor otomatis diaktifkan. Upgrade ini terjadi selama jendela pemeliharaan yang Anda tentukan untuk klaster Anda. Untuk informasi selengkapnya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Peningkatan versi minor otomatis dikomunikasikan terlebih dahulu melalui peristiwa klaster DB Amazon RDS dengan kategori `maintenance` dan ID `RDS-EVENT-0156`. Untuk informasi selengkapnya, lihat Kategori acara Amazon RDS dan pesan acara untuk Aurora.

## Mengontrol peningkatan cluster DB Anda secara manual ke versi baru

Jika Anda mengaktifkan pengaturan pemutakhiran versi minor Otomatis, Aurora secara otomatis memutakhirkan cluster DB Anda ke versi minor default atau versi minor yang lebih baru. Aurora biasanya menjadwalkan upgrade otomatis dua kali setahun untuk cluster DB yang memiliki pengaturan upgrade versi minor Auto diaktifkan. Peningkatan ini dimulai selama periode pemeliharaan yang ditentukan pelanggan.

Untuk mematikan upgrade versi minor otomatis, nonaktifkan upgrade versi minor otomatis pada instans DB apa pun dalam cluster Aurora. Aurora melakukan upgrade versi minor otomatis hanya jika semua instans DB di cluster Anda mengaktifkan pengaturan.

> **Note**
>
> Untuk peningkatan wajib seperti akhir masa pakai versi minor, Aurora meningkatkan cluster DB meskipun pengaturan peningkatan versi minor Otomatis dinonaktifkan. Anda mendapatkan pengingat tetapi tidak ada pemberitahuan acara RDS. Aurora upgrade cluster Anda terjadi dalam jendela pemeliharaan setelah batas waktu upgrade wajib telah berlalu.

Karena peningkatan versi mayor menimbulkan beberapa risiko kompatibilitas, tingkatkan ini tidak terjadi secara otomatis. Anda harus memulai ini, kecuali jika ada penghentian versi utama. Kami menyarankan Anda menguji aplikasi Anda secara menyeluruh dengan versi database baru sebelum memutakhirkan klaster Anda ke versi utama.

Untuk informasi selengkapnya tentang meningkatkan klaster DB ke versi mayor Aurora baru, lihat Memutakhirkan Amazon Aurora SQL My DB cluster dan Meningkatkan klaster DB Amazon Aurora PostgreSQL.

### Peningkatan Amazon Aurora wajib

Untuk perbaikan kritis tertentu, Aurora mungkin melakukan upgrade terkelola ke level patch yang lebih baru dalam versi minor yang sama. Dalam hal ini, Aurora meningkatkan klaster Anda bahkan jika upgrade versi minor otomatis dimatikan. Sebelum melakukannya, Aurora mengkomunikasikan proses peningkatan terperinci. Detailnya mencakup waktu pencapaian tertentu, dampak pada cluster DB Anda, dan tindakan yang disarankan. Upgrade terkelola tersebut terjadi secara otomatis dalam jendela pemeliharaan cluster.

### Menguji klaster DB Anda dengan versi Aurora baru sebelum melakukan peningkatan

Anda dapat menguji proses peningkatan dan fungsi versi baru dengan aplikasi dan beban kerja Anda. Pilih salah satu metode berikut:

- Kloning klaster Anda menggunakan fitur klon basis data cepat Amazon Aurora. Lakukan peningkatan dan pengujian pasca-peningkatan pada klaster baru.
- Pulihkan dari snapshot klaster untuk membuat klaster Aurora baru. Anda dapat membuat snapshot klaster sendiri dari klaster Aurora yang ada. Aurora juga secara otomatis membuat snapshot berkala bagi Anda untuk setiap klaster Anda. Anda kemudian dapat memulai peningkatan versi untuk klaster baru tersebut. Anda dapat bereksperimen pada salinan klaster yang ditingkatkan sebelum memutuskan apakah akan meningkatkan klaster asli Anda.

Untuk informasi selengkapnya tentang berbagai cara membuat klaster baru untuk pengujian ini, lihat Mengkloning volume untuk klaster DB Amazon Aurora dan Membuat snapshot klaster DB.

## Dukungan versi Amazon Aurora

Jika Amazon Aurora DB Anda memiliki dependensi kompleks pada perilaku mesin database tertentu, sebaiknya Anda melakukan pengujian ekstensif sebelum meningkatkan ke versi mesin database

yang lebih baru. Ada opsi dukungan jangka panjang sehingga Anda dapat mempertahankan cluster DB Anda pada versi mesin Aurora tertentu bahkan setelah digantikan oleh versi yang lebih baru. Bagian berikut menjelaskan opsi dukungan jangka panjang untuk cluster Aurora DB Anda.

## Dukungan jangka panjang untuk versi minor Amazon Aurora tertentu

Untuk setiap versi utama Aurora, versi minor tertentu ditetapkan sebagai versi long-term-support (LTS), dan tersedia setidaknya selama tiga tahun. Artinya, setidaknya satu versi minor per versi mayor tersedia lebih lama dari 12 bulan yang biasa. Biasanya, Aurora mengingatkan Anda enam bulan sebelum akhir periode ini. Aurora mengkomunikasikan hal berikut tentang proses upgrade.

- Waktu tonggak tertentu
- Dampaknya pada cluster DB Anda
- Tindakan yang disarankan

Pemberitahuan dengan pemberitahuan kurang dari enam bulan mengkomunikasikan hal-hal penting, seperti masalah keamanan, yang memerlukan tindakan lebih cepat.

Versi minor LTS hanya mencakup perbaikan kritis (melalui versi patch). Versi LTS tidak menyertakan fitur baru yang dirilis setelah diperkenalkan. Setahun sekali, klaster DB yang berjalan pada versi minor LTS di-patch ke versi patch terbaru untuk rilis LTS. Aurora menambal cluster Anda sehingga Anda mendapat manfaat dari perbaikan keamanan dan stabilitas kumulatif. Jika ada perbaikan penting, Aurora mungkin menambal versi minor LTS lebih sering, seperti untuk keamanan, yang perlu diterapkan.

> **Note**
>
> Jika Anda ingin tetap menggunakan versi minor LTS selama siklus hidupnya, pastikan untuk menonaktifkan upgrade versi minor otomatis untuk instans DB Anda. Untuk menghindari memutakhirkan cluster DB Anda secara otomatis dari versi minor LTS, kosongkan kotak centang Aktifkan peningkatan versi minor otomatis pada instans DB apa pun di cluster Aurora Anda.

Untuk nomor versi untuk semua Aurora versi LTS, lihat Aurora Dukungan SQL jangka panjang saya () rilis LTS dan Menggunakan rilis dukungan jangka panjang (LTS) Aurora PostgreSQL.

## Dukungan yang Diperluas Amazon RDS untuk versi Aurora tertentu

Dengan Amazon RDS Extended Support, Anda dapat terus menjalankan database Anda pada versi mesin utama melewati akhir Aurora dari tanggal dukungan standar dengan biaya tambahan. Selama RDS Extended Support, Amazon RDS akan memasok patch untuk Critical and High CVEs seperti yang didefinisikan oleh peringkat tingkat keparahan CVSS National Vulnerability Database (NVD). Untuk informasi selengkapnya, lihat Amazon RDS Extended Support dengan .

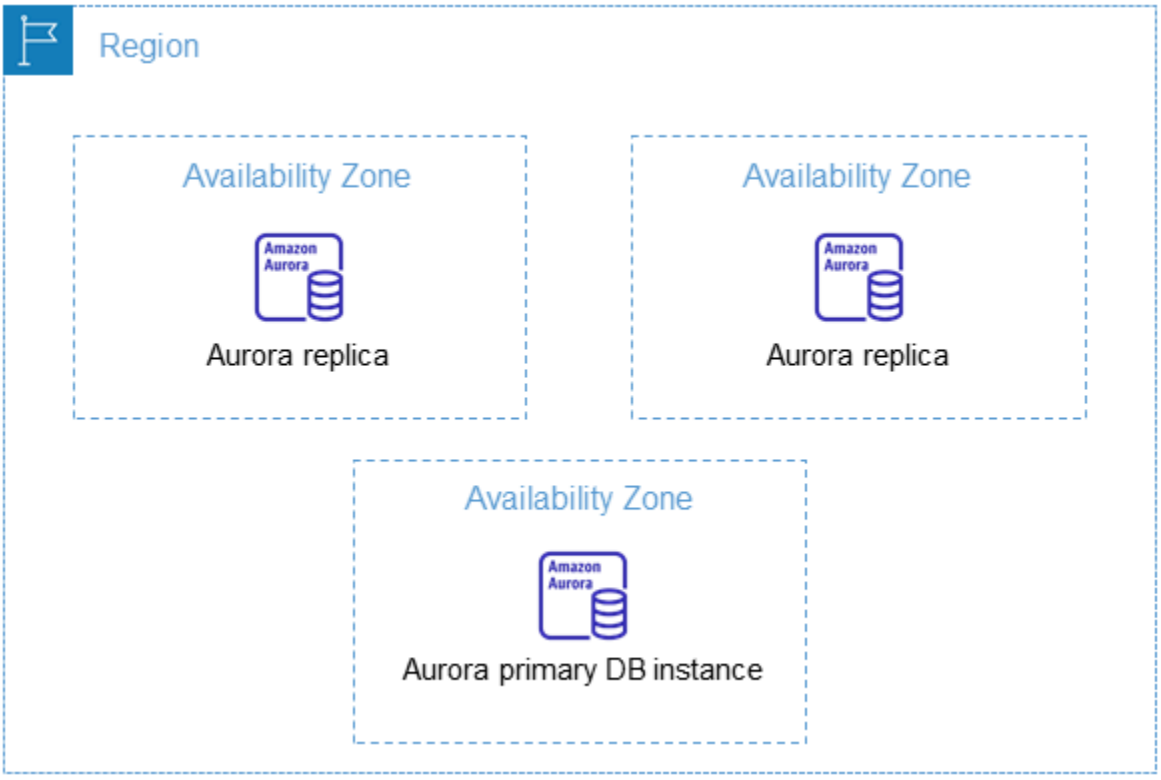
RDS Extended Support hanya tersedia pada versi Aurora tertentu. Lihat informasi yang lebih lengkap di Versi mayor Amazon Aurora.

# Wilayah dan Zona Ketersediaan

Sumber daya komputasi cloud Amazon di-hosting di beberapa lokasi di seluruh dunia. Lokasi ini terdiri dari AWS Wilayah dan Zona Ketersediaan. Setiap Wilayah AWS  adalah wilayah geografis yang terpisah. Setiap AWS Wilayah memiliki beberapa lokasi terisolasi yang dikenal sebagai Availability Zone.

> **Note**
>
> Untuk informasi tentang menemukan Availability Zone untuk suatu AWS Wilayah, lihat Menjelaskan Availability Zone Anda di EC2 dokumentasi Amazon.

Amazon beroperasi state-of-the-art, pusat data yang sangat tersedia. Meskipun jarang, kegagalan yang memengaruhi ketersediaan instans DB yang berada di lokasi yang sama dapat terjadi. Jika Anda meng-hosting semua instans DB di satu lokasi yang terpengaruh oleh kegagalan tersebut, tidak satu pun instans DB Anda akan tersedia.




Penting untuk diingat bahwa setiap AWS Wilayah sepenuhnya independen. Aktivitas Amazon RDS apa pun yang Anda lakukan (misalnya, membuat instance database atau mencantumkan instance

database yang tersedia) hanya berjalan di Wilayah default Anda saat ini. AWS AWS Wilayah default dapat diubah di konsol, atau dengan mengatur variabel `AWS_DEFAULT_REGION`lingkungan. Atau dapat diganti dengan menggunakan `--region` parameter dengan (). AWS Command Line Interface AWS CLI Untuk informasi selengkapnya, lihat Mengonfigurasi AWS Command Line Interface, khususnya bagian tentang variabel lingkungan dan opsi baris perintah.

Amazon RDS mendukung AWS Wilayah khusus yang disebut AWS GovCloud (US). Hal ini dirancang untuk memungkinkan lembaga pemerintah AS dan pelanggan memindahkan beban kerja yang lebih sensitif ke cloud. Wilayah AWS GovCloud (US) memenuhi persyaratan peraturan dan kepatuhan khusus pemerintah AS. Untuk informasi lebih lanjut, lihat Apa itu AWS GovCloud (US)?

Untuk membuat atau bekerja dengan instans Amazon RDS DB di AWS Wilayah tertentu, gunakan titik akhir layanan regional yang sesuai.

> **Note**
>
> Aurora tidak mendukung Zona Lokal.

## AWS Daerah

Setiap AWS Wilayah dirancang untuk diisolasi dari AWS Wilayah lain. Rancangan ini mencapai toleransi kesalahan dan stabilitas sebesar mungkin.

Saat melihat sumber daya, Anda hanya melihat sumber daya yang terkait dengan AWS Wilayah yang Anda tentukan. Ini karena AWS Wilayah terisolasi satu sama lain, dan kami tidak secara otomatis mereplikasi sumber daya di seluruh AWS Wilayah.

### Ketersediaan wilayah

Ketika menggunakan klaster DB Aurora dengan operasi antarmuka baris perintah atau API, pastikan bahwa Anda menentukan titik akhir regional.

Topik

- Ketersediaan Wilayah Aurora MySQL
- Ketersediaan Wilayah Aurora PostgreSQL

### Ketersediaan Wilayah Aurora MySQL

Tabel berikut menunjukkan AWS Wilayah di mana Aurora MySQL saat ini tersedia dan titik akhir untuk setiap Wilayah.

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| AS Timur (Ohio) | us–east-2 | rds.us-east-2.amazonaws.com | HTTPS | |
| AS Timur (Virginia Utara) | us-east-1 | rds.us-east-1.amazonaws.com | HTTPS | |
| AS Barat (California Utara) | us-west-1 | rds.us-west-1.amazonaws.com | HTTPS | |
| AS Barat (Oregon) | us-west-2 | rds.us-west-2.amazonaws.com | HTTPS | |
| Afrika (Cape Town) | af-south-1 | rds.af-south-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Hong Kong) | ap-east-1 | rds.ap-east-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Hyderabad) | ap-south-2 | rds.ap-south-2.amazonaws.com | HTTPS | |
| Asia Pasifik (Jakarta) | ap-southeast-3 | rds.ap-southeast-3.amazonaws.com | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| Asia Pasifik (Malaysia) | ap-southeast-5 | rds.ap-southeast-5.amazonaws.com | HTTPS | |
| Asia Pasifik (Melbourne) | ap-southeast-4 | rds.ap-southeast-4.amazonaws.com | HTTPS | |
| Asia Pasifik (Mumbai) | ap-south-1 | rds.ap-south-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Osaka) | ap-northeast-3 | rds.ap-northeast-3.amazonaws.com | HTTPS | |
| Asia Pasifik (Seoul) | ap-northeast-2 | rds.ap-northeast-2.amazonaws.com | HTTPS | |
| Asia Pasifik (Singapura) | ap-southeast-1 | rds.ap-southeast-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Sydney) | ap-southeast-2 | rds.ap-southeast-2.amazonaws.com | HTTPS | |
| Asia Pasifik (Taipei) | ap-timur-2 | rds.ap-east-2.amazonaws.com | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| Asia Pasifik (Thailand) | ap-tenggara-7 | rds.ap-southeast-7.amazonaws.com | HTTPS | |
| Asia Pacific (Tokyo) | ap-northeast-1 | rds.ap-northeast-1.amazonaws.com | HTTPS | |
| Kanada (Pusat) | ca-central-1 | rds.ca-central-1.amazonaws.com | HTTPS | |
| Kanada Barat (Calgary) | ca-west-1 | rds.ca-west-1.amazonaws.com | HTTPS | |
| Eropa (Frankfurt) | eu-central-1 | rds.eu-central-1.amazonaws.com | HTTPS | |
| Eropa (Irlandia) | eu-west-1 | rds.eu-west-1.amazonaws.com | HTTPS | |
| Eropa (London) | eu-west-2 | rds.eu-west-2.amazonaws.com | HTTPS | |
| Eropa (Milan) | eu-south-1 | rds.eu-south-1.amazonaws.com | HTTPS | |
| Eropa (Paris) | eu-west-3 | rds.eu-west-3.amazonaws.com | HTTPS | |
| Eropa (Spanyol) | eu-south-2 | rds.eu-south-2.amazonaws.com | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| Eropa (Stockholm) | eu-north-1 | rds.eu-north-1.amazonaws.com | HTTPS | |
| Eropa (Zürich) | eu-central-2 | rds.eu-central-2.amazonaws.com | HTTPS | |
| Israel (Tel Aviv) | il-central-1 | rds.il-central-1.amazonaws.com | HTTPS | |
| Meksiko (Tengah) | mx-pusat-1 | rds.mx-central-1.amazonaws.com | HTTPS | |
| Timur Tengah (Bahrain) | me-south-1 | rds.me-south-1.amazonaws.com | HTTPS | |
| Timur Tengah (UAE) | me-central-1 | rds.me-central-1.amazonaws.com | HTTPS | |
| Amerika Selatan (Sao Paulo) | sa-east-1 | rds.sa-east-1.amazonaws.com | HTTPS | |
| AWS GovCloud (AS-Timur) | us-gov-east-1 | rds.us-gov-east-1.amazonaws.com | HTTPS | |
| AWS GovCloud (AS-Barat) | us-gov-west-1 | rds.us-gov-west-1.amazonaws.com | HTTPS | |

Ketersediaan Wilayah Aurora PostgreSQL

Tabel berikut menunjukkan AWS Wilayah di mana Aurora PostgreSQL saat ini tersedia dan titik akhir untuk setiap Wilayah.

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| AS Timur (Ohio) | us–east-2 | rds.us-east-2.amazonaws.com | HTTPS | |
| AS Timur (Virginia Utara) | us-east-1 | rds.us-east-1.amazonaws.com | HTTPS | |
| AS Barat (California Utara) | us-west-1 | rds.us-west-1.amazonaws.com | HTTPS | |
| AS Barat (Oregon) | us-west-2 | rds.us-west-2.amazonaws.com | HTTPS | |
| Afrika (Cape Town) | af-south-1 | rds.af-south-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Hong Kong) | ap-east-1 | rds.ap-east-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Hyderabad) | ap-south-2 | rds.ap-south-2.amazonaws.com | HTTPS | |
| Asia Pasifik (Jakarta) | ap-southeast-3 | rds.ap-southeast-3.amazonaws.com | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| Asia Pasifik (Malaysia) | ap-southeast-5 | rds.ap-southeast-5.amazonaws.com | HTTPS | |
| Asia Pasifik (Melbourne) | ap-southeast-4 | rds.ap-southeast-4.amazonaws.com | HTTPS | |
| Asia Pasifik (Mumbai) | ap-south-1 | rds.ap-south-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Osaka) | ap-northeast-3 | rds.ap-northeast-3.amazonaws.com | HTTPS | |
| Asia Pasifik (Seoul) | ap-northeast-2 | rds.ap-northeast-2.amazonaws.com | HTTPS | |
| Asia Pasifik (Singapura) | ap-southeast-1 | rds.ap-southeast-1.amazonaws.com | HTTPS | |
| Asia Pasifik (Sydney) | ap-southeast-2 | rds.ap-southeast-2.amazonaws.com | HTTPS | |
| Asia Pasifik (Taipei) | ap-timur-2 | rds.ap-east-2.amazonaws.com | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| Asia Pasifik (Thailand) | ap-tenggara 7 | rds.ap-southeast-7.amazonaws.com | HTTPS | |
| Asia Pacific (Tokyo) | ap-northeast-1 | rds.ap-northeast-1.amazonaws.com | HTTPS | |
| Kanada (Pusat) | ca-central-1 | rds.ca-central-1.amazonaws.com | HTTPS | |
| Kanada Barat (Calgary) | ca-west-1 | rds.ca-west-1.amazonaws.com | HTTPS | |
| Eropa (Frankfurt) | eu-central-1 | rds.eu-central-1.amazonaws.com | HTTPS | |
| Eropa (Irlandia) | eu-west-1 | rds.eu-west-1.amazonaws.com | HTTPS | |
| Eropa (London) | eu-west-2 | rds.eu-west-2.amazonaws.com | HTTPS | |
| Eropa (Milan) | eu-south-1 | rds.eu-south-1.amazonaws.com | HTTPS | |
| Eropa (Paris) | eu-west-3 | rds.eu-west-3.amazonaws.com | HTTPS | |
| Eropa (Spanyol) | eu-south-2 | rds.eu-south-2.amazonaws.com | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| Eropa (Stockholm) | eu-north-1 | rds.eu-north-1.amazonaws.com | HTTPS | |
| Eropa (Zürich) | eu-central-2 | rds.eu-central-2.amazonaws.com | HTTPS | |
| Israel (Tel Aviv) | il-central-1 | rds.il-central-1.amazonaws.com | HTTPS | |
| Meksiko (Tengah) | mx-pusat-1 | rds.mx-central-1.amazonaws.com | HTTPS | |
| Timur Tengah (Bahrain) | me-south-1 | rds.me-south-1.amazonaws.com | HTTPS | |
| Timur Tengah (UAE) | me-central-1 | rds.me-central-1.amazonaws.com | HTTPS | |
| Amerika Selatan (Sao Paulo) | sa-east-1 | rds.sa-east-1.amazonaws.com | HTTPS | |
| AWS GovCloud (AS-Timur) | us-gov-east-1 | rds.us-gov-east-1.amazonaws.com | HTTPS | |
| AWS GovCloud (AS-Barat) | us-gov-west-1 | rds.us-gov-west-1.amazonaws.com | HTTPS | |

# Zona Ketersediaan

Zona Ketersediaan adalah lokasi yang terisolasi di Wilayah AWS tertentu. Setiap Wilayah memiliki beberapa Zona Ketersediaan (AZ) yang dirancang untuk menyediakan ketersediaan tinggi bagi Wilayah tersebut. AZ diidentifikasi oleh kode AWS Wilayah diikuti oleh pengidentifikasi huruf (misalnya,`us-east-1a`). Jika membuat VPC dan subnet, bukan menggunakan VPC default, Anda akan menentukan setiap subnet dalam AZ tertentu. Saat Anda membuat klaster DB Aurora, Aurora membuat instans utama di salah satu subnet dalam grup subnet DB VPC. Dengan demikian, instans tersebut akan dikaitkan dengan AZ tertentu yang dipilih oleh Aurora.

Setiap cluster Aurora DB menyimpan salinan penyimpanannya dalam tiga terpisah yang AZs dipilih secara otomatis oleh Aurora dari grup subnet AZs DB Anda. Setiap instans DB di cluster harus berada di salah satu dari ketiganya AZs.

Saat Anda membuat instans DB di cluster Anda, Aurora secara otomatis memilih AZ yang sesuai untuk instance tersebut jika Anda tidak menentukan AZ.

Gunakan EC2 perintah describe-availability-zonesAmazon sebagai berikut untuk menjelaskan Availability Zone dalam Wilayah tertentu yang diaktifkan untuk akun Anda.

```
aws ec2 describe-availability-zones --region region-name
```

Misalnya, untuk mendeskripsikan Zona Ketersediaan dalam Wilayah AS Timur (Virginia Utara) (us-east-1) yang diaktifkan untuk akun Anda, jalankan perintah berikut:

```
aws ec2 describe-availability-zones --region us-east-1
```

Untuk mempelajari cara menentukan AZ saat membuat klaster atau menambahkan instans ke klaster, lihat Konfigurasikan jaringan untuk klaster DB.

# Zona waktu lokal untuk klaster DB Amazon Aurora

Secara default, zona waktu untuk klaster DB Amazon Aurora adalah Waktu Universal Terkoordinasi (UTC). Anda dapat mengatur zona waktu untuk instans di klaster DB ke zona waktu lokal untuk aplikasi Anda.

Untuk mengatur zona waktu lokal bagi klaster DB, atur parameter zona waktu ke salah satu nilai yang mendukung. Anda mengatur parameter ini di grup parameter klaster untuk klaster DB.

- Untuk Aurora MySQL, nama parameter ini adalah `time_zone`. Untuk informasi tentang praktik terbaik untuk mengatur parameter `time_zone`, lihat Mengoptimalkan operasi stempel waktu.
- Untuk Aurora PostgreSQL, nama parameter ini adalah `timezone`.

Saat Anda mengatur parameter zona waktu untuk klaster DB, semua instans di klaster DB berubah menggunakan zona waktu lokal baru. Dalam beberapa kasus, klaster Aurora DB lainnya mungkin menggunakan grup parameter klaster yang sama. Jika demikian, semua instans dalam klaster DB tersebut juga berubah menggunakan zona waktu lokal baru. Untuk informasi tentang parameter tingkat klaster, lihat Parameter klaster DB dan instans DB Amazon Aurora.

Setelah Anda mengatur zona waktu lokal, semua koneksi baru ke basis data mencerminkan perubahan tersebut. Dalam beberapa kasus, Anda mungkin memiliki koneksi terbuka ke basis data saat mengubah zona waktu lokal. Jika demikian, pembaruan zona waktu lokal tidak akan terlihat hingga Anda menutup koneksi dan membuka koneksi baru.

Jika Anda mereplikasi di seluruh AWS Wilayah, cluster DB sumber replikasi dan replika menggunakan grup parameter yang berbeda. Grup parameter unik untuk suatu AWS Wilayah. Untuk menggunakan zona waktu lokal yang sama bagi setiap instans, pastikan untuk mengatur parameter zona waktu dalam grup parameter untuk sumber replikasi dan replika.

Saat Anda memulihkan klaster DB dari snapshot klaster DB, zona waktu lokal diatur ke UTC. Anda dapat memperbarui zona waktu ke zona waktu lokal Anda setelah pemulihan selesai. Dalam beberapa kasus, Anda dapat memulihkan klaster DB ke titik waktu tertentu. Jika demikian, zona waktu lokal untuk klaster DB yang dipulihkan adalah pengaturan zona waktu dari grup parameter klaster DB yang dipulihkan.

Tabel berikut mencantumkan beberapa nilai yang dapat Anda gunakan untuk mengatur zona waktu lokal. Untuk mencantumkan semua zona waktu yang tersedia, Anda dapat menggunakan kueri SQL berikut:

- Aurora MySQL: `select * from mysql.time_zone_name;`
- Aurora PostgreSQL: `select * from pg_timezone_names;`

> **Note**
>
> Untuk beberapa zona waktu, nilai waktu untuk rentang tanggal tertentu mungkin salah dilaporkan seperti yang tercantum dalam tabel. Untuk zona waktu Australia, singkatan zona

waktu yang ditampilkan adalah nilai yang sudah tidak berlaku sebagaimana tercantum dalam tabel.

| Zona waktu | Catatan |
| --- | --- |
| `Africa/Harare` | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 28 Feb 1903 21.49.40 GMT menjadi 28 Feb 1903 21.55.48 GMT. |
| `Africa/Monrovia` | |
| `Africa/Nairobi` | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 31 Des 1939 21.30.00 GMT menjadi 31 Des 1959 21.15.15 GMT. |
| `Africa/Windhoek` | |
| `America/Bogota` | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 23 Nov 1914 04.56.16 GMT menjadi 23 Nov 1914 04.56.20 GMT. |
| `America/Caracas` | |
| `America/Chihuahua` | |
| `America/Cuiaba` | |
| `America/Denver` | |
| `America/Fortaleza` | Dalam beberapa kasus, untuk klaster DB di Wilayah Amerika Selatan (Sao Paulo), waktu tidak ditampilkan dengan benar untuk zona waktu Brasil yang baru diubah. Jika demikian, atur ulang parameter zona waktu klaster DB ke `America/Fortaleza` . |
| `America/Guatemala` | |
| `America/Halifax` | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 27 Okt 1918 05.00.00 GMT menjadi 31 Okt 1918 05.00.00 GMT. |

| Zona waktu | Catatan |
| --- | --- |
| America/Manaus | Jika klaster DB Anda berada di zona waktu Amerika Selatan (Cuiaba) dan waktu yang diharapkan tidak ditampilkan dengan benar untuk zona waktu Brasil yang baru diubah, atur ulang parameter zona waktu klaster DB ke America/Manaus . |
| America/Matamoros | |
| America/Monterrey | |
| America/Montevideo | |
| America/Noronha | |
| America/Phoenix | |
| America/Tijuana | |
| Asia/Ashgabat | |
| Asia/Baghdad | |
| Asia/Baku | |
| Asia/Bangkok | |
| Asia/Beirut | |
| Asia/Calcutta | |
| Asia/Kabul | |
| Asia/Karachi | |
| Asia/Kathmandu | |

| Zona waktu | Catatan |
| --- | --- |
| Asia/Muscat | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 31 Des 1919 20.05.36 GMT menjadi 31 Des 1919 20.05.40 GMT. |
| Asia/Riyadh | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 13 Mar 1947 20.53.08 GMT menjadi 31 Des 1949 20.53.08 GMT. |
| Asia/Seoul | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Nov 1904 15.30.00 GMT menjadi 07 Sep 1945 15.00.00 GMT. |
| Asia/Shanghai | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 31 Des 1927 15.54.08 GMT menjadi 02 Jun 1940 16.00.00 GMT. |
| Asia/Singapore | |
| Asia/Taipei | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Sep 1937 16.00.00 GMT menjadi 29 Sep 1979 15.00.00 GMT. |
| Asia/Tehran | |
| Asia/Tokyo | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Sep 1937 15.00.00 GMT menjadi 31 Des 1937 15.00.00 GMT. |
| Asia/Ulaanbaatar | |
| Atlantic/Azores | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 24 Mei 1911 01.54.32 GMT menjadi 01 Jan 1912 01.54.32 GMT. |
| Australia/Adelaide | Singkatan untuk zona waktu ini ditampilkan sebagai CST, bukan ACDT/ACST. |
| Australia/Brisbane | Singkatan untuk zona waktu ini ditampilkan sebagai EST, bukan AEDT/AEST. |
| Australia/Darwin | Singkatan untuk zona waktu ini ditampilkan sebagai CST, bukan ACDT/ACST. |

| Zona waktu | Catatan |
| --- | --- |
| Australia/Hobart | Singkatan untuk zona waktu ini ditampilkan sebagai EST, bukan AEDT/AEST. |
| Australia/Perth | Singkatan untuk zona waktu ini dikembalikan sebagai WST, bukan AWST AWDT/. |
| Australia/Sydney | Singkatan untuk zona waktu ini ditampilkan sebagai EST, bukan AEDT/AEST. |
| Brazil/East | |
| Canada/Saskatchewan | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 27 Okt 1918 08.00.00 GMT menjadi 31 Okt 1918 08.00.00 GMT. |
| Europe/Amsterdam | |
| Europe/Athens | |
| Europe/Dublin | |
| Europe/Helsinki | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Apr 1921 22.20.08 GMT menjadi 30 Apr 1921 22.20.11 GMT. |
| Europe/Paris | |
| Europe/Prague | |
| Europe/Sarajevo | |
| Pacific/Auckland | |
| Pacific/Guam | |
| Pacific/Honolulu | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 21 Mei 1933 11.30.00 GMT menjadi 30 Sep 1945 11.30.00 GMT. |

| Zona waktu | Catatan |
| --- | --- |
| Pacific/Samoa | Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 01 Jan 1911 11.22.48 GMT menjadi 01 Jan 1950 11.30.00 GMT. |
| US/Alaska | |
| US/Central | |
| US/Eastern | |
| US/East-Indiana | |
| US/Pacific | |
| UTC | |

# Fitur yang didukung di Amazon Aurora oleh dan mesin Wilayah AWS Aurora DB

Mesin basis data Aurora yang kompatibel dengan MySQL dan PostgreSQL mendukung beberapa fitur dan opsi Amazon Aurora dan Amazon RDS. Dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data, dan di seluruh Wilayah AWS. Untuk mengidentifikasi dukungan versi mesin basis data Aurora dan ketersediaan untuk fitur tertentu Wilayah AWS, Anda dapat menggunakan bagian berikut.

Beberapa dari fitur ini adalah kemampuan khusus Aurora. Misalnya, Aurora Serverless, Basis data global Aurora, dan dukungan untuk integrasi dengan layanan pembelajaran AWS mesin tidak didukung oleh Amazon RDS. Fitur lain, seperti Proksi Amazon RDS, didukung oleh Amazon Aurora dan Amazon RDS.

Daerah yang Didukung dan mesin DB

- Konvensi tabel
- Daerah yang Didukung dan mesin Aurora DB untuk Penerapan Biru/Hijau
- Daerah yang Didukung dan engine Aurora DB untuk konfigurasi penyimpanan cluster
- Daerah yang Didukung dan mesin Aurora DB untuk aliran aktivitas database
- Daerah yang Didukung dan mesin Aurora DB untuk mengekspor data cluster ke Amazon S3
- Daerah yang Didukung dan mesin Aurora DB untuk mengekspor data snapshot ke Amazon S3
- Daerah yang Didukung dan mesin DB untuk database global Aurora
- Daerah yang Didukung dan mesin Aurora DB untuk IAM otentikasi database
- Daerah yang Didukung dan mesin Aurora DB untuk otentikasi Kerberos
- Daerah yang Didukung dan mesin DB untuk pembelajaran mesin Aurora
- Daerah yang Didukung dan engine Aurora DB untuk Performance Insights
- Daerah yang Didukung dan mesin Aurora DB untuk integrasi Nol-ETL
- Daerah yang Didukung dan mesin Aurora DB untuk Amazon RDS Proxy
- Daerah yang Didukung dan engine Aurora DB untuk integrasi Secrets Manager
- Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2
- Aurora Serverless v1
- Daerah yang Didukung dan engine Aurora DB untuk RDS Data API
- Daerah yang Didukung dan engine Aurora DB untuk patching zero-downtime () ZDP

- Daerah yang Didukung dan engine DB untuk fitur asli mesin Aurora

## Konvensi tabel

Tabel-tabel di bagian fitur tersebut menggunakan pola berikut untuk menentukan nomor versi dan tingkat ketersediaan:

- Versi x.y – Hanya mendukung versi tertentu saja.
- Versi x.y dan yang lebih tinggi – Versi yang ditentukan dan semua versi minor yang lebih tinggi dari versi utamanya didukung. Misalnya, “versi 10.11 dan yang lebih tinggi” berarti versi 10.11, 10.11.1, dan 10.12 tersedia.

## Daerah yang Didukung dan mesin Aurora DB untuk Penerapan Biru/Hijau

Deployment blue/green menyalin lingkungan basis data produksi di lingkungan pementasan yang terpisah dan tersinkron. Dengan menggunakan Amazon RDS Blue/Green Deployment, Anda dapat membuat perubahan pada database di lingkungan pementasan tanpa memengaruhi lingkungan produksi. Misalnya, Anda dapat meningkatkan versi mesin DB besar atau kecil, mengubah parameter basis data, atau membuat perubahan skema di lingkungan pementasannya. Setelah siap, Anda dapat mempromosikan lingkungan pementasan menjadi lingkungan basis data produksi baru. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

### Penerapan Biru/Hijau dengan Aurora My SQL

Fitur Penerapan Biru/Hijau tersedia untuk semua versi Aurora My SQL Wilayah AWS.

### Penerapan Biru/Hijau dengan Aurora Postgre SQL

Wilayah dan versi mesin berikut tersedia untuk Penerapan Biru/Hijau dengan Aurora Postgre. SQL

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Semua Wilayah AWS | Versi 16.1 dan lebih tinggi | Versi 15.4 dan yang lebih tinggi | Versi 14.9 dan yang lebih tinggi | Versi 13.12 dan yang lebih tinggi | Versi 12.16 dan yang lebih tinggi | Versi 11.21 dan yang lebih tinggi |

# Daerah yang Didukung dan engine Aurora DB untuk konfigurasi penyimpanan cluster

Amazon Aurora memiliki dua konfigurasi penyimpanan cluster DB, Aurora I/O-Optimized and Aurora StandardUntuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

## Aurora I/O-Optimized

Aurora I/O-Optimized tersedia di semua Wilayah AWS untuk versi Amazon Aurora berikut:

- Aurora SQL Versi saya 3.03.1 dan lebih tinggi
- Aurora Postgre SQL versi 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, dan 13.10 dan lebih tinggi

## Aurora Standard

Aurora Standard tersedia di semua Wilayah AWS untuk semua versi Aurora My dan SQL Aurora Postgre. SQL

# Daerah yang Didukung dan mesin Aurora DB untuk aliran aktivitas database

Dengan menggunakan aliran aktivitas basis data di Aurora, Anda dapat memantau dan mengatur alarm untuk aktivitas audit di basis data Aurora. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Aliran aktivitas basis data tidak didukung untuk fitur berikut:

- Aurora Serverless v1
- Aurora Serverless v2
- Babelfish untuk Aurora Postgre SQL

Topik

- Aliran aktivitas database dengan Aurora My SQL
- Aliran aktivitas database dengan Aurora Postgre SQL

## Aliran aktivitas database dengan Aurora My SQL

Wilayah dan versi mesin berikut tersedia untuk aliran aktivitas database dengan SQL Aurora My.

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| AS Timur (Virginia Utara) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| AS Timur (Ohio) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| AS Barat (California Utara) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| AS Barat (Oregon) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Afrika (Cape Town) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Mumbai) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| Asia Pasifik (Seoul) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Kanada (Pusat) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Tidak tersedia | Tidak tersedia |
| Tiongkok (Beijing) | Tidak tersedia | Tidak tersedia |
| Tiongkok (Ningxia) | Tidak tersedia | Tidak tersedia |
| Eropa (Frankfurt) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Eropa (Irlandia) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Eropa (London) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Eropa (Milan) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Eropa (Paris) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Eropa (Spanyol) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| Eropa (Stockholm) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Eropa (Zürich) | Tidak tersedia | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Timur Tengah (UAE) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Semua versi yang tersedia | Aurora versi 2.11 dan yang lebih tinggi |

## Aliran aktivitas database dengan Aurora Postgre SQL

Wilayah dan versi mesin berikut tersedia untuk aliran aktivitas database dengan SQL Aurora Postgre.

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| AS Timur (Virginia Utara) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| AS Barat (California Utara) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Afrika (Cape Town) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Asia Pasifik (Jakarta) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Mumbai) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Asia Pasifik (Singapura) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Tiongkok (Beijing) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Tiongkok (Ningxia) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Eropa (Frankfurt) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (London) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Milan) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Paris) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Eropa (Spanyol) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Zürich) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Timur Tengah (UAE) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Amerika Selatan (Sao Paulo) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Semua versi yang tersedia | Semua versi yang tersedia | Semua versi yang tersedia | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

## Daerah yang Didukung dan mesin Aurora DB untuk mengekspor data cluster ke Amazon S3

Anda dapat mengekspor data klaster DB Aurora ke bucket Amazon S3. Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi selengkapnya, lihat Mengekspor data klaster DB ke Amazon S3.

Mengekspor data klaster ke S3 tersedia di Wilayah AWS berikut:

- AS Timur (N. Virginia)
- AS Timur (Ohio)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Asia Pasifik (Hong Kong)
- Asia Pasifik (Hyderabad)
- Asia Pasifik (Jakarta)
- Asia Pasifik (Melbourne)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)

- Asia Pasifik (Tokyo)
- (Canada (Central)
- Kanada Barat (Calgary)
- Tiongkok (Ningxia)
- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)
- Eropa (Spanyol)
- Eropa (Stockholm)
- Eropa (Zürich)
- Israel (Tel Aviv)
- Timur Tengah (UAE)
- Amerika Selatan (Sao Paulo)

Topik


- Mengekspor data cluster ke S3 dengan Aurora My SQL
- Mengekspor data cluster ke S3 dengan Aurora Postgre SQL


## Mengekspor data cluster ke S3 dengan Aurora My SQL

Semua versi Aurora My SQL engine yang tersedia saat ini mendukung ekspor data cluster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora Milik Saya. SQL

## Mengekspor data cluster ke S3 dengan Aurora Postgre SQL

Semua versi SQL mesin Aurora Postgre yang tersedia saat ini mendukung ekspor data cluster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora Postgre. SQL

# Daerah yang Didukung dan mesin Aurora DB untuk mengekspor data snapshot ke Amazon S3

Anda dapat mengekspor data snapshot klaster DB Aurora ke bucket Amazon S3. Anda dapat mengekspor snapshot manual dan snapshot sistem otomatis. Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Mengekspor snapshot ke S3 tersedia di semua Wilayah AWS kecuali yang berikut:

- Asia Pasifik (Malaysia)
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

Topik

- Mengekspor data snapshot ke S3 dengan Aurora My SQL
- Mengekspor data snapshot ke S3 dengan Aurora Postgre SQL

## Mengekspor data snapshot ke S3 dengan Aurora My SQL

Semua versi Aurora My SQL engine yang tersedia saat ini mendukung ekspor data snapshot cluster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora Milik Saya. SQL

## Mengekspor data snapshot ke S3 dengan Aurora Postgre SQL

Semua versi SQL mesin Aurora Postgre yang tersedia saat ini mendukung ekspor data snapshot cluster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora Postgre. SQL

# Daerah yang Didukung dan mesin DB untuk database global Aurora

Basis data global Aurora adalah database tunggal yang mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh Wilayah. Ini memberikan toleransi kesalahan bawaan untuk penerapan Anda karena instans

DB tidak bergantung pada satu Wilayah AWS, tetapi pada beberapa Wilayah dan Zona Ketersediaan yang berbeda. Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

Topik

- Basis data global Aurora dengan Aurora MySQL
- Basis data global Aurora dengan Aurora PostgreSQL

## Basis data global Aurora dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk basis data global dengan Aurora MySQL.

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AS Timur (Virginia Utara) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Afrika (Cape Town) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.1 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.1 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 3.02.0 dan yang lebih tinggi | Versi 2.11.2 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 3.04.0 dan lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Asia Pasifik (Melbourne) | Versi 3.03.0 dan yang lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.3 dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Tiongkok (Beijing) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.2 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.2 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Eropa (London) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Eropa (Milan) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.1 dan yang lebih tinggi |
| Eropa (Paris) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 3.02.0 dan yang lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| Eropa (Zürich) | Versi 3.02.0 dan yang lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |
| Israel (Tel Aviv) | Versi 3.02.0 dan yang lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.1 dan yang lebih tinggi |
| Timur Tengah (UEA) | Versi 3.02.0 dan yang lebih tinggi | Versi 2.07.6 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.1 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.0 dan yang lebih tinggi |

## Basis data global Aurora dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk basis data global dengan Aurora PostgreSQL.

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| AS Timur (Virginia Utara) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| | | | | | | | lebih tinggi |
| Afrika (Cape Town) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Malaysia) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Seoul) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Kanada (Pusat) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Tiongkok (Beijing) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Eropa (Frankfurt) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (London) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Milan) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Eropa (Paris) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Zürich) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Timur Tengah (UEA) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| AWS GovCloud (AS-Timur) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

## Daerah yang Didukung dan mesin Aurora DB untuk IAM otentikasi database

Dengan otentikasi IAM database di Aurora, Anda dapat mengautentikasi ke cluster DB Anda menggunakan AWS Identity and Access Management (IAM) otentikasi database. Dengan metode autentikasi ini, Anda tidak perlu menggunakan kata sandi saat terhubung ke klaster DB. Sebagai gantinya, Anda menggunakan token autentikasi. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

Topik

- IAMotentikasi database dengan Aurora My SQL
- IAMotentikasi database dengan Aurora Postgre SQL

### IAMotentikasi database dengan Aurora My SQL

IAMotentikasi database dengan Aurora SQL My tersedia di semua Wilayah untuk versi berikut:

- Aurora My SQL 3 - Semua versi yang tersedia
- Aurora My SQL 2 - Semua versi yang tersedia

### IAMotentikasi database dengan Aurora Postgre SQL

IAMotentikasi database dengan Aurora SQL Postgre tersedia di semua Wilayah untuk versi mesin berikut:

- Aurora Postgre SQL 16 - Semua versi yang tersedia
- Aurora Postgre SQL 15 - Semua versi yang tersedia
- Aurora Postgre SQL 14 - Semua versi yang tersedia
- Aurora Postgre SQL 13 - Semua versi yang tersedia
- Aurora Postgre SQL 12 - Semua versi yang tersedia
- Aurora Postgre SQL 11 — Semua versi yang tersedia

## Daerah yang Didukung dan mesin Aurora DB untuk otentikasi Kerberos

Dengan menggunakan autentikasi Kerberos di Aurora, Anda dapat mendukung autentikasi eksternal pengguna basis data menggunakan Kerberos dan Microsoft Active Directory. Penggunaan Kerberos dan Active Directory memberikan manfaat masuk tunggal dan autentikasi terpusat pengguna basis data. Kerberos dan Active Directory tersedia dengan AWS Directory Service for Microsoft Active Directory, fitur dari. AWS Directory Service Untuk informasi selengkapnya, lihat Autentikasi Kerberos.

Topik

- Autentikasi Kerberos dengan Aurora MySQL
- Autentikasi Kerberos dengan Aurora PostgreSQL

### Autentikasi Kerberos dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Autentikasi Kerberos dengan Aurora MySQL.

| Wilayah | Aurora MySQL versi 3 |
| --- | --- |
| AS Timur (Virginia Utara) | Versi 3.03.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 |
| --- | --- |
| AS Timur (Ohio) | Versi 3.03.0 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 3.03.0 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 3.03.0 dan yang lebih tinggi |
| Afrika (Cape Town) | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Tidak tersedia |
| Asia Pasifik (Jakarta) | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia |
| Asia Pasifik (Mumbai) | Versi 3.03.0 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Tidak tersedia |
| Asia Pasifik (Seoul) | Versi 3.03.0 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 3.03.0 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 3.03.0 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 3.03.0 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 3.03.0 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Tidak tersedia |
| Tiongkok (Beijing) | Versi 3.03.0 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 3.03.0 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 3.03.0 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 3.03.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 |
|---|---|
| Eropa (London) | Versi 3.03.0 dan yang lebih tinggi |
| Eropa (Milan) | Tidak tersedia |
| Eropa (Paris) | Versi 3.03.0 dan yang lebih tinggi |
| Eropa (Spanyol) | Tidak tersedia |
| Eropa (Stockholm) | Versi 3.03.0 dan yang lebih tinggi |
| Eropa (Zürich) | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia |
| Timur Tengah (Bahrain) | Tidak tersedia |
| Timur Tengah (UEA) | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Versi 3.03.0 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 3.03.0 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 3.03.0 dan yang lebih tinggi |

## Autentikasi Kerberos dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Autentikasi Kerberos dengan Aurora PostgreSQL.

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|
| AS Timur (Virginia Utara) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AS Timur (Ohio) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|
| AS Barat (California Utara) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AS Barat (Oregon) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Afrika (Cape Town) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hyderabad) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Jakarta) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Melbourne) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|
| Asia Pasifik (Mumbai) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Osaka) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Seoul) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Singapura) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Sydney) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Tokyo) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Kanada (Pusat) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Kanada Barat (Calgary) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Tiongkok (Beijing) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|
| Tiongkok (Ningxia) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Frankfurt) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Irlandia) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (London) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Milan) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Eropa (Paris) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Spanyol) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Eropa (Stockholm) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Zürich) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|
| Timur Tengah (UEA) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AWS GovCloud (AS-Timur) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AWS GovCloud (AS-Barat) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

## Daerah yang Didukung dan mesin DB untuk pembelajaran mesin Aurora

Dengan menggunakan pembelajaran mesin Amazon Aurora, Anda dapat mengintegrasikan cluster Aurora DB Anda dengan salah satu layanan pembelajaran AWS mesin berikut, tergantung pada kebutuhan Anda. Mereka masing-masing mendukung kasus penggunaan pembelajaran mesin tertentu.

Amazon Bedrock adalah layanan yang dikelola sepenuhnya yang membuat model fondasi terkemuka dari perusahaan AI tersedia melaluiAPI, bersama dengan perkakas pengembang untuk membantu membangun dan menskalakan aplikasi AI generatif.

Amazon Comprehend adalah layanan NLP pemrosesan bahasa alami () yang digunakan untuk mengekstrak wawasan dari dokumen. Dengan menggunakan machine learning Aurora dengan Amazon Comprehend, Anda dapat menentukan sentimen teks dalam tabel basis data Anda.

SageMaker AI adalah layanan pembelajaran mesin berfitur lengkap. Ilmuwan data menggunakan Amazon SageMaker AI untuk membangun, melatih, dan menguji model pembelajaran mesin

untuk berbagai tugas inferensi, seperti deteksi penipuan. Dengan menggunakan pembelajaran mesin Aurora dengan SageMaker AI, pengembang database dapat menggunakan fungsionalitas SageMaker AI dalam kode. SQL

Tidak semua Wilayah AWS mendukung semua layanan pembelajaran mesin. Hanya Wilayah AWS dukungan tertentu pembelajaran mesin Aurora dan dengan demikian menyediakan akses ke layanan ini dari cluster Aurora DB. Proses integrasi untuk machine learning Aurora juga berbeda menurut mesin basis data. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora.

Topik

- Pembelajaran mesin Aurora dengan Aurora My SQL
- Pembelajaran mesin Aurora dengan Aurora Postgre SQL

## Pembelajaran mesin Aurora dengan Aurora My SQL

Amazon Bedrock hanya didukung di Aurora SQL My versi 3.06 dan lebih tinggi. Untuk informasi tentang ketersediaan Wilayah untuk Amazon Bedrock, lihat Dukungan model oleh Wilayah AWS di Panduan Pengguna Amazon Bedrock.

Pembelajaran mesin Aurora dengan Amazon Comprehend dan Amazon SageMaker AI didukung untuk Aurora My dalam daftar di tabel. SQL Wilayah AWS Selain memiliki versi Aurora My SQL tersedia, Wilayah AWS harus juga mendukung layanan yang ingin Anda gunakan. Untuk daftar Wilayah AWS di mana Amazon SageMaker AI tersedia, lihat titik akhir Amazon SageMaker AI dan kuota di. Referensi Umum Amazon Web Services Untuk daftar Wilayah AWS tempat Amazon Comprehend tersedia, lihat Amazon Comprehend endpoint dan kuota di. Referensi Umum Amazon Web Services

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| AS Timur (Virginia Utara) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| AS Barat (Oregon) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Afrika (Cape Town) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 3.04.0 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Melbourne) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07.3 dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| Kanada (Pusat) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Tiongkok (Beijing) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (London) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (Milan) | Tidak tersedia | Tidak tersedia |
| Eropa (Paris) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Eropa (Zürich) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Israel (Tel Aviv) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| Timur Tengah (Bahrain) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Timur Tengah (UAE) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan yang lebih tinggi |

## Pembelajaran mesin Aurora dengan Aurora Postgre SQL

Untuk informasi tentang dukungan versi untuk Amazon Bedrock di Aurora SQL Postgre, lihat. Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock

Pembelajaran mesin Aurora dengan Amazon Comprehend dan Amazon SageMaker AI didukung untuk Aurora Postgre dalam daftar di tabel. SQL Wilayah AWS Selain memiliki versi Aurora Postgre Anda SQL tersedia, Wilayah AWS harus juga mendukung layanan yang ingin Anda gunakan. Untuk daftar Wilayah AWS di mana Amazon SageMaker AI tersedia, lihat titik akhir Amazon SageMaker AI dan kuota di. Referensi Umum Amazon Web Services Untuk daftar Wilayah AWS tempat Amazon Comprehend tersedia, lihat Amazon Comprehend endpoint dan kuota di. Referensi Umum Amazon Web Services

Wilayah dan versi mesin berikut tersedia untuk pembelajaran mesin Aurora dengan Aurora Postgre. SQL

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| AS Timur (Virginia Utara) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Afrika (Cape Town) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Asia Pasifik (Jakarta) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Asia Pasifik (Sydney) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Kanada (Pusat) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Tiongkok (Beijing) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Eropa (Irlandia) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Eropa (London) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Eropa (Milan) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Eropa (Paris) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Eropa (Zürich) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |

| Wilayah | Aurora Postgre 16 SQL | Aurora Postgre 15 SQL | Aurora Postgre 14 SQL | Aurora Postgre 13 SQL | Aurora Postgre 12 SQL | Aurora Postgre 11 SQL |
|---|---|---|---|---|---|---|
| Israel (Tel Aviv) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Timur Tengah (UAE) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.3 dan yang lebih tinggi | Versi 12.4 dan yang lebih tinggi | Versi 11.9, 11.12, dan yang lebih tinggi |

# Daerah yang Didukung dan engine Aurora DB untuk Performance Insights

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Wawasan Performa memperlengkap fitur pemantauan Amazon RDS yang sudah ada untuk mengilustrasikan dan membantu Anda menganalisis performa basis data. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan muatan basis data pada muatan instans DB Amazon RDS dan memfilter muatan menurut peristiwa tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi selengkapnya, lihat Ikhtisar Wawasan Performa tentang Amazon Aurora.

Untuk informasi dukungan Wilayah, mesin DB, dan kelas instans untuk fitur Wawasan Performa, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa.

Topik

- Wawasan Performa dengan Aurora MySQL

- Wawasan Performa dengan Aurora PostgreSQL
- Wawasan Performa dengan Aurora Nirserver

## Wawasan Performa dengan Aurora MySQL

> **Note**
>
> Dukungan versi engine berbeda untuk Wawasan Performa dengan Aurora MySQL jika Anda mengaktifkan kueri paralel. Untuk informasi selengkapnya tentang kueri paralel, lihat Kueri paralel untuk Amazon Aurora My SQL.

Topik

- Wawasan Performa dengan Aurora MySQL dan kueri paralel dimatikan
- Wawasan Performa dengan Aurora MySQL dan kueri paralel dinyalakan

### Wawasan Performa dengan Aurora MySQL dan kueri paralel dimatikan

Wilayah dan versi mesin berikut tersedia untuk Wawasan Performa dengan Aurora MySQL dan kueri paralel dimatikan.

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AS Timur (Virginia Utara) | Semua versi | Semua versi |
| AS Timur (Ohio) | Semua versi | Semua versi |
| AS Barat (California Utara) | Semua versi | Semua versi |
| AS Barat (Oregon) | Semua versi | Semua versi |
| Afrika (Cape Town) | Semua versi | Semua versi |
| Asia Pasifik (Hong Kong) | Semua versi | Semua versi |
| Asia Pasifik (Hyderabad) | Semua versi | Semua versi |
| Asia Pasifik (Jakarta) | Semua versi | Semua versi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Asia Pasifik (Malaysia) | Semua versi | Semua versi |
| Asia Pasifik (Melbourne) | Semua versi | Semua versi |
| Asia Pasifik (Mumbai) | Semua versi | Semua versi |
| Asia Pasifik (Osaka) | Semua versi | Semua versi |
| Asia Pasifik (Seoul) | Semua versi | Semua versi |
| Asia Pasifik (Singapura) | Semua versi | Semua versi |
| Asia Pasifik (Sydney) | Semua versi | Semua versi |
| Asia Pasifik (Tokyo) | Semua versi | Semua versi |
| Kanada (Pusat) | Semua versi | Semua versi |
| Kanada Barat (Calgary) | Semua versi | Semua versi |
| Tiongkok (Beijing) | Semua versi | Semua versi |
| Tiongkok (Ningxia) | Semua versi | Semua versi |
| Eropa (Frankfurt) | Semua versi | Semua versi |
| Eropa (Irlandia) | Semua versi | Semua versi |
| Eropa (London) | Semua versi | Semua versi |
| Eropa (Milan) | Semua versi | Semua versi |
| Eropa (Paris) | Semua versi | Semua versi |
| Eropa (Spanyol) | Semua versi | Semua versi |
| Eropa (Stockholm) | Semua versi | Semua versi |
| Eropa (Zürich) | Semua versi | Semua versi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Israel (Tel Aviv) | Semua versi | Semua versi |
| Timur Tengah (Bahrain) | Semua versi | Semua versi |
| Timur Tengah (UEA) | Semua versi | Semua versi |
| Amerika Selatan (Sao Paulo) | Semua versi | Semua versi |
| AWS GovCloud (AS-Timur) | Semua versi | Semua versi |
| AWS GovCloud (AS-Barat) | Semua versi | Semua versi |

Wawasan Performa dengan Aurora MySQL dan kueri paralel dinyalakan

Wilayah dan versi mesin berikut tersedia untuk Wawasan Performa dengan Aurora MySQL dan kueri paralel dinyalakan.

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AS Timur (Virginia Utara) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| AS Timur (Ohio) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| AS Barat (California Utara) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| AS Barat (Oregon) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Afrika (Cape Town) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Asia Pasifik (Hyderabad) | Tidak tersedia | Semua versi |
| Asia Pasifik (Jakarta) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Melbourne) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Kanada (Pusat) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Tiongkok (Beijing) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Tiongkok (Ningxia) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Frankfurt) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Irlandia) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (London) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Milan) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Paris) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Spanyol) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Stockholm) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Eropa (Zürich) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Israel (Tel Aviv) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Timur Tengah (UEA) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AWS GovCloud (AS-Timur) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Tidak tersedia | Versi 2.09.0 dan yang lebih tinggi |

## Wawasan Performa dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Wawasan Performa dengan Aurora PostgreSQL.

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 | Aurora PostgreSQL 10 |
|---|---|---|---|---|---|---|---|
| AS Timur (Virginia Utara) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AS Timur (Ohio) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AS Barat (California Utara) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AS Barat (Oregon) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Afrika (Cape Town) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Hong Kong) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 | Aurora PostgreSQL 10 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Hyderabad) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Jakarta) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Malaysia) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Melbourne) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Mumbai) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Osaka) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Seoul) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 | Aurora PostgreSQL 10 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Singapura) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Sydney) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Asia Pasifik (Tokyo) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Kanada (Pusat) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Kanada Barat (Calgary) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Tiongkok (Beijing) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Tiongkok (Ningxia) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Frankfurt) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Irlandia) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (London) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 | Aurora PostgreSQL 10 |
|---|---|---|---|---|---|---|---|
| Eropa (Milan) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Paris) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Spanyol) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Stockholm) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Eropa (Zürich) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Israel (Tel Aviv) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Timur Tengah (Bahrain) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Timur Tengah (UEA) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| Amerika Selatan (Sao Paulo) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 | Aurora PostgreSQL 10 |
|---|---|---|---|---|---|---|---|
| AWS GovCloud (AS-Timur) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |
| AWS GovCloud (AS-Barat) | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi | Semua versi |

### Wawasan Performa dengan Aurora Nirserver

Aurora Serverless v2 mendukung Wawasan Performa untuk semua versi yang kompatibel dengan MySQL dan PostgreSQL. Kami menyarankan Anda mengatur kapasitas minimum minimal 2 unit kapasitas Aurora ()ACUs.

Aurora Serverless v1 tidak mendukung Wawasan Performa.

## Daerah yang Didukung dan mesin Aurora DB untuk integrasi Nol-ETL

Integrasi Amazon Aurora Zero-ETL adalah solusi yang dikelola sepenuhnya untuk membuat data transaksional tersedia di Amazon Redshift atau setelah ditulis ke cluster Aurora. Amazon SageMaker Untuk informasi selengkapnya, lihat Integrasi nol-ETL.

Topik

- Integrasi Aurora MySQL nol-ETL
- Integrasi Aurora PostgreSQL Zero-ETL

### Integrasi Aurora MySQL nol-ETL

Wilayah dan versi mesin berikut tersedia untuk integrasi Aurora MySQL Zero-ETL dengan Amazon Redshift dan. Amazon SageMaker

| Wilayah | Integrasi nol-ETL dengan Amazon Redshift untuk Aurora MySQL versi 3 | Integrasi nol-ETL dengan untuk Aurora Amazon SageMaker MySQL versi 3 |
|---|---|---|
| AS Timur (Virginia Utara) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| AS Timur (Ohio) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| AS Barat (California Utara) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| AS Barat (Oregon) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Afrika (Cape Town) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Jakarta) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Melbourne) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Mumbai) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Osaka) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Asia Pasifik (Seoul) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Asia Pasifik (Singapura) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Asia Pasifik (Sydney) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Kanada (Pusat) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Kanada Barat (Calgary) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Tiongkok (Beijing) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |

| Wilayah | Integrasi nol-ETL dengan Amazon Redshift untuk Aurora MySQL versi 3 | Integrasi nol-ETL dengan untuk Aurora Amazon SageMaker MySQL versi 3 |
|---|---|---|
| Tiongkok (Ningxia) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Eropa (Frankfurt) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Eropa (Irlandia) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Eropa (London) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Eropa (Milan) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Eropa (Paris) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Eropa (Spanyol) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Eropa (Stockholm) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| Eropa (Zürich) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Israel (Tel Aviv) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Timur Tengah (Bahrain) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Timur Tengah (UEA) | Versi 3.05.2 dan lebih tinggi | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Versi 3.05.2 dan lebih tinggi | Versi 3.05.2 dan lebih tinggi |
| AWS GovCloud (AS-Timur) | Tidak tersedia | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia | Tidak tersedia |

## Integrasi Aurora PostgreSQL Zero-ETL

Wilayah dan versi mesin berikut tersedia untuk integrasi Aurora PostgreSQL Zero-ETL dengan Amazon Redshift.

| Wilayah | Aurora PostgreSQL 16 |
| --- | --- |
| AS Timur (Ohio) | Versi 16.4 dan lebih tinggi |
| AS Timur (Virginia Utara) | Versi 16.4 dan lebih tinggi |
| AS Barat (California Utara) | Versi 16.4 dan lebih tinggi |
| AS Barat (Oregon) | Versi 16.4 dan lebih tinggi |
| Afrika (Cape Town) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Osaka) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Seoul) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Singapura) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Sydney) | Versi 16.4 dan lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 16.4 dan lebih tinggi |
| Kanada (Pusat) | Versi 16.4 dan lebih tinggi |
| Kanada Barat (Calgary) | Versi 16.4 dan lebih tinggi |
| Tiongkok (Beijing) | Versi 16.4 dan lebih tinggi |
| Tiongkok (Ningxia) | Versi 16.4 dan lebih tinggi |
| Eropa (Frankfurt) | Versi 16.4 dan lebih tinggi |

| Wilayah | Aurora PostgreSQL 16 |
| --- | --- |
| Eropa (Irlandia) | Versi 16.4 dan lebih tinggi |
| Eropa (London) | Versi 16.4 dan lebih tinggi |
| Eropa (Milan) | Versi 16.4 dan lebih tinggi |
| Eropa (Paris) | Versi 16.4 dan lebih tinggi |
| Eropa (Spanyol) | Versi 16.4 dan lebih tinggi |
| Eropa (Stockholm) | Versi 16.4 dan lebih tinggi |
| Eropa (Zürich) | Versi 16.4 dan lebih tinggi |
| Israel (Tel Aviv) | Versi 16.4 dan lebih tinggi |
| Timur Tengah (Bahrain) | Versi 16.4 dan lebih tinggi |
| Timur Tengah (UEA) | Versi 16.4 dan lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 16.4 dan lebih tinggi |
| AWS GovCloud (AS-Timur) | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia |

## Daerah yang Didukung dan mesin Aurora DB untuk Amazon RDS Proxy

Proksi Amazon RDS adalah proksi basis data yang sepenuhnya terkelola dengan ketersediaan tinggi yang membuat aplikasi lebih dapat ditingkatkan dengan menggabungkan dan berbagi koneksi basis data yang telah dibuat. Untuk informasi selengkapnya tentang Proksi RDS, lihat Proksi Amazon RDS untuk Aurora.

Topik

- Proksi Amazon RDS dengan Aurora MySQL
- Proksi Amazon RDS dengan Aurora PostgreSQL

## Proksi Amazon RDS dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Proksi RDS dengan Aurora MySQL.

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AS Timur (Virginia Utara) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Afrika (Cape Town) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Asia Pasifik (Osaka) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Thailand) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Tiongkok (Beijing) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (London) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Eropa (Milan) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (Paris) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Eropa (Zürich) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Israel (Tel Aviv) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Timur Tengah (UEA) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Meksiko (Tengah) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 3.01.0 dan yang lebih tinggi | Versi 2.07 dan versi 2.11 dan yang lebih tinggi |

## Proksi Amazon RDS dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Proksi RDS dengan Aurora PostgreSQL.

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| AS Timur (Virginia Utara) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| | | | | | | | lebih tinggi |
| Afrika (Cape Town) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Malaysia) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Seoul) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Asia Pasifik (Thailand) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Asia Pasifik (Tokyo) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Tiongkok (Beijing) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Tiongkok (Ningxia) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (London) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Eropa (Milan) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Paris) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Eropa (Zürich) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Israel (Tel Aviv) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Timur Tengah (UEA) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 17 | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 | Aurora PostgreSQL 12 | Aurora PostgreSQL 11 |
|---|---|---|---|---|---|---|---|
| Meksiko (Tengah) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 17.4 dan lebih tinggi | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.4 dan yang lebih tinggi | Versi 12.8 dan yang lebih tinggi | Versi 11.9 dan versi 11.13 dan yang lebih tinggi |

# Daerah yang Didukung dan engine Aurora DB untuk integrasi Secrets Manager

Dengan AWS Secrets Manager, Anda dapat mengganti kredensi hard-code dalam kode Anda, termasuk kata sandi database, dengan panggilan ke API Secrets Manager untuk mengambil rahasia secara terprogram. Untuk informasi selengkapnya tentang Secrets Manager, lihat AWS Secrets Manager Panduan Pengguna.

Anda dapat menentukan Amazon Aurora mengelola kata sandi pengguna master di Secrets Manager untuk klaster DB Aurora. Aurora menghasilkan kata sandi, menyimpannya di Secrets Manager, dan merotasinya secara teratur. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.

Integrasi Secrets Manager tersedia di semua Wilayah AWS.

# Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2

Aurora Serverless v2 adalah fitur penskalaan otomatis sesuai permintaan yang dirancang sebagai pendekatan hemat biaya untuk menjalankan beban kerja yang terputus-putus atau tidak dapat diprediksi di Amazon Aurora. Fitur ini secara otomatis meningkatkan/menurunkan kapasitas sesuai kebutuhan aplikasi Anda. Penskalaan lebih cepat dan lebih granular dibandingkan dengan Aurora Serverless v1. dengan Aurora Serverless v2, setiap cluster dapat berisi instance DB penulis dan beberapa instance DB pembaca. Anda dapat menggabungkan Aurora Serverless v2 dan instance DB tradisional yang disediakan dalam cluster yang sama. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2.

Topik

- Aurora Serverless v2 dengan Aurora MySQL
- Aurora Serverless v2 dengan Aurora PostgreSQL

## Aurora Serverless v2 dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v2 dengan Aurora MySQL.

| Wilayah | Aurora MySQL versi 3 |
| --- | --- |
| AS Timur (Virginia Utara) | Versi 3.02.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 |
| --- | --- |
| AS Timur (Ohio) | Versi 3.02.0 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 3.02.0 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 3.02.0 dan yang lebih tinggi |
| Afrika (Cape Town) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 3.02.3 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 3.04.3, 3.05.2, 3.06.1, 3.07.1, dan lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 3.02.3 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Seoul) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 3.02.0 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 3.02.0 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 3.02.0 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 3.04.0 dan lebih tinggi |
| Tiongkok (Beijing) | Versi 3.02.2 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 3.02.2 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 3.02.0 dan yang lebih tinggi |

| Wilayah | Aurora MySQL versi 3 |
| --- | --- |
| Eropa (Irlandia) | Versi 3.02.0 dan yang lebih tinggi |
| Eropa (London) | Versi 3.02.0 dan yang lebih tinggi |
| Eropa (Milan) | Versi 3.02.0 dan yang lebih tinggi |
| Eropa (Paris) | Versi 3.02.0 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 3.02.3 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 3.02.0 dan yang lebih tinggi |
| Eropa (Zürich) | Versi 3.02.3 dan yang lebih tinggi |
| Israel (Tel Aviv) | Versi 3.02.3 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Versi 3.02.0 dan yang lebih tinggi |
| Timur Tengah (UEA) | Versi 3.02.3 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 3.02.0 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 3.02.2 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 3.02.2 dan yang lebih tinggi |

Batas ACU atas dan bawah untuk Aurora Serverless v2 Kapasitas dapat bervariasi tergantung pada versi mesin Anda. Lihat perinciannya di Aurora Serverless v2 kapasitas.

## Aurora Serverless v2 dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v2 dengan Aurora PostgreSQL.

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 |
|---|---|---|---|---|
| AS Timur (Virginia Utara) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| AS Timur (Ohio) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| AS Barat (California Utara) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| AS Barat (Oregon) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Afrika (Cape Town) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Hyderabad) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.6 dan yang lebih tinggi | Versi 13.9 dan yang lebih tinggi |
| Asia Pasifik (Jakarta) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 16.1 dan lebih tinggi | Versi 15.4 dan yang lebih tinggi | Versi 14.6, 14.9 dan lebih tinggi | Versi 13.9, 13.12 dan lebih tinggi |
| Asia Pasifik (Melbourne) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.6 dan yang lebih tinggi | Versi 13.9 dan yang lebih tinggi |
| Asia Pasifik (Mumbai) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Osaka) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 |
|---|---|---|---|---|
| Asia Pasifik (Seoul) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Singapura) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Sydney) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Kanada (Pusat) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Kanada Barat (Calgary) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.6, 14.8 dan lebih tinggi | Versi 13.9, 13.11 dan lebih tinggi |
| Tiongkok (Beijing) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Tiongkok (Ningxia) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Eropa (Frankfurt) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Eropa (Irlandia) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Eropa (London) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Eropa (Milan) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 |
|---|---|---|---|---|
| Eropa (Paris) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Eropa (Spanyol) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.6 dan yang lebih tinggi | Versi 13.9 dan yang lebih tinggi |
| Eropa (Stockholm) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Eropa (Zürich) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.6 dan yang lebih tinggi | Versi 13.9 dan yang lebih tinggi |
| Israel (Tel Aviv) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.6 dan yang lebih tinggi | Versi 13.9 dan yang lebih tinggi |
| Timur Tengah (Bahrain) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| Timur Tengah (UEA) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.6 dan yang lebih tinggi | Versi 13.9 dan yang lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| AWS GovCloud (AS-Timur) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |
| AWS GovCloud (AS-Barat) | Versi 16.1 dan lebih tinggi | Versi 15.2 dan yang lebih tinggi | Versi 14.3 dan yang lebih tinggi | Versi 13.6 dan yang lebih tinggi |

Batas ACU atas dan bawah untuk Aurora Serverless v2 Kapasitas dapat bervariasi tergantung pada versi mesin Anda. Lihat perinciannya di Aurora Serverless v2 kapasitas.

# Aurora Serverless v1

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Kami sangat menyarankan untuk memutakhirkan Aurora Serverless v1 Cluster DB untuk Aurora Serverless v2 sebelum tanggal tersebut. Upgrade dapat melibatkan perubahan dalam nomor versi utama dari mesin database. Oleh karena itu, penting untuk merencanakan, menguji, dan menerapkan peralihan ini sebelum tanggal. end-of-life Mulai 8 Januari 2025, pelanggan tidak lagi dapat membuat yang baru Aurora Serverless v1 cluster atau instance dengan CLI AWS Management Console atau. Untuk informasi tentang proses migrasi, lihatUpgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2.
> Aurora Serverless v2 sisik lebih cepat dan dengan cara yang lebih granular. Aurora Serverless v2 juga memiliki kompatibilitas lebih dengan fitur Aurora lainnya seperti instans DB pembaca. Anda bisa belajar tentang Aurora Serverless v2 di Menggunakan Aurora Serverless v2.

Aurora Serverless v1 adalah fitur penskalaan otomatis sesuai permintaan yang dirancang sebagai pendekatan hemat biaya untuk menjalankan beban kerja yang terputus-putus atau tidak dapat diprediksi di Amazon Aurora. Fitur ini secara otomatis memulai, mematikan, dan meningkatkan/menurunkan kapasitas, sesuai kebutuhan aplikasi Anda, menggunakan satu instans DB di setiap klaster. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.

Topik

- Aurora Serverless v1 dengan Aurora MySQL
- Aurora Serverless v1 dengan Aurora PostgreSQL

## Aurora Serverless v1 dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v1 dengan Aurora MySQL.

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AS Timur (Virginia Utara) | Tidak tersedia | Versi 2.11.4 |
| AS Timur (Ohio) | Tidak tersedia | Versi 2.11.4 |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| AS Barat (California Utara) | Tidak tersedia | Versi 2.11.4 |
| AS Barat (Oregon) | Tidak tersedia | Versi 2.11.4 |
| Afrika (Cape Town) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hyderabad) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Jakarta) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Mumbai) | Tidak tersedia | Versi 2.11.4 |
| Asia Pasifik (Osaka) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Seoul) | Tidak tersedia | Versi 2.11.4 |
| Asia Pasifik (Singapura) | Tidak tersedia | Versi 2.11.4 |
| Asia Pasifik (Sydney) | Tidak tersedia | Versi 2.11.4 |
| Asia Pasifik (Tokyo) | Tidak tersedia | Versi 2.11.4 |
| Kanada (Pusat) | Tidak tersedia | Versi 2.11.4 |
| Kanada Barat (Calgary) | Tidak tersedia | Tidak tersedia |
| Tiongkok (Beijing) | Tidak tersedia | Tidak tersedia |
| Tiongkok (Ningxia) | Tidak tersedia | Versi 2.11.4 |
| Eropa (Frankfurt) | Tidak tersedia | Versi 2.11.4 |
| Eropa (Irlandia) | Tidak tersedia | Versi 2.11.4 |

| Wilayah | Aurora MySQL versi 3 | Aurora MySQL versi 2 |
|---|---|---|
| Eropa (London) | Tidak tersedia | Versi 2.11.4 |
| Eropa (Milan) | Tidak tersedia | Tidak tersedia |
| Eropa (Paris) | Tidak tersedia | Versi 2.11.4 |
| Eropa (Spanyol) | Tidak tersedia | Tidak tersedia |
| Eropa (Stockholm) | Tidak tersedia | Tidak tersedia |
| Eropa (Zürich) | Tidak tersedia | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Tidak tersedia | Tidak tersedia |
| Timur Tengah (UEA) | Tidak tersedia | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Tidak tersedia | Tidak tersedia |
| AWS GovCloud (AS-Timur) | Tidak tersedia | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia | Tidak tersedia |

## Aurora Serverless v1 dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v1 dengan Aurora PostgreSQL.

| Wilayah | Aurora PostgreSQL 13 |
|---|---|
| AS Timur (Virginia Utara) | Versi 13.12 |
| AS Timur (Ohio) | Versi 13.12 |
| AS Barat (California Utara) | Versi 13.12 |
| AS Barat (Oregon) | Versi 13.12 |

| Wilayah | Aurora PostgreSQL 13 |
| --- | --- |
| Afrika (Cape Town) | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Tidak tersedia |
| Asia Pasifik (Hyderabad) | Tidak tersedia |
| Asia Pasifik (Jakarta) | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia |
| Asia Pasifik (Mumbai) | Versi 13.12 |
| Asia Pasifik (Osaka) | Tidak tersedia |
| Asia Pasifik (Seoul) | Versi 13.12 |
| Asia Pasifik (Singapura) | Versi 13.12 |
| Asia Pasifik (Sydney) | Versi 13.12 |
| Asia Pasifik (Tokyo) | Versi 13.12 |
| Kanada (Pusat) | Versi 13.12 |
| Kanada Barat (Calgary) | Tidak tersedia |
| Tiongkok (Beijing) | Tidak tersedia |
| Tiongkok (Ningxia) | Tidak tersedia |
| Eropa (Frankfurt) | Versi 13.12 |
| Eropa (Irlandia) | Versi 13.12 |
| Eropa (London) | Versi 13.12 |
| Eropa (Milan) | Tidak tersedia |

| Wilayah | Aurora PostgreSQL 13 |
| --- | --- |
| Eropa (Paris) | Versi 13.12 |
| Eropa (Spanyol) | Tidak tersedia |
| Eropa (Stockholm) | Tidak tersedia |
| Eropa (Zürich) | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia |
| Timur Tengah (Bahrain) | Tidak tersedia |
| Timur Tengah (UEA) | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Tidak tersedia |
| AWS GovCloud (AS-Timur) | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia |

## Daerah yang Didukung dan engine Aurora DB untuk RDS Data API

RDS Data API (Data API) menyediakan antarmuka layanan web ke cluster Amazon Aurora DB. Alih-alih mengelola koneksi basis data dari aplikasi klien, Anda dapat menjalankan perintah SQL pada titik akhir HTTPS. Untuk informasi selengkapnya, lihat Menggunakan Amazon RDS Data API.

Topik

- Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan
- API data dengan Aurora MySQL Serverless v2 dan disediakan
- API Data dengan Aurora PostgreSQL Serverless v1
- API Data dengan Aurora MySQL Serverless v1

### Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora PostgreSQL Serverless v2 dan kluster DB yang disediakan.

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 |
|---|---|---|---|---|
| AS Timur (Virginia Utara) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| AS Timur (Ohio) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| AS Barat (California Utara) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| AS Barat (Oregon) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Afrika (Cape Town) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Hong Kong) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Hyderabad) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Jakarta) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Malaysia) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Melbourne) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Mumbai) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Osaka) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 |
|---|---|---|---|---|
| Asia Pasifik (Seoul) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Singapura) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Sydney) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Asia Pasifik (Tokyo) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Kanada (Pusat) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Kanada Barat (Calgary) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Tiongkok (Beijing) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Tiongkok (Ningxia) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Eropa (Frankfurt) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Eropa (Irlandia) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Eropa (London) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Eropa (Milan) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |

| Wilayah | Aurora PostgreSQL 16 | Aurora PostgreSQL 15 | Aurora PostgreSQL 14 | Aurora PostgreSQL 13 |
|---|---|---|---|---|
| Eropa (Paris) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Eropa (Spanyol) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Eropa (Stockholm) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Eropa (Zürich) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Timur Tengah (UEA) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| Amerika Selatan (Sao Paulo) | Versi 16.1 dan lebih tinggi | Versi 15.3 dan lebih tinggi | Versi 14.8 dan lebih tinggi | Versi 13.11 dan lebih tinggi |
| AWS GovCloud (AS-Timur) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia | Tidak tersedia | Tidak tersedia | Tidak tersedia |

## API data dengan Aurora MySQL Serverless v2 dan disediakan

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora MySQL Serverless v2 dan kluster DB yang disediakan.

| Wilayah | Aurora MySQL versi 3 | | | |
|---|---|---|---|---|
| AS Timur (Ohio) | Versi 3.07 dan lebih tinggi | | | |
| AS Timur (Virginia Utara) | Versi 3.07 dan lebih tinggi | | | |
| AS Barat (California Utara) | Versi 3.07 dan lebih tinggi | | | |
| AS Barat (Oregon) | Versi 3.07 dan lebih tinggi | | | |
| Afrika (Cape Town) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Hong Kong) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Hyderabad) | Tidak tersedia | | | |
| Asia Pasifik (Jakarta) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Malaysia) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Melbourne) | Tidak tersedia | | | |
| Asia Pasifik (Mumbai) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Osaka) | Versi 3.07 dan lebih tinggi | | | |

| Wilayah | Aurora MySQL versi 3 | | | |
|---|---|---|---|---|
| Asia Pasifik (Seoul) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Singapura) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Sydney) | Versi 3.07 dan lebih tinggi | | | |
| Asia Pasifik (Tokyo) | Versi 3.07 dan lebih tinggi | | | |
| Kanada (Pusat) | Versi 3.07 dan lebih tinggi | | | |
| Kanada Barat (Calgary) | Tidak tersedia | | | |
| Tiongkok (Beijing) | Tidak tersedia | | | |
| Tiongkok (Ningxia) | Tidak tersedia | | | |
| Eropa (Frankfurt) | Versi 3.07 dan lebih tinggi | | | |
| Eropa (Irlandia) | Versi 3.07 dan lebih tinggi | | | |
| Eropa (London) | Versi 3.07 dan lebih tinggi | | | |
| Eropa (Milan) | Versi 3.07 dan lebih tinggi | | | |

| Wilayah | Aurora MySQL versi 3 | | | |
|---|---|---|---|---|
| Eropa (Paris) | Versi 3.07 dan lebih tinggi | | | |
| Eropa (Spanyol) | Tidak tersedia | | | |
| Eropa (Stockholm) | Versi 3.07 dan lebih tinggi | | | |
| Eropa (Zürich) | Tidak tersedia | | | |
| Israel (Tel Aviv) | Tidak tersedia | | | |
| Timur Tengah (Bahrain) | Versi 3.07 dan lebih tinggi | | | |
| Timur Tengah (UEA) | Versi 3.07 dan lebih tinggi | | | |
| Amerika Selatan (Sao Paulo) | Versi 3.07 dan lebih tinggi | | | |
| AWS GovCloud (AS-Timur) | Tidak tersedia | | | |
| AWS GovCloud (AS-Barat) | Tidak tersedia | | | |

## API Data dengan Aurora PostgreSQL Serverless v1

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora PostgreSQL Serverless v1.

| Wilayah | Aurora PostgreSQL 13 | Aurora PostgreSQL 11 |
|---|---|---|
| AS Timur (Virginia Utara) | versi 13.9 | versi 11.18 |
| AS Timur (Ohio) | versi 13.9 | versi 11.18 |

| Wilayah | Aurora PostgreSQL 13 | Aurora PostgreSQL 11 |
|---|---|---|
| AS Barat (California Utara) | versi 13.9 | versi 11.18 |
| AS Barat (Oregon) | versi 13.9 | versi 11.18 |
| Afrika (Cape Town) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hong Kong) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Hyderabad) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Jakarta) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Mumbai) | versi 13.9 | versi 11.18 |
| Asia Pasifik (Osaka) | Tidak tersedia | Tidak tersedia |
| Asia Pasifik (Seoul) | versi 13.9 | versi 11.18 |
| Asia Pasifik (Singapura) | versi 13.9 | versi 11.18 |
| Asia Pasifik (Sydney) | versi 13.9 | versi 11.18 |
| Asia Pasifik (Tokyo) | versi 13.9 | versi 11.18 |
| Kanada (Pusat) | versi 13.9 | versi 11.18 |
| Tiongkok (Beijing) | Tidak tersedia | Tidak tersedia |
| Tiongkok (Ningxia) | Tidak tersedia | Tidak tersedia |
| Eropa (Frankfurt) | versi 13.9 | versi 11.18 |
| Eropa (Irlandia) | versi 13.9 | versi 11.18 |
| Eropa (London) | versi 13.9 | versi 11.18 |

| Wilayah | Aurora PostgreSQL 13 | Aurora PostgreSQL 11 |
|---|---|---|
| Eropa (Milan) | Tidak tersedia | Tidak tersedia |
| Eropa (Paris) | versi 13.9 | versi 11.18 |
| Eropa (Spanyol) | Tidak tersedia | Tidak tersedia |
| Eropa (Stockholm) | Tidak tersedia | Tidak tersedia |
| Eropa (Zürich) | Tidak tersedia | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia | Tidak tersedia |
| Timur Tengah (Bahrain) | Tidak tersedia | Tidak tersedia |
| Timur Tengah (UEA) | Tidak tersedia | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Tidak tersedia | Tidak tersedia |
| AWS GovCloud (AS-Timur) | Tidak tersedia | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia | Tidak tersedia |

## API Data dengan Aurora MySQL Serverless v1

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora MySQL Serverless v1.

| Wilayah | Aurora MySQL versi 2 |
|---|---|
| AS Timur (Virginia Utara) | Versi: 2.11.3 |
| AS Timur (Ohio) | Versi: 2.11.3 |
| AS Barat (California Utara) | Versi: 2.11.3 |
| AS Barat (Oregon) | Versi: 2.11.3 |
| Afrika (Cape Town) | Tidak tersedia |

| Wilayah | Aurora MySQL versi 2 |
| --- | --- |
| Asia Pasifik (Hong Kong) | Tidak tersedia |
| Asia Pasifik (Hyderabad) | Tidak tersedia |
| Asia Pasifik (Jakarta) | Tidak tersedia |
| Asia Pasifik (Malaysia) | Tidak tersedia |
| Asia Pasifik (Melbourne) | Tidak tersedia |
| Asia Pasifik (Mumbai) | Versi: 2.11.3 |
| Asia Pasifik (Osaka) | Tidak tersedia |
| Asia Pasifik (Seoul) | Versi: 2.11.3 |
| Asia Pasifik (Singapura) | Versi: 2.11.3 |
| Asia Pasifik (Sydney) | Versi: 2.11.3 |
| Asia Pasifik (Tokyo) | Versi: 2.11.3 |
| Kanada (Pusat) | Versi: 2.11.3 |
| Kanada Barat (Calgary) | Tidak tersedia |
| Tiongkok (Beijing) | Tidak tersedia |
| Tiongkok (Ningxia) | Versi: 2.11.3 |
| Eropa (Frankfurt) | Versi: 2.11.3 |
| Eropa (Irlandia) | Versi: 2.11.3 |
| Eropa (London) | Versi: 2.11.3 |
| Eropa (Milan) | Tidak tersedia |
| Eropa (Paris) | Versi: 2.11.3 |

| Wilayah | Aurora MySQL versi 2 |
| --- | --- |
| Eropa (Spanyol) | Tidak tersedia |
| Eropa (Stockholm) | Tidak tersedia |
| Eropa (Zürich) | Tidak tersedia |
| Israel (Tel Aviv) | Tidak tersedia |
| Timur Tengah (Bahrain) | Tidak tersedia |
| Timur Tengah (UEA) | Tidak tersedia |
| Amerika Selatan (Sao Paulo) | Tidak tersedia |
| AWS GovCloud (AS-Timur) | Tidak tersedia |
| AWS GovCloud (AS-Barat) | Tidak tersedia |

## Daerah yang Didukung dan engine Aurora DB untuk patching zero-downtime () ZDP

Peningkatan klaster DB Aurora kemungkinan akan menimbulkan pemadaman layanan saat basis data dinonaktifkan dan sedang ditingkatkan. Secara default, jika Anda memulai peningkatan saat basis data sibuk, Anda kehilangan semua koneksi dan transaksi yang sedang diproses oleh klaster DB. Jika Anda menunggu hingga basis data idle untuk melakukan peningkatan, Anda mungkin harus menunggu lama.

Fitur zero-downtime patching (ZDP) mencoba, dengan upaya terbaik, untuk mempertahankan koneksi klien melalui peningkatan Aurora. Jika ZDP selesai dengan sukses, sesi aplikasi dipertahankan dan mesin database restart saat upgrade sedang berlangsung. Pengaktifan ulang mesin basis data dapat menyebabkan penurunan throughput yang berlangsung selama beberapa detik hingga kira-kira satu menit.

Untuk informasi rinci tentang kondisi dan versi mesin yang ZDP tersedia untuk SQL upgrade Aurora My, lihat. Menggunakan zero-downtime patching

Untuk informasi rinci tentang kondisi dan versi mesin yang ZDP tersedia untuk peningkatan Aurora PostgreSQL, lihat. Peningkatan rilis minor dan patching nol-waktu henti

## Daerah yang Didukung dan engine DB untuk fitur asli mesin Aurora

Mesin basis data Aurora juga mendukung fitur dan fungsionalitas tambahan khusus untuk Aurora. Beberapa fitur asli mesin mungkin memiliki dukungan terbatas atau hak istimewa terbatas untuk mesin, versi, atau Wilayah DB Aurora tertentu.

Topik

- Fitur asli mesin untuk Aurora My SQL
- Fitur asli mesin untuk Aurora Postgre SQL

### Fitur asli mesin untuk Aurora My SQL

Berikut ini adalah fitur asli mesin untuk Aurora My. SQL

- Audit Lanjutan
- Backtrack
- Pertanyaan injeksi kesalahan
- Penerusan tulis dalam klaster
- Kueri paralel

### Fitur asli mesin untuk Aurora Postgre SQL

Berikut ini adalah fitur asli mesin untuk Aurora Postgre. SQL

- Babelfish
- Pertanyaan injeksi kesalahan
- Manajemen rencana kueri

# Koneksi titik akhir Amazon Aurora

Amazon Aurora biasanya menangani sebuah klaster yang terdiri dari beberapa instans DB, bukan instans tunggal. Setiap koneksi ditangani oleh instans DB tertentu. Saat Anda terhubung ke klaster Aurora, nama dan port host yang Anda tentukan akan mengarah ke sebuah handler perantara yang

disebut titik akhir. Aurora menggunakan mekanisme titik akhir untuk menyederhanakan koneksi tersebut. Dengan demikian, Anda tidak perlu melakukan hardcode semua nama host atau menulis logika Anda sendiri untuk menyeimbangkan dan mengalihkan koneksi ketika beberapa instance DB tidak tersedia.

Untuk tugas Aurora tertentu, beberapa instans atau grup instans yang berbeda menjalankan peran yang berbeda. Misalnya, instance utama menangani semua pernyataan bahasa definisi data (DDL) dan bahasa manipulasi data (DML). Hingga 15 Replika Aurora dapat menangani lalu lintas kueri hanya baca.

Topik


- Jenis titik akhir Aurora
- Melihat titik akhir untuk klaster Aurora
- Bagaimana titik akhir Aurora berfungsi dengan ketersediaan tinggi
- Titik akhir cluster untuk Amazon Aurora
- Titik akhir pembaca untuk Amazon Aurora
- Titik akhir instans untuk Amazon Aurora
- Titik akhir khusus untuk Amazon Aurora


## Jenis titik akhir Aurora

Dengan menggunakan titik akhir, Anda dapat memetakan setiap koneksi ke instans atau grup instans yang sesuai berdasarkan kasus penggunaan Anda. Misalnya, untuk melakukan DDL pernyataan, Anda dapat terhubung ke instance mana pun yang merupakan instance utama. Untuk melakukan kueri, Anda dapat terhubung ke titik akhir pembaca, dengan Aurora secara otomatis melakukan penyeimbangan koneksi di antara semua Replika Aurora. Untuk klaster dengan instans DB yang memiliki kapasitas atau konfigurasi berbeda, Anda dapat melakukan koneksi ke titik akhir kustom yang terkait dengan subset instans DB yang berbeda. Untuk diagnosis atau penyetelan, Anda dapat melakukan koneksi ke titik akhir instans tertentu untuk memeriksa detail tentang instans DB tertentu.

Endpoint direpresentasikan sebagai Aurora-spesifik URL yang berisi alamat host dan port. Jenis titik akhir berikut tersedia dari klaster DB Aurora.

Titik akhir klaster

Connect ke instance utama klaster Anda untuk mengembangkan dan menguji aplikasi, dan melakukan transformasi seperti `INSERT` pernyataan danDDL,DML, dan ETL operasi. Temukan

lokasi titik akhir cluster dengan menggunakan AWS Management Console,, atau Amazon AWS CLI RDSAPI, seperti yang dijelaskan dalamMelihat titik akhir untuk klaster Aurora.

Untuk informasi selengkapnya tentang titik akhir klaster, lihatTitik akhir cluster untuk Amazon Aurora.

Titik akhir pembaca

Lakukan kueri. Aurora secara otomatis melakukan penyeimbangan koneksi di antara semua Replika Aurora. Temukan lokasi titik akhir pembaca dengan menggunakan AWS Management Console,, atau Amazon AWS CLI RDSAPI, seperti yang dijelaskan dalamMelihat titik akhir untuk klaster Aurora.

Untuk informasi lebih lanjut tentang titik akhir pembaca, lihatTitik akhir pembaca untuk Amazon Aurora.

Titik akhir instans

Periksa detail tentang instance DB tertentu untuk diagnosis atau penyetelan. Anda dapat menemukan lokasi titik akhir instance untuk setiap instance Anda di AWS Management Console satu-satunya, pada halaman detail instance untuk instance Anda.

Untuk informasi selengkapnya tentang titik akhir instance, lihatTitik akhir instans untuk Amazon Aurora.

Titik akhir kustom

Connect ke subset yang berbeda dari instans DB pada cluster DB. Ini berguna ketika Anda memiliki kapasitas dan konfigurasi instans yang berbeda dalam cluster DB Anda. Temukan lokasi titik akhir kustom dengan menggunakan AWS Management Console,, atau Amazon AWS CLI RDSAPI, seperti yang dijelaskan dalamMelihat titik akhir untuk klaster Aurora.

Untuk informasi selengkapnya tentang titik akhir kustom, lihatTitik akhir khusus untuk Amazon Aurora.

Titik akhir penulis Basis Data Global Aurora

Aurora Global Database memiliki jenis endpoint khusus yang melayani tujuan yang sama dengan titik akhir cluster dari cluster Aurora mandiri. Ini menangani permintaan tulis dan baca. Ketika cluster sekunder menjadi cluster primer baru karena switchover atau failover, Aurora secara otomatis mengalihkan titik akhir ini untuk menunjuk ke titik akhir cluster dari cluster primer baru, di cluster lain. Wilayah AWS Dengan begitu, Anda tidak perlu menyandikan AWS Region ke dalam string koneksi untuk aplikasi Anda, dan Anda tidak perlu mengubah string koneksi saat tata letak

database global berubah. Aurora membuat titik akhir ini saat Anda menyiapkan Database Global Aurora, misalnya dengan memilih Tambah Wilayah untuk klaster Aurora di. AWS Management Console

Untuk informasi tentang bagaimana Anda dapat menggunakan jenis titik akhir ini dengan Aurora Global Database, lihat. Menghubungkan ke Amazon Aurora Global Database

## Melihat titik akhir untuk klaster Aurora

Meskipun Anda hanya dapat menemukan lokasi titik akhir instance pada halaman detail instance di halaman AWS Management Console, Anda dapat menggunakan konsol AWS CLI, atau Amazon RDS API untuk menemukan lokasi cluster, pembaca, dan titik akhir kustom.

Console

Di AWS Management Console, temukan titik akhir cluster, titik akhir pembaca, dan titik akhir kustom apa pun di halaman detail instance untuk klaster Anda. Anda melihat titik akhir instans di halaman detail untuk setiap instans. Saat Anda terhubung, tambahkan nomor port terkait, mengikuti titik dua, ke nama titik akhir yang ditampilkan pada halaman detail.

AWS CLI

Dengan AWS CLI, Anda menemukan penulis, pembaca, dan titik akhir khusus apa pun dalam output describe-db-clustersperintah. Misalnya, perintah berikut menunjukkan atribut endpoint untuk semua cluster di Region Anda saat ini AWS .

```
aws rds describe-db-clusters --query '*[].
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomEndpoints:CustomEndpoints}'
```

Amazon RDS API

Dengan Amazon RDSAPI, Anda mengambil titik akhir dengan memanggil operasi DescribeDBCluster Endpoints.

## Bagaimana titik akhir Aurora berfungsi dengan ketersediaan tinggi

Untuk cluster di mana ketersediaan tinggi penting, gunakan titik akhir cluster untuk koneksi baca/tulis atau tujuan umum dan titik akhir pembaca untuk koneksi hanya-baca. Titik akhir penulis dan pembaca mengelola failover instans basis data dengan lebih baik daripada titik akhir instans. Tidak

seperti titik akhir instans, titik akhir penulis dan titik akhir pembaca mengubah secara otomatis instans basis data yang dihubungi jika instans basis data di klaster menjadi tidak tersedia. Untuk informasi selengkapnya tentang titik akhir cluster dan pembaca, lihat Titik akhir cluster untuk Amazon Aurora danTitik akhir pembaca untuk Amazon Aurora.

Jika instans DB primer dari klaster DB gagal, Aurora secara otomatis melakukan failover ke instans DB primer yang baru. Hal tersebut dilakukan dengan mempromosikan Replika Aurora yang sudah ada ke instans DB primer yang baru, atau membuat instans DB primer yang baru. Jika failover terjadi, Anda dapat menggunakan titik akhir cluster untuk menyambung kembali ke instans DB primer yang baru dipromosikan atau dibuat, atau menggunakan titik akhir pembaca untuk menyambung kembali ke salah satu Replika Aurora di cluster DB. Selama failover, titik akhir pembaca mungkin akan mengarahkan koneksi ke instans DB primer yang baru dari klaster DB untuk waktu yang singkat setelah Replika Aurora dipromosikan menjadi instans DB primer yang baru.

Jika Anda merancang logika aplikasi Anda sendiri untuk mengelola koneksi untuk titik akhir instans, Anda dapat secara manual atau programatis menemukan serangkaian instans DB yang tersedia dalam klaster DB. Gunakan describe-db-clusters AWS CLI perintah atau DescribeDBClustersRDSAPIoperasi untuk menemukan cluster DB dan titik akhir pembaca, instans DB, apakah instans DB adalah pembaca, dan tingkatan promosinya. Anda kemudian dapat mengonfirmasi kelas instansnya setelah failover, dan melakukan koneksi ke titik akhir instans yang sesuai.

Untuk informasi selengkapnya tentang failover, lihat Toleransi kesalahan untuk klaster DB Aurora.

Untuk informasi selengkapnya tentang ketersediaan tinggi di Amazon Aurora, lihat. Ketersediaan yang tinggi untuk Amazon Aurora

## Titik akhir cluster untuk Amazon Aurora

Titik akhir klaster (atau titik akhir penulis) untuk sebuah klaster DB Aurora terhubung ke instans DB primer saat ini untuk klaster DB tersebut. Endpoint ini adalah satu-satunya yang dapat melakukan operasi tulis seperti DDL pernyataan. Oleh karena itu, titik akhir klaster adalah titik akhir tempat Anda melakukan koneksi saat Anda pertama kali mengatur klaster atau ketika klaster Anda hanya berisi satu instans DB.

Setiap klaster DB Aurora memiliki satu titik akhir klaster dan satu instans DB primer.

Anda menggunakan titik akhir cluster untuk semua operasi penulisan di cluster DB, termasuk sisipan, pembaruan, penghapusan, dan perubahan. DDL Anda juga dapat menggunakan titik akhir klaster untuk operasi baca, seperti kueri.

Titik akhir klaster memberikan dukungan failover untuk koneksi baca/tulis ke klaster DB. Jika instans DB primer saat ini pada sebuah klaster DB gagal, Aurora secara otomatis melakukan failover ke instans DB primer yang baru. Selama failover, klaster DB tersebut terus melayani permintaan koneksi ke titik akhir klaster dari instans DB primer yang baru, dengan interupsi layanan yang minimal.

Contoh berikut menggambarkan titik akhir cluster untuk cluster Aurora My DB. SQL

```
mydbcluster.cluster-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Setiap cluster Aurora memiliki satu titik akhir cluster bawaan, yang nama dan atribut lainnya dikelola oleh Aurora. Anda tidak dapat membuat, menghapus, atau mengubah titik akhir jenis ini.

Anda menggunakan titik akhir klaster saat mengelola klaster, melakukan operasi ekstrak, transformasi, load (ETL), atau mengembangkan dan menguji aplikasi. Titik akhir klaster terhubung ke instans primer dari klaster tersebut. Instance utama adalah satu-satunya instans DB di mana Anda dapat membuat tabel dan indeks, menjalankan `INSERT` pernyataan, dan melakukan DML operasi lainnyaDDL.

Alamat IP fisik yang ditunjukkan oleh titik akhir cluster berubah ketika mekanisme failover mempromosikan instance DB baru menjadi read/write primary instance for the cluster. If you use any form of connection pooling or other multiplexing, be prepared to flush or reduce the time-to-live for any cached DNS information. Doing so ensures that you don't try to establish a read/write koneksi ke instans DB yang menjadi tidak tersedia atau sekarang hanya-baca setelah failover.

## Titik akhir pembaca untuk Amazon Aurora

Titik akhir pembaca untuk cluster Aurora DB menyediakan dukungan penyeimbangan koneksi untuk koneksi hanya-baca ke cluster DB. Gunakan titik akhir pembaca untuk operasi baca, seperti kueri. Dengan memproses laporan tersebut di Replika Aurora hanya baca, titik akhir ini mengurangi overhead pada instans primer. Hal ini juga membantu klaster menskalakan kapasitas untuk menangani kueri `SELECT` simultan, berbanding lurus dengan jumlah Replika Aurora di klaster. Setiap klaster DB Aurora memiliki satu titik akhir pembaca.

Jika klaster berisi satu atau lebih Replika Aurora, titik akhir pembaca menyeimbangkan setiap permintaan koneksi di antara Replika Aurora. Dalam hal ini, Anda hanya dapat menjalankan pernyataan hanya baca seperti `SELECT` dalam sesi tersebut. Jika klaster hanya berisi satu instans primer dan tidak ada Replika Aurora, titik akhir pembaca terhubung ke instans primer. Dengan begitu, Anda dapat melakukan operasi baca melalui titik akhir.

Contoh berikut mengilustrasikan titik akhir pembaca untuk sebuah klaster DB Aurora MySQL.

```
mydbcluster.cluster-ro-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Anda menggunakan titik akhir pembaca untuk koneksi hanya baca untuk klaster Aurora Anda. Titik akhir ini menggunakan mekanisme penyeimbangan koneksi untuk membantu klaster Anda menangani beban kerja intensif kueri. Titik akhir pembaca adalah titik akhir yang Anda berikan ke aplikasi yang melakukan pelaporan atau operasi hanya baca lain pada klaster.

Titik akhir pembaca menyeimbangkan koneksi ke Replika Aurora yang tersedia di cluster Aurora DB. Itu tidak menyeimbangkan kueri individu. Jika Anda ingin menyeimbangkan setiap kueri untuk mendistribusikan beban kerja baca untuk kluster DB, buka koneksi baru ke titik akhir pembaca untuk setiap kueri.

Setiap klaster Aurora memiliki satu titik akhir pembaca default, yang nama dan atribut lainnya dikelola oleh Aurora. Anda tidak dapat membuat, menghapus, atau mengubah titik akhir jenis ini.

Jika klaster Anda hanya berisi target utama (instance atau grup pecahan DB) dan tidak ada Replika Aurora, titik akhir pembaca terhubung ke instance utama. Dalam hal ini, Anda dapat melakukan operasi tulis melalui titik akhir ini.

> **Tip**
>
> Melalui Proksi RDS, Anda dapat membuat titik akhir hanya baca tambahan untuk sebuah klaster Aurora. Titik akhir ini melakukan penyeimbangan koneksi yang sama dengan titik akhir pembaca Aurora. Aplikasi dapat terhubung kembali lebih cepat ke titik akhir proksi daripada titik akhir pembaca Aurora jika instans pembaca menjadi tidak tersedia. Titik akhir proksi juga dapat memanfaatkan fitur proksi lainnya seperti multiplexing. Lihat informasi yang lebih lengkap di Menggunakan titik akhir pembaca dengan klaster Aurora.

## Titik akhir instans untuk Amazon Aurora

Titik akhir instans melakukan koneksi ke sebuah instans DB tertentu di dalam klaster Aurora. Setiap instans DB dalam sebuah klaster DB memiliki titik akhir yang unik. Jadi, ada satu titik akhir instans untuk instans DB primer saat ini di klaster DB, dan ada satu titik akhir untuk setiap Replika Aurora di klaster DB tersebut.

Titik akhir instans menyediakan kontrol langsung atas koneksi ke klaster DB untuk skenario yang tidak memungkinkan penggunaan titik akhir klaster atau titik akhir pembaca. Misalnya, aplikasi klien

Anda mungkin memerlukan penyeimbangan koneksi yang lebih halus berdasarkan jenis beban kerja. Dalam hal ini, Anda dapat mengonfigurasi beberapa klien untuk melakukan koneksi ke Replika Aurora yang berbeda dalam klaster DB untuk mendistribusikan beban kerja baca. Untuk contoh yang menggunakan titik akhir instance untuk meningkatkan kecepatan koneksi setelah failover untuk Aurora Postgre, lihat. SQL Failover cepat dengan Amazon Aurora PostgreSQL Untuk contoh yang menggunakan titik akhir instance untuk meningkatkan kecepatan koneksi setelah failover untuk Aurora My, SQL lihat Dukungan failover MariaDB Connector/J - kasus Amazon Aurora.

Contoh berikut menggambarkan titik akhir instance untuk instance DB di cluster Aurora My DB. SQL

```
mydbinstance.c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Setiap instans DB dalam klaster Aurora memiliki titik akhir instans default, yang nama dan atribut lainnya dikelola oleh Aurora. Anda tidak dapat membuat, menghapus, atau memodifikasi titik akhir semacam ini. Anda mungkin akrab dengan titik akhir instance jika Anda menggunakan AmazonRDS. Namun, dengan Aurora, Anda biasanya lebih sering menggunakan titik akhir penulis dan pembaca dibandingkan titik akhir instans.

Dalam operasi day-to-day Aurora, cara utama Anda menggunakan titik akhir instans adalah dengan mendiagnosis masalah kapasitas atau kinerja yang memengaruhi satu instance tertentu dalam klaster Aurora. Saat terhubung dengan instans tertentu, Anda dapat memeriksa variabel status, metrik, dan sebagainya. Dengan melakukan hal tersebut, Anda dapat menentukan apa yang terjadi untuk instans itu yang berbeda dengan apa yang terjadi untuk instans lain dalam klaster.

Dalam kasus penggunaan tingkat lanjut, Anda mungkin mengonfigurasi beberapa instans DB secara berbeda dari yang lain. Dalam hal ini, gunakan titik akhir instans untuk terhubung langsung ke instans yang lebih kecil, lebih besar, atau yang memiliki karakteristik yang berbeda dari yang lain. Selain itu, siapkan prioritas failover agar instans DB khusus ini menjadi pilihan terakhir untuk mengambil alih sebagai instans primer. Kami menyarankan Anda menggunakan titik akhir kustom dan bukan titik akhir instans dalam kasus tersebut. Tindakan tersebut akan menyederhanakan pengelolaan koneksi dan ketersediaan yang tinggi saat Anda menambahkan lebih banyak instans DB ke klaster Anda.

## Titik akhir khusus untuk Amazon Aurora

Titik akhir kustom untuk sebuah klaster Aurora merepresentasikan serangkaian instans DB yang Anda pilih. Saat Anda terhubung ke titik akhir, Aurora melakukan penyeimbangan koneksi dan memilih salah satu instance dalam grup untuk menangani koneksi. Anda menentukan instans mana yang dirujuk oleh titik akhir, dan Anda memutuskan apa tujuan dari titik akhir tersebut.

Sebuah klaster DB Aurora tidak akan memiliki titik akhir kustom sebelum Anda membuatnya. Anda dapat membuat hingga lima titik akhir khusus untuk setiap klaster Aurora yang disediakan atau Aurora Serverless v2 klaster. Anda tidak dapat menggunakan endpoint kustom untuk Aurora Serverless v1 klaster.

Endpoint kustom menyediakan koneksi database yang seimbang berdasarkan kriteria selain kemampuan baca-saja atau baca/tulis dari instans DB. Misalnya, Anda dapat menentukan titik akhir kustom untuk terhubung ke instance yang menggunakan kelas AWS instance tertentu atau grup parameter DB tertentu. Selanjutnya Anda dapat memberi tahu kelompok pengguna tertentu tentang titik akhir kustom ini. Misalnya, Anda dapat mengarahkan pengguna internal ke instans berkapasitas rendah untuk pembuatan laporan atau kueri ad hoc (sekali waktu), dan mengarahkan lalu lintas produksi ke instans berkapasitas tinggi.

Karena koneksinya dapat menuju ke setiap instans DB yang dikaitkan dengan titik akhir kustom, kami menyarankan Anda untuk memastikan bahwa semua instans DB yang berada dalam grup tersebut memiliki beberapa karakteristik serupa. Hal tersebut akan memastikan bahwa performa, kapasitas memori, dsb, konsisten untuk semua orang yang melakukan koneksi ke titik akhir tersebut.

Fitur ini ditujukan bagi pengguna tingkat lanjut dengan beban kerja khusus yang kebutuhannya tidak akan terpenuhi jika semua Replika Aurora di klaster tetap identik. Dengan titik akhir kustom, Anda dapat memprediksi kapasitas instans DB yang digunakan untuk setiap koneksi. Saat Anda menggunakan titik akhir kustom, Anda biasanya tidak menggunakan titik akhir pembaca untuk klaster tersebut.

Contoh berikut menggambarkan titik akhir kustom untuk instance DB di cluster Aurora My DB. SQL

```
myendpoint.cluster-custom-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Anda menggunakan titik akhir kustom untuk menyederhanakan manajemen koneksi saat klaster Anda berisi beberapa instans DB dengan pengaturan kapasitas dan konfigurasi yang berbeda-beda.

Sebelumnya, Anda mungkin telah menggunakan CNAMES mekanisme untuk mengatur alias Layanan Nama Domain (DNS) dari domain Anda sendiri untuk mencapai hasil yang serupa. Dengan menggunakan titik akhir kustom, Anda dapat menghindari memperbarui CNAME catatan saat klaster Anda tumbuh atau menyusut. Endpoint kustom juga berarti bahwa Anda dapat menggunakan koneksi Transport Layer Security/Secure Sockets Layer (TLS/SSL terenkripsi).

Daripada menggunakan satu instans DB untuk setiap tujuan khusus dan melakukan koneksi titik akhir instans, Anda dapat memiliki beberapa grup instans DB khusus. Dalam hal ini, setiap grup memiliki

titik akhir kustom tersendiri. Dengan cara ini, Aurora dapat melakukan penyeimbangan koneksi di antara semua instance yang didedikasikan untuk tugas-tugas seperti pelaporan atau penanganan produksi atau kueri internal. Endpoint kustom mendistribusikan koneksi di seluruh instance secara pasif, menggunakan DNS untuk mengembalikan alamat IP salah satu instance secara acak. Jika salah satu instans DB dalam sebuah grup tidak tersedia, Aurora mengarahkan koneksi titik akhir kustom berikutnya ke salah satu instans DB yang terkait dengan titik akhir yang sama.

Topik


- Pertimbangan untuk titik akhir kustom di Amazon Aurora
- Membuat endpoint kustom
- Melihat titik akhir kustom
- Mengedit titik akhir kustom
- Menghapus titik akhir kustom
- AWS CLI contoh untuk titik akhir khusus untuk Amazon Aurora


## Pertimbangan untuk titik akhir kustom di Amazon Aurora

Gunakan bagian berikut untuk mengelola, menentukan properti, dan menggunakan aturan keanggotaan untuk titik akhir kustom.

Topik


- Mengelola titik akhir kustom
- Menentukan properti untuk titik akhir kustom
- Aturan keanggotaan untuk titik akhir kustom


### Mengelola titik akhir kustom

Karena klaster Aurora yang baru dibuat tidak memiliki titik akhir kustom, Anda harus membuat dan mengelola objek ini sendiri. Anda melakukannya menggunakan AWS Management Console, AWS CLI, atau Amazon RDS API.

> **Note**
>
> Anda juga harus membuat dan mengelola titik akhir kustom untuk klaster Aurora yang dipulihkan dari snapshot. Titik akhir kustom tidak disertakan dalam snapshot. Anda

membuatnya kembali setelah memulihkan, dan memilih nama titik akhir baru jika klaster yang dipulihkan berada di wilayah yang sama dengan yang asli.

Untuk bekerja dengan titik akhir kustom dari AWS Management Console, Anda menavigasi ke halaman detail untuk klaster Aurora Anda dan menggunakan kontrol di bawah bagian Titik Akhir Kustom.

Untuk bekerja dengan titik akhir kustom dari AWS CLI, Anda dapat menggunakan operasi ini:

- create-db-cluster-endpoint
- describe-db-cluster-endpoints
- modify-db-cluster-endpoint
- delete-db-cluster-endpoint

Untuk menangani titik akhir kustom melalui API Amazon RDS, Anda dapat menggunakan fungsi berikut:

- Buat DBCluster Endpoint
- Jelaskan Titik DBCluster Akhir
- Ubah Titik DBCluster Akhir
- Hapus DBCluster Endpoint

Menentukan properti untuk titik akhir kustom

Panjang maksimum untuk nama titik akhir kustom adalah 63 karakter. Format namanya adalah sebagai berikut:

```
endpoint_name.cluster-custom-customer_DNS_identifier.AWS_Region.rds.amazonaws.com
```

Anda tidak dapat menggunakan kembali nama titik akhir kustom yang sama untuk lebih dari satu klaster di Wilayah AWS yang sama. Pengenal DNS kustom adalah pengenal unik yang terkait dengan identitas Anda Akun AWS . Wilayah AWS

Setiap titik akhir kustom memiliki jenis terkait yang menentukan instans DB mana yang memenuhi syarat untuk dikaitkan dengan titik akhir tersebut. Saat ini, jenisnya bisa `READER` atau`ANY`. Pertimbangan berikut berlaku untuk jenis titik akhir kustom:

- Anda tidak dapat memilih jenis titik akhir kustom di AWS Management Console. Semua titik akhir kustom yang Anda buat melalui AWS Management Console memiliki jenis. `ANY`

  Anda dapat mengatur dan memodifikasi tipe endpoint kustom menggunakan AWS CLI atau Amazon RDS API.
- Hanya instans DB pembaca yang dapat menjadi bagian dari titik akhir kustom `READER`.
- Instans DB pembaca dan penulis dapat menjadi bagian dari titik akhir kustom `ANY`. Aurora mengarahkan koneksi ke titik akhir klaster dengan jenis `ANY` ke instans DB terkait mana saja dengan probabilitas yang sama. Jenis `ANY` berlaku untuk klaster yang menggunakan topologi replikasi mana pun.
- Jika Anda mencoba membuat titik akhir kustom dengan jenis yang tidak sesuai berdasarkan konfigurasi replikasi untuk klaster, Aurora akan memunculkan pesan kesalahan.

Aturan keanggotaan untuk titik akhir kustom

Saat Anda menambahkan instans DB ke titik akhir kustom atau menghapusnya dari titik akhir kustom, koneksi apa pun yang ada ke instans DB tersebut tetap aktif.

Anda dapat menetapkan daftar instans DB untuk disertakan dalam, atau dikecualikan dari, sebuah titik akhir kustom. Kami menyebut daftar ini masing-masing sebagai daftar statis dan pengecualian. Anda dapat menggunakan inclusion/exclusion mekanisme untuk membagi lebih lanjut grup instans DB, dan untuk memastikan bahwa kumpulan titik akhir kustom mencakup semua instance DB di cluster. Setiap titik akhir kustom hanya dapat berisi salah satu jenis daftar tersebut.

Dalam AWS Management Console:

- Pilihannya ditunjukkan oleh kotak centang Lampirkan instans masa depan yang ditambahkan ke klaster ini. Jika Anda membiarkan kotak centang ini kosong, titik akhir kustom akan menggunakan daftar statis yang hanya berisi instans DB yang ditentukan pada halaman. Jika Anda memilih kotak centang ini, titik akhir kustom akan menggunakan daftar pengecualian. Dalam hal ini, titik akhir kustom merepresentasikan semua instans DB di dalam klaster (termasuk yang Anda tambahkan di masa mendatang) kecuali yang tidak dipilih di halaman tersebut.
- Konsol tidak memungkinkan Anda menentukan jenis titik akhir. Titik akhir kustom apa pun yang dibuat menggunakan konsol akan memiliki jenis `ANY`.

  Oleh karena itu, Aurora tidak mengubah keanggotaan titik akhir kustom ketika instans DB berganti peran antara penulis dan pembaca karena failover atau promosi.

Di API RDS Amazon AWS CLI dan Amazon:

- Anda dapat menentukan jenis titik akhir. Oleh karena itu, ketika jenis endpoint disetel ke`READER`, keanggotaan endpoint secara otomatis disesuaikan selama failover dan promosi.

  Misalnya, titik akhir kustom dengan tipe `READER` menyertakan Replika Aurora yang kemudian dipromosikan menjadi instance DB penulis. Instans penulis baru ini tidak lagi menjadi bagian dari titik akhir kustom.
- Anda dapat menambahkan anggota individu ke dan menghapusnya dari daftar setelah perannya berubah. Gunakan modify-db-cluster-endpoint AWS CLI perintah atau operasi Modify DBCluster Endpoint API.

Anda dapat mengaitkan instans DB dengan lebih dari satu titik akhir kustom. Misalnya, anggaplah Anda menambahkan instans DB baru ke sebuah klaster, atau Aurora menambahkan instans DB secara otomatis melalui mekanisme penskalaan otomatis. Dalam hal ini, instans DB ditambahkan ke semua titik akhir kustom yang memenuhi syarat. Titik akhir mana instans DB ditambahkan didasarkan pada jenis titik akhir kustom `READER` atau`ANY`, ditambah daftar statis atau pengecualian yang ditentukan untuk setiap titik akhir. Misalnya, jika titik akhir tersebut mencakup daftar statis instans DB, Replika Aurora yang baru tidak ditambahkan ke titik akhir tersebut. Sebaliknya, jika titik akhir tersebut memiliki daftar pengecualian, Replika Aurora yang baru akan ditambahkan ke titik akhir, jika tidak disebutkan dalam daftar pengecualian dan perannya cocok dengan jenis titik akhir kustom tersebut.

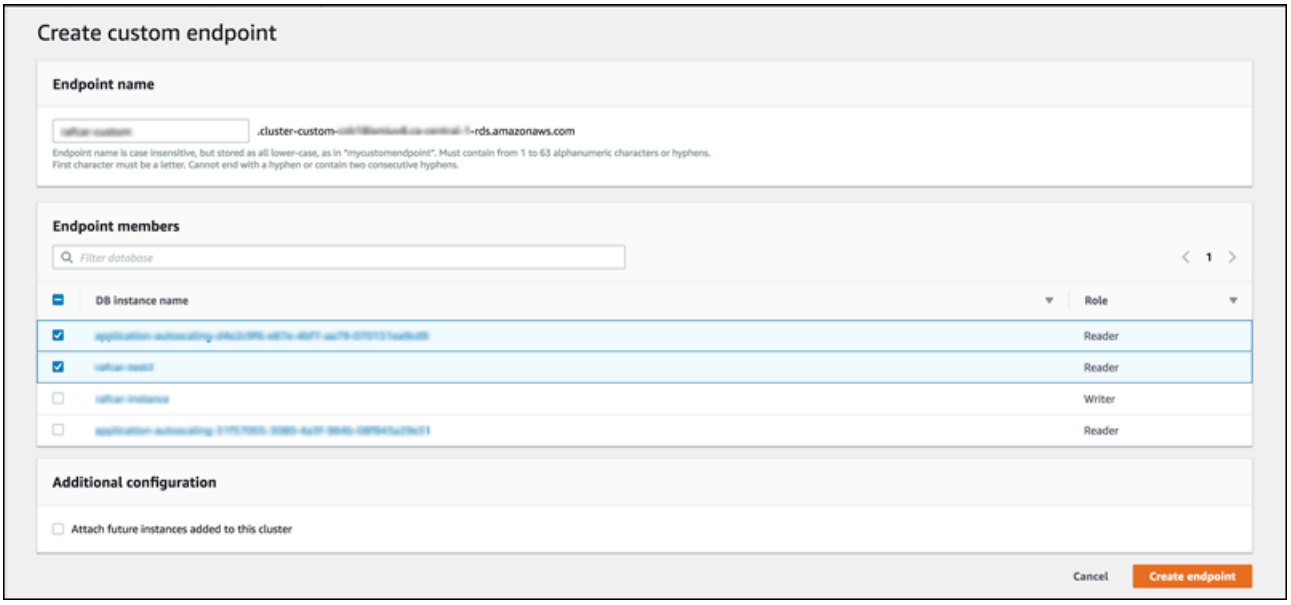
Jika Replika Aurora menjadi tidak tersedia, Replika Aurora ini akan tetap dikaitkan dengan titik akhir kustom. Misalnya, Replika Aurora tetap menjadi bagian dari titik akhir kustom jika tidak berkondisi baik, dihentikan, di-boot ulang, dan seterusnya. Namun, Anda tidak dapat melakukan koneksi ke Replika Aurora melalui titik akhir tersebut hingga tersedia kembali.

## Membuat endpoint kustom

Buat titik akhir kustom menggunakan AWS Management Console, AWS CLI, atau Amazon RDS API.

Konsol

Untuk membuat endpoint kustom dengan AWS Management Console, buka halaman detail cluster dan pilih `Create custom endpoint` tindakan di bagian Endpoints. Pilih nama untuk titik akhir kustom, yang unik untuk ID pengguna dan wilayah Anda. Untuk memilih daftar instans DB yang tetap sama bahkan saat ukuran klaster bertambah, kosongkan kotak centang Lampirkan instans masa depan yang ditambahkan ke klaster ini. Saat Anda memilih kotak centang tersebut, titik akhir kustom akan secara dinamis menambahkan instans baru saat Anda menambahkannya ke klaster.




Anda tidak dapat memilih jenis titik akhir kustom di AWS Management Console. Semua titik akhir kustom yang Anda buat melalui AWS Management Console memiliki jenis. ANY

AWS CLI

Untuk membuat endpoint kustom dengan AWS CLI, jalankan create-db-cluster-endpointperintah.

Perintah berikut membuat titik akhir kustom yang dilampirkan ke klaster tertentu. Awalnya, titik akhir ini dikaitkan dengan semua instans Replika Aurora di dalam klaster. Perintah berikutnya mengaitkannya dengan serangkaian instans DB tertentu dalam klaster.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --endpoint-type reader \
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --static-members instance_name_1 instance_name_2
```

Untuk Windows:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
```

```
  --endpoint-type reader ^
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --static-members instance_name_1 instance_name_2
```

API RDS

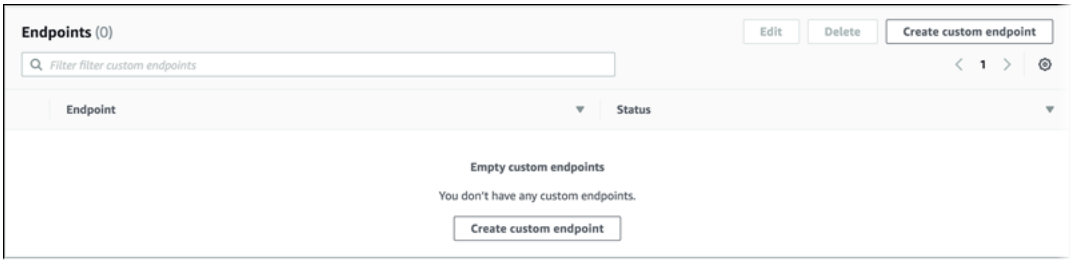
Untuk membuat endpoint kustom dengan RDS API, jalankan operasi Create DBCluster Endpoint.

## Melihat titik akhir kustom

Konsol

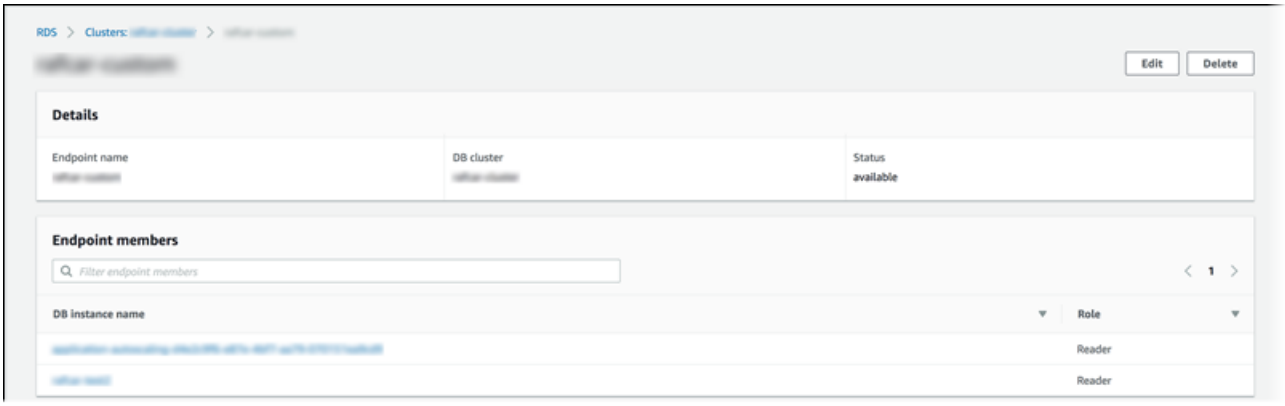
Untuk melihat titik akhir kustom dengan AWS Management Console, buka halaman detail cluster untuk cluster dan lihat di bawah bagian Endpoints. Bagian ini berisi informasi tentang titik akhir kustom saja. Detail untuk titik akhir default tercantum dalam bagian Detail utama. Untuk melihat detail titik akhir kustom tertentu, pilih nama titik akhir guna menampilkan halaman detail untuk titik akhir tersebut.

Tangkapan layar berikut menunjukkan bahwa daftar titik akhir kustom untuk klaster Aurora awalnya kosong.




Setelah Anda membuat beberapa titik akhir kustom untuk klaster tersebut, titik akhir ini ditampilkan di bagian Titik Akhir.

Dengan mengklik halaman detail, akan terlihat instans DB mana yang saat ini dikaitkan dengan titik akhir tersebut.




Untuk melihat detail tambahan tentang apakah instans DB yang baru ditambahkan ke klaster juga secara otomatis ditambahkan ke titik akhir, buka halaman Edit untuk titik akhir.

AWS CLI

Untuk melihat titik akhir kustom dengan AWS CLI, jalankan describe-db-cluster-endpointsperintah.

Perintah berikut akan menampilkan titik akhir kustom yang terkait dengan klaster tertentu di wilayah tertentu. Output-nya mencakup titik akhir default dan titik akhir kustom.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-endpoints --region region_name \
  --db-cluster-identifier cluster_id
```

Untuk Windows:

```
aws rds describe-db-cluster-endpoints --region region_name ^
  --db-cluster-identifier cluster_id
```

Hal berikut menunjukkan beberapa output sampel dari perintah `describe-db-cluster-endpoints`. `EndpointType` yang bernilai `WRITER` atau `READER` menunjukkan titik akhir baca/tulis dan titik akhir hanya baca default untuk klaster tersebut. `EndpointType` yang bernilai `CUSTOM` menunjukkan titik akhir yang Anda buat dan pilih untuk instans DB terkait. Salah satu titik akhir memiliki bidang `StaticMembers` yang terisi, yang menunjukkan bahwa titik akhir tersebut dikaitkan dengan rangkaian instans DB spesifik. Titik akhir lainnya memiliki bidang `ExcludedMembers` yang terisi, menunjukkan bahwa titik akhir ini dikaitkan dengan semua instans DB selain yang tercantum dalam `ExcludedMembers`. Jenis kedua dari titik akhir kustom ini akan bertambah untuk mengakomodasi instans baru saat Anda menambahkannya ke klaster.

```
{
 "DBClusterEndpoints": [
  {
   "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "WRITER"
  },
  {
   "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "READER"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "powers-of-2",
   "ExcludedMembers": [],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "powers-of-2.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-08",
     "custom-endpoint-demo-01",
     "custom-endpoint-demo-02"
   ],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "eight-and-higher",
   "ExcludedMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-02",
     "custom-endpoint-demo-07",
```

```
      "custom-endpoint-demo-05",
      "custom-endpoint-demo-03",
      "custom-endpoint-demo-06",
      "custom-endpoint-demo-01"
    ],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "available",
    "EndpointType": "CUSTOM",
    "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
}
```

RDS API

Untuk melihat titik akhir kustom dengan RDSAPI, jalankan operasi D escribeDBCluster Endpoints.html.

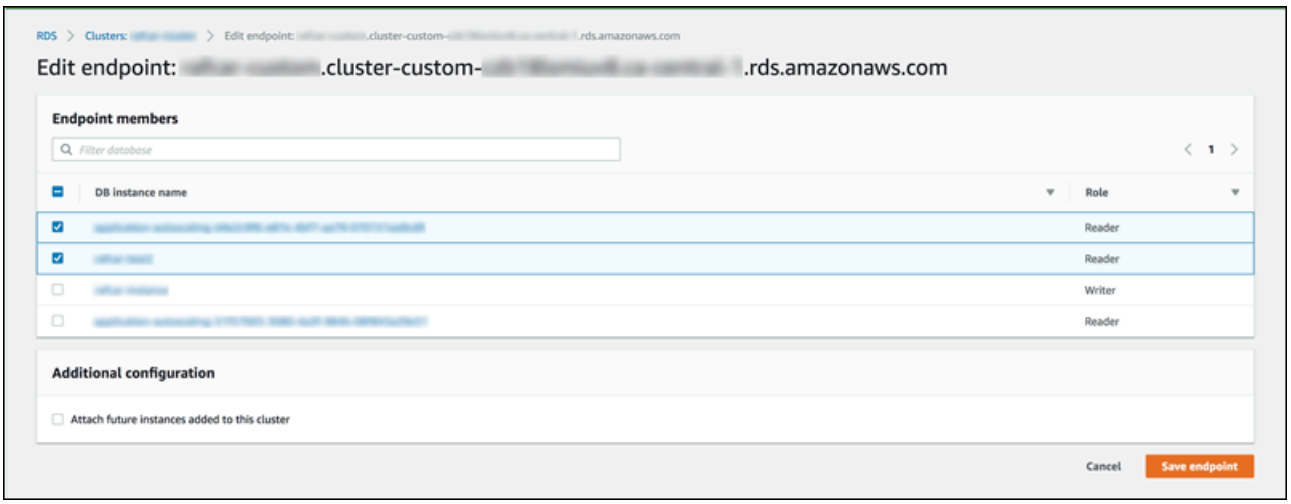
## Mengedit titik akhir kustom

Anda dapat mengubah mengedit titik akhir kustom untuk mengubah instans DB mana yang dikaitkan dengan titik akhir tersebut. Anda juga dapat mengubah titik akhir di antara daftar statis dan daftar pengecualian. Jika Anda membutuhkan detail selengkapnya tentang properti titik akhir ini, lihat Aturan keanggotaan untuk titik akhir kustom.

Anda dapat terus terhubung ke dan menggunakan titik akhir kustom saat perubahan dari tindakan edit sedang berlangsung.

Konsol

Untuk mengedit titik akhir kustom dengan AWS Management Console, Anda dapat memilih titik akhir pada halaman detail cluster, atau membuka halaman detail untuk titik akhir, dan memilih tindakan Edit.




AWS CLI

Untuk mengedit titik akhir kustom dengan AWS CLI, jalankan modify-db-cluster-endpointperintah.

Perintah berikut akan mengubah rangkaian instans DB yang diterapkan ke titik akhir kustom dan secara opsional beralih di antara perilaku daftar statis atau daftar pengecualian. Parameter `--static-members` dan `--excluded-members` mengambil daftar pengidentifikasi instans DB yang dipisahkan oleh spasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --excluded-members db-instance-id-4 db-instance-id-5 \
  --region region_name
```

Untuk Windows:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
  --region region_name
```

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --excluded-members db-instance-id-4 db-instance-id-5 ^
  --region region_name
```

RDS API

Untuk mengedit titik akhir kustom dengan RDSAPI, jalankan operasi M odifyDBCluster Endpoint.html.

## Menghapus titik akhir kustom

Hapus titik akhir kustom menggunakan AWS Management Console, AWS CLI, atau Amazon RDSAPI.

Konsol

Untuk menghapus titik akhir kustom dengan AWS Management Console, buka halaman detail cluster, pilih titik akhir kustom yang sesuai, dan pilih tindakan Hapus.

AWS CLI

Untuk menghapus titik akhir kustom dengan AWS CLI, jalankan delete-db-cluster-endpointperintah.

Perintah berikut akan menghapus titik akhir kustom. Anda tidak perlu menentukan klaster terkait, tetapi Anda harus menentukan wilayah.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 \
  --region region_name
```

Untuk Windows:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 ^
```

```
  --region region_name
```

RDS API

Untuk menghapus titik akhir kustom dengan RDSAPI, jalankan operasi D eleteDBCluster Endpoint.

## AWS CLI contoh untuk titik akhir khusus untuk Amazon Aurora

Tutorial berikut menggunakan AWS CLI contoh dengan sintaks shell Unix untuk menunjukkan bagaimana Anda dapat mendefinisikan cluster dengan beberapa instance DB “kecil” dan beberapa instance DB “besar”, dan membuat titik akhir khusus untuk terhubung ke setiap set instance DB. Untuk menjalankan perintah serupa pada sistem Anda sendiri, Anda harus cukup akrab dengan dasar-dasar bekerja dengan cluster Aurora dan AWS CLI penggunaan untuk menyediakan nilai Anda sendiri untuk parameter seperti wilayah, grup subnet, dan grup keamanan. VPC

Contoh ini menunjukkan langkah pengaturan awal: membuat klaster Aurora dan menambahkan instans DB ke dalamnya. Ini adalah klaster heterogen, yang berarti tidak semua instans DB memiliki kapasitas yang sama. Sebagian besar instance menggunakan kelas AWS instancedb.r4.4xlarge, tetapi dua instance DB terakhir menggunakan. db.r4.16xlarge Masing-masing create-db-instance perintah sampel ini mencetak outputnya ke layar dan menyimpan salinan file JSON dalam file untuk pemeriksaan nanti.

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.04.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION

for i in 01 02 03 04 05 06 07 08
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done

for i in 09 10
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
```

```
    --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.16xlarge \
    --region $REGION \
    | tee custom-endpoint-demo-${i}.json
done
```

Instans yang lebih besar dicadangkan untuk jenis kueri pelaporan khusus. Agar instans tersebut tidak dipromosikan menjadi instans primer, contoh berikut ini mengubah tingkat promosinya ke prioritas terendah. Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

```
for i in 09 10
do
  aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --region $REGION --promotion-tier 15
done
```

Misalnya, Anda ingin menggunakan dua instans “lebih besar” hanya untuk kueri yang membutuhkan banyak sumber daya. Untuk melakukannya, Anda dapat membuat titik akhir hanya baca kustom. Kemudian Anda dapat menambahkan daftar statis anggota sehingga titik akhir hanya terhubung ke instans DB tersebut. Instans tersebut sudah berada di tingkat promosi terendah, sehingga tidak mungkin dipromosikan menjadi instans primer. Jika salah satunya dipromosikan menjadi instans utama, instans tersebut menjadi tidak dapat dijangkau melalui titik akhir ini karena kita menetapkan jenis READER dan bukan jenis ANY.

Contoh berikut menunjukkan perintah create and modify endpoint, dan contoh JSON output yang menunjukkan status awal dan modifikasi dari endpoint kustom.

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
```

```
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

Titik akhir READER default untuk klaster dapat terhubung ke instans DB “kecil” atau “besar”, sehingga tidak praktis untuk memprediksi performa dan skalabilitas kueri ketika klaster menjadi sibuk. Untuk membagi beban kerja secara merata di antara sejumlah rangkaian instans DB, Anda dapat mengabaikan titik akhir READER default dan membuat titik akhir kustom kedua yang terhubung ke semua instans DB lainnya. Contoh berikut melakukannya dengan membuat titik akhir kustom, lalu menambahkan daftar pengecualian. Instans DB lain yang Anda tambahkan ke klaster di masa mendatang akan secara otomatis ditambahkan ke titik akhir ini. Jenis ANY berarti bahwa titik akhir ini terkait dengan total delapan instans: instans primer dan tujuh Replika Aurora lainnya. Jika contohnya menggunakan jenis READER, titik akhir kustom hanya akan dikaitkan dengan tujuh Replika Aurora.

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-
endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:c7tj4example:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

Contoh berikut memeriksa status titik akhir untuk klaster ini. Cluster masih memiliki titik akhir cluster aslinya, dengan `EndPointType` of`WRITER`, yang masih akan Anda gunakan untuk administrasiETL, dan operasi penulisan lainnya. Klaster ini masih memiliki titik akhir `READER` aslinya, yang tidak akan

Anda gunakan karena setiap koneksi yang diterimanya dapat diarahkan ke instans DB "kecil" atau "besar". Dengan titik akhir kustom, perilaku tersebut menjadi dapat diprediksi, dengan koneksi yang dipastikan akan menggunakan salah satu instans DB “kecil” atau “besar” berdasarkan titik akhir yang Anda tentukan.

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "small-instances",
            "Status": "modifying"
        },
        {
            "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
```

```
            "CustomEndpointType": "READER",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
            "ExcludedMembers": [],
            "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [
                "custom-endpoint-demo-10",
                "custom-endpoint-demo-09"
            ],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "big-instances",
            "Status": "available"
        }
    ]
}
```

Contoh terakhir menunjukkan koneksi basis data berurutan ke titik akhir kustom terhubung dengan berbagai instans DB dalam klaster Aurora. Titik akhir `small-instances` selalu terhubung ke instans DB `db.r4.4xlarge`, yang merupakan host bernomor lebih rendah dalam klaster ini.

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-02 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-07 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
```

```
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-01 |
+--------------------------+
```

Titik akhir `big-instances` selalu terhubung ke instans DB `db.r4.16xlarge`, yang merupakan dua host dengan nomor tertinggi dalam klaster ini.

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-10 |
+--------------------------+

$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-09 |
+--------------------------+
```

# Kelas instans Amazon Aurora DB

Kelas instans DB menentukan komputasi dan kapasitas memori instans DB Amazon Aurora. Kelas instans DB yang Anda butuhkan tergantung pada kebutuhan daya dan memori pemrosesan Anda.

Sebuah kelas instans DB terdiri dari jenis dan ukuran kelas instans DB. Misalnya, db.r6g adalah tipe kelas instans DB yang dioptimalkan memori yang didukung oleh prosesor Graviton2. AWS Dalam jenis kelas instans db.r6g, db.r6g.2xlarge adalah kelas instans DB. Ukuran kelas ini adalah 2xlarge.

Untuk informasi selengkapnya tentang harga kelas instans, lihat Harga Amazon RDS.

Untuk informasi selengkapnya tentang jenis kelas instans DB, engine DB yang didukung Wilayah AWS, didukung, atau spesifikasi perangkat keras untuk kelas instans DB, lihat bagian berikut.

Topik


- Jenis kelas instans DB
- Mesin DB yang didukung untuk kelas instans DB
- Menentukan dukungan kelas instans DB di Wilayah AWS
- Spesifikasi perangkat keras kelas instans DB untuk Aurora


## Jenis kelas instans DB

Amazon Aurora mendukung kelas instans DB untuk kasus penggunaan berikut:

- Aurora Serverless v2
- Memori yang dioptimalkan
- Performa yang dapat melonjak
- Optimized Reads

Untuk informasi selengkapnya tentang jenis EC2 instans Amazon, lihat Jenis instans dalam EC2 dokumentasi Amazon.

### Jenis kelas instans Aurora Serverless v2

Jenis Aurora Serverless v2 berikut tersedia:

- db.serverless – Jenis kelas instans DB khusus yang digunakan oleh Aurora Serverless v2. Aurora menyesuaikan komputasi, memori, dan sumber daya jaringan secara dinamis saat beban kerja berubah. Untuk detail penggunaan, lihat Menggunakan Aurora Serverless v2.

## Jenis kelas instance yang dioptimalkan untuk memori

Rangkaian X dengan memori yang dioptimalkan mendukung kelas instans berikut:

- db.x2g - Kelas instans yang dioptimalkan untuk aplikasi intensif memori dan didukung oleh prosesor Graviton2. AWS Kelas instans ini menawarkan biaya rendah per GiB memori.

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton2. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.

Rangkaian R dengan memori yang dioptimalkan mendukung jenis kelas instans berikut:

- db.r8g - Kelas instans yang didukung oleh prosesor Graviton4. AWS Kelas instans ini ideal untuk menjalankan beban kerja yang memerlukan banyak memori . Instans ini menawarkan ukuran instans yang lebih besar dengan v CPUs dan memori hingga 3x lebih banyak daripada instans db.r7g berbasis Graviton3 generasi ketujuh AWS .

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton4. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.r7g - Kelas instance yang didukung oleh prosesor Graviton3. AWS Kelas instans ini ideal untuk menjalankan beban kerja yang memerlukan banyak memori .

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton3. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.r7i — Kelas instans yang didukung oleh prosesor Intel Xeon Scalable Generasi ke-4. Kelas instance ini bersertifikat SAP dan ideal untuk menjalankan beban kerja intensif memori . Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor Intel Xeon Scalable Generasi ke-4. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.

- db.r6g - Kelas instance yang didukung oleh prosesor Graviton2. AWS Kelas instans ini ideal untuk menjalankan beban kerja yang memerlukan banyak memori tinggi dan latensi rendah.

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton2. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.r6i – Kelas instans yang didukung oleh prosesor Intel Xeon Scalable Generasi ke-3. Kelas instance ini bersertifikat SAP dan sangat cocok untuk beban kerja intensif memori .
- db.r5 – Kelas instans yang dioptimalkan untuk aplikasi yang memakan banyak memori. Kelas instans ini menawarkan peningkatan performa jaringan dan Amazon Elastic Block Store (Amazon EBS). Mereka didukung oleh Sistem AWS Nitro, kombinasi perangkat keras khusus dan hypervisor ringan.
- db.r4 - Kelas instance ini hanya didukung untuk Aurora MySQL 2.x dan Aurora PostgreSQL 11 dan 12 versi. Untuk semua cluster Aurora DB yang menggunakan kelas instans db.r4 DB, kami sarankan Anda meningkatkan ke kelas instans generasi yang lebih tinggi sesegera mungkin.

  Kelas instans db.r4 tidak tersedia untuk konfigurasi penyimpanan klaster Aurora I/O-Optimized.

## Jenis kelas instans performa yang dapat melonjak

Jenis kelas instans DB performa yang dapat melonjak berikut tersedia:

- db.t4g - Kelas instance tujuan umum yang didukung oleh prosesor Graviton2 berbasis ARM. AWS Kelas instans ini memberikan performa harga yang lebih baik daripada kelas instans DB performa yang dapat melonjak sebelumnya untuk serangkaian beban kerja tujuan umum yang dapat melonjak. Instans Amazon RDS db.t4g dikonfigurasi untuk mode Tidak Terbatas. Artinya, instans tersebut dapat melampaui garis dasar dalam jangka waktu 24 jam dengan biaya tambahan.

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton2. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.t3 – Kelas instans yang memberikan tingkat performa dasar, dengan kemampuan untuk melonjak hingga penggunaan CPU penuh. Instans db.t3 dikonfigurasi untuk mode Tidak Terbatas. Kelas instans ini memberikan kapasitas komputasi yang lebih besar dibandingkan kelas instans db.t2 sebelumnya. Produk ini didukung oleh Nitro System AWS , kombinasi perangkat keras khusus dan hypervisor ringan. Sebaiknya gunakan kelas instans ini hanya untuk server pengujian dan pengembangan, atau server non-produksi lainnya.

- db.t2 – Kelas instans yang memberikan tingkat performa dasar, dengan kemampuan untuk melonjak hingga penggunaan CPU penuh. Instans db.t2 dikonfigurasi untuk mode Tidak Terbatas. Sebaiknya gunakan kelas instans ini hanya untuk server pengujian dan pengembangan, atau server non-produksi lainnya.

  Kelas instans db.t2 tidak tersedia untuk konfigurasi penyimpanan klaster Aurora I/O-Optimized.

> **Note**
>
> Sebaiknya gunakan kelas instans DB T hanya untuk server pengembangan, pengujian, atau non-produksi lainnya. Rekomendasi lebih mendetail untuk kelas instans T dapat dilihat di Menggunakan kelas instans T untuk pengembangan dan pengujian.

Untuk spesifikasi perangkat keras kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

### Dioptimalkan Membaca tipe kelas instance

Jenis kelas instans Optimized Reads berikut tersedia:

- db.r6gd - Kelas instans yang didukung oleh prosesor Graviton2. AWS Kelas instans ini ideal untuk menjalankan beban kerja intensif memori dan menawarkan penyimpanan tingkat blok SSD NVMe berbasis lokal untuk aplikasi yang membutuhkan penyimpanan lokal berkecepatan tinggi dan latensi rendah.
- db.r6id – Kelas instans yang didukung prosesor Intel Xeon Scalable Generasi ke-3. Kelas instans ini bersertifikat SAP dan ideal untuk beban kerja yang menggunakan banyak memori. Mereka menawarkan memori maksimum 1 TiB dan hingga 7, 6 TB penyimpanan SSD berbasis langsung NVMe.

## Mesin DB yang didukung untuk kelas instans DB

Tabel berikut menunjukkan kelas instans DB yang didukung untuk mesin Amazon Aurora DB.

db.tanpa server — Aurora Serverless v2 kelas instance dengan penskalaan kapasitas otomatis

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.serverless | Lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2. | Lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2. |

db.x2g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton2 AWS

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.x2g.16xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.x2g.12xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.x2g.8xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.x2g.4xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.x2g.2xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.x2g.xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.x2g.large | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |

db.r6gd - Kelas instance Reads yang Dioptimalkan yang didukung oleh prosesor Graviton2 AWS

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r6gd.16xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |
| db.r6gd.12xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |
| db.r6gd.8xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |
| db.r6gd.4xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |
| db.r6gd.2xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |
| db.r6gd.xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |

db.r6id – Kelas instans Optimized Reads

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r6id.32xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |
| db.r6id.24xlarge | Tidak | 16.1 dan lebih tinggi, 15.4 dan lebih tinggi, 14.9 dan lebih tinggi |

db.r8g — kelas instans yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton4 AWS

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r8g.48xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.24xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.16xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.12xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.8xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.4xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r8g.2xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r8g.large | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |

db.r7g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton3 AWS

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r7g.16xlarge | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |
| db.r7g.12xlarge | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |
| db.r7g.8xlarge | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |
| db.r7g.4xlarge | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
| --- | --- | --- |
| db.r7g.2xlarge | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |
| db.r7g.xlarge | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |
| db.r7g.large | 2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.7 dan lebih tinggi, 13.10 dan lebih tinggi |

db.r7i - kelas instance yang dioptimalkan untuk memori

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
| --- | --- | --- |
| db.r7i.48xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.24xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.16xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.12xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.8xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r7i.4xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.2xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.xlarge | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |
| db.r7i.large | 3.08.0 dan lebih tinggi | 16.3 dan lebih tinggi, 15.7 dan lebih tinggi, 14.12 dan lebih tinggi, 13.15 dan lebih tinggi |

db.r6g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton2 AWS

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r6g.16xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.r6g.12xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.r6g.8xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r6g.4xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.r6g.2xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.r6g.xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |
| db.r6g.large | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.8 dan lebih tinggi, 11.9, 11.12 dan lebih tinggi |

db.r6i – kelas instans dengan memori yang dioptimalkan

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r6i.32xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.24xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.16xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r6i.12xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.8xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.4xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.2xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.xlarge | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |
| db.r6i.large | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13,5 dan lebih tinggi, 12.9 dan lebih tinggi |

db.r5 – kelas instans dengan memori yang dioptimalkan

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r5.24xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |
| db.r5.16xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |
| db.r5.12xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |
| db.r5.8xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r5.4xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |
| db.r5.2xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |
| db.r5.xlarge | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |
| db.r5.large | Semua versi yang tersedia saat ini | Semua versi yang tersedia saat ini |

db.r4 – kelas instans dengan memori yang dioptimalkan

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.r4.16xlarge | Semua versi 2.x; tidak didukung dalam versi 3.x | Semua didukung 11 dan 12 versi |
| db.r4.8xlarge | Semua versi 2.x; tidak didukung dalam versi 3.x | Semua didukung 11 dan 12 versi |
| db.r4.4xlarge | Semua versi 2.x; tidak didukung dalam versi 3.x | Semua didukung 11 dan 12 versi |
| db.r4.2xlarge | Semua versi 2.x; tidak didukung dalam versi 3.x | Semua didukung 11 dan 12 versi |
| db.r4.xlarge | Semua versi 2.x; tidak didukung dalam versi 3.x | Semua didukung 11 dan 12 versi |
| db.r4.large | Semua versi 2.x; tidak didukung dalam versi 3.x | Semua didukung 11 dan 12 versi |

db.t4g — kelas instance berkinerja pecah yang ditenagai oleh prosesor Graviton2 AWS

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.t4g.2xlarge | Tidak | Tidak |

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.t4g.xlarge | Tidak | Tidak |
| db.t4g.large | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.7 dan lebih tinggi, 11.12 dan lebih tinggi |
| db.t4g.medium | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.7 dan lebih tinggi, 11.12 dan lebih tinggi |
| db.t4g.small | Tidak | Tidak |

db.t3 – kelas instans performa yang dapat melonjak

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.t3.2xlarge | Tidak | Tidak |
| db.t3.xlarge | Tidak | Tidak |
| db.t3.large | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.7 dan lebih tinggi, 11.12 dan lebih tinggi |
| db.t3.medium | Semua versi yang tersedia saat ini | 16.1 dan lebih tinggi, 15.2 dan lebih tinggi, 14.3 dan lebih tinggi, 13.3 dan lebih tinggi, 12.7 dan lebih tinggi, 11.12 dan lebih tinggi |
| db.t3.small | Semua versi 2.x; tidak didukung dalam versi 3.x | Tidak |
| db.t3.micro | Tidak | Tidak |

db.t2 – kelas instans performa yang dapat melonjak

| Kelas instans | Aurora Saya SQL | Aurora Postgre SQL |
|---|---|---|
| db.t2.medium | Semua versi 2.x; tidak didukung dalam versi 3.x | Tidak |
| db.t2.small | Semua versi 2.x; tidak didukung dalam versi 3.x | Tidak |

## Menentukan dukungan kelas instans DB di Wilayah AWS

Untuk menentukan kelas instans DB yang didukung oleh mesin DB di Wilayah AWS tertentu, Anda dapat menggunakan beberapa pendekatan. Anda dapat menggunakan AWS Management Console, halaman RDSHarga Amazon, atau AWS CLI perintah describe-orderable-db-instance-options.

> **Note**
>
> Ketika Anda melakukan operasi dengan AWS Management Console, secara otomatis menampilkan kelas instans DB yang didukung untuk mesin DB tertentu, versi mesin DB, dan Wilayah AWS. Contoh operasi yang dapat Anda lakukan termasuk membuat dan mengubah instans DB.

Daftar Isi


- Menggunakan halaman RDS harga Amazon untuk menentukan dukungan kelas instans DB di Wilayah AWS
- Menggunakan AWS CLI untuk menentukan dukungan kelas instans DB di Wilayah AWS
  - Menyusun daftar kelas instans DB yang didukung oleh versi mesin DB tertentu di Wilayah AWS
  - Menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS


### Menggunakan halaman RDS harga Amazon untuk menentukan dukungan kelas instans DB di Wilayah AWS

Anda dapat menggunakan halaman Harga Amazon Aurora untuk menentukan kelas instans DB yang didukung oleh masing-masing mesin DB di Wilayah AWS tertentu.

Untuk menggunakan halaman harga guna menentukan kelas instans DB yang didukung oleh masing-masing mesin DB di sebuah Wilayah

1. Buka Harga Amazon Aurora.
2. Pilih mesin Amazon Aurora di bagian Kalkulator Harga AWS .
3. Di Pilih Wilayah, pilih Wilayah AWS.
4. Di Opsi Konfigurasi Klaster, pilih opsi konfigurasi.
5. Gunakan bagian instans yang kompatibel untuk melihat kelas instans DB yang didukung.
6. (Opsional) Pilih opsi lain di kalkulator, lalu pilih Simpan dan lihat ringkasan atau Simpan dan tambahkan layanan.

## Menggunakan AWS CLI untuk menentukan dukungan kelas instans DB di Wilayah AWS

Anda dapat menggunakan AWS CLI untuk menentukan kelas instans DB mana yang didukung untuk mesin DB tertentu dan versi mesin DB dalam file Wilayah AWS.

Untuk menggunakan AWS CLI contoh berikut, masukkan nilai yang valid untuk mesin DB, versi mesin DB, kelas instans DB, dan Wilayah AWS. Tabel berikut menunjukkan nilai-nilai mesin DB yang valid.

| Nama mesin | Nilai mesin dalam CLI perintah | Informasi selengkapnya tentang versi |
|---|---|---|
| Aurora saya yang kompatibel dengan SQL 5.7 dan 8.0 yang kompatibel | `aurora-mysql` | Pembaruan mesin database untuk Amazon Aurora SQL Versi saya 2 dan pembaruan mesin Database untuk Amazon Aurora Versi SQL saya 3 di Catatan Rilis untuk Aurora My SQL |
| Aurora Postgre SQL | `aurora-postgresql` | Catatan Rilis untuk Aurora Postgre SQL |

Untuk informasi tentang Wilayah AWS nama, lihatAWS Daerah.

Contoh berikut menunjukkan bagaimana menentukan dukungan kelas instance DB dalam Wilayah AWS menggunakan AWS CLI perintah describe-orderable-db-instance-options.

Topik

- Menyusun daftar kelas instans DB yang didukung oleh versi mesin DB tertentu di Wilayah AWS
- Menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS

### Menyusun daftar kelas instans DB yang didukung oleh versi mesin DB tertentu di Wilayah AWS

Untuk daftar kelas instans DB yang didukung oleh versi mesin DB tertentu dalam Wilayah AWS, jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Untuk Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

Output juga menunjukkan mode mesin yang didukung untuk setiap kelas instans DB.

Misalnya, perintah berikut mencantumkan kelas instans DB yang didukung untuk versi 13.6 dari mesin Aurora SQL Postgre DB di US East (Virginia N.).

Untuk Linux, macOS, atau Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
```

```
    --output table \
    --region us-east-1
```

Untuk Windows:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}"  ^
    --output table ^
    --region us-east-1
```

Menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS

Untuk menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS, jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Untuk Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

Output juga menunjukkan mode mesin yang didukung untuk setiap versi mesin DB.

Misalnya, perintah berikut mencantumkan versi mesin DB dari mesin Aurora Postgre SQL DB yang mendukung kelas instans db.r5.large DB di AS Timur (Virginia N.).

Untuk Linux, macOS, atau Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Untuk Windows:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region us-east-1
```

## Spesifikasi perangkat keras kelas instans DB untuk Aurora

Dalam tabel  di bagian ini, Anda dapat menemukan detail perangkat keras tentang kelas instans Amazon RDS DB untuk Aurora.

Untuk informasi tentang dukungan mesin DB Aurora untuk setiap kelas instans DB, lihat Mesin DB yang didukung untuk kelas instans DB.

Topik

- Terminologi perangkat keras untuk kelas instans DB untuk Aurora
- Spesifikasi perangkat keras untuk kelas instans yang dioptimalkan memori
- Spesifikasi perangkat keras untuk kelas instans performa yang dapat dibobol

### Terminologi perangkat keras untuk kelas instans DB untuk Aurora

Terminologi berikut digunakan untuk menjelaskan spesifikasi perangkat keras untuk kelas instans DB:

vCPU

Jumlah unit pemrosesan pusat virtual (CPUs). CPU virtual adalah unit kapasitas yang dapat Anda gunakan untuk membandingkan kelas instans DB. Alih-alih membeli atau menyewa prosesor

tertentu untuk digunakan selama beberapa bulan atau tahun, Anda menyewa kapasitas per jam. Tujuan kami adalah menyediakan jumlah kapasitas CPU yang konsisten dan spesifik, dalam batas perangkat keras sebenarnya yang mendasarinya.

ECU

Ukuran relatif kekuatan pemrosesan integer dari EC2 instance Amazon. Untuk memudahkan pengembang membandingkan kapasitas CPU antara kelas instans yang berbeda, kami telah menetapkan Unit EC2 Komputasi Amazon. Jumlah CPU yang dialokasikan untuk instance tertentu dinyatakan dalam satuan EC2 Compute Units ini. Satu ECU saat ini menyediakan kapasitas CPU yang setara dengan prosesor 1.0-1.2 GHz 2007 Opteron atau 2007 Xeon.

Memori (GiB)

RAM, dalam gibibyte, dialokasikan ke instans DB. Sering kali ada rasio yang konsisten antara memori dan vCPU. Sebagai contoh, ambil kelas instans db.r4, yang memiliki rasio memori terhadap vCPU serupa dengan kelas instans db.r5. Namun, untuk sebagian besar kasus penggunaan, kelas instans db.r5 memberikan performa yang lebih baik dan lebih konsisten dibandingkan kelas instans db.r4.

Maks. Bandwidth EBS (Mbps)

Bandwidth EBS maksimum dalam megabit per detik. Bagi dengan 8 untuk mendapatkan throughput yang diharapkan dalam megabyte per detik.

> **Note**
>
> Angka ini mengacu pada bandwidth I/O untuk penyimpanan lokal dalam instans DB. Ini tidak berlaku untuk komunikasi dengan volume klaster Aurora.

Bandwith jaringan

Kecepatan jaringan relatif terhadap kelas instans DB lainnya.

Untuk informasi tentang penggunaan CloudWatch metrik Amazon untuk memantau throughput instans Aurora DB Anda, lihat dan. Mengevaluasi penggunaan instans DB untuk Aurora My SQL dengan metrik Amazon CloudWatch Mengevaluasi penggunaan instans DB untuk Aurora CloudWatch PostgreSQL dengan metrik

## Spesifikasi perangkat keras untuk kelas instans yang dioptimalkan memori

Tabel berikut menunjukkan spesifikasi komputasi, memori, penyimpanan, dan bandwidth untuk kelas instance yang dioptimalkan memori.

db.x2g — kelas instance yang dioptimalkan untuk memori dengan prosesor Graviton2 AWS

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.x2g.16xlarge | 64 | — | 1024 | Khusus yang dioptimalkan EBS | 19.000 | 25 |
| db.x2g.12xlarge | 48 | — | 768 | Khusus yang dioptimalkan EBS | 14,250 | 20 |
| db.x2g.8xlarge | 32 | — | 512 | Khusus yang dioptimalkan EBS | 9.500 | 12 |
| db.x2g.4xlarge | 16 | — | 256 | Khusus yang dioptimalkan EBS | 4,750 | Hingga 10 |
| db.x2g.2xlarge | 8 | — | 128 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |
| db.x2g.xlarge | 4 | — | 64 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |
| db.x2g.large | 2 | — | 32 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |

db.r8g — kelas instans yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton4 AWS

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r8g.48xlarge | 192 | — | 1536 | Khusus yang dioptimalkan EBS | 40.000 | 50 |
| db.r8g.24xlarge | 96 | — | 768 | Khusus yang dioptimalkan EBS | 30.000 | 40 |
| db.r8g.16xlarge | 64 | — | 512 | Khusus yang dioptimalkan EBS | 20.000 | 30 |
| db.r8g.12xlarge | 48 | — | 384 | Khusus yang dioptimalkan EBS | 15.000 | 22.5 |
| db.r8g.8xlarge | 32 | — | 256 | Khusus yang dioptimalkan EBS | 10.000 | 15 |
| db.r8g.4xlarge | 16 | — | 128 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 15 |
| db.r8g.2xlarge | 8 | — | 64 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 15 |
| db.r8g.xlarge | 4 | — | 32 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r8g.large | 2 | — | 16 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |

db.r7i — kelas instans yang dioptimalkan untuk memori yang didukung oleh prosesor Intel Xeon Scalable generasi ke-4

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r7i.48xlarge | 192 | — | 1536 | Khusus yang dioptimalkan EBS | 40.000 | 50 |
| db.r7i.24xlarge | 96 | — | 768 | Khusus yang dioptimalkan EBS | 30.000 | 37,5 |
| db.r7i.16xlarge | 64 | — | 512 | Khusus yang dioptimalkan EBS | 20.000 | 25 |
| db.r7i.12xlarge | 48 | — | 384 | Khusus yang dioptimalkan EBS | 15.000 | 18.75 |
| db.r7i.8xlarge | 32 | — | 256 | Khusus yang dioptimalkan EBS | 10.000 | 12,5 |

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r7i.4xlarge | 16 | — | 128 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r7i.2xlarge | 8 | — | 64 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r7i.xlarge | 4 | — | 32 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r7i.large | 2 | — | 16 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |

db.r7g — kelas instance yang dioptimalkan untuk memori dengan prosesor Graviton3 AWS

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r7g.16xlarge | 64 | — | 512 | Khusus yang dioptimalkan EBS | 20.000 | 30 |
| db.r7g.12xlarge | 48 | — | 384 | Khusus yang dioptimalkan EBS | 15.000 | 22.5 |
| db.r7g.8xlarge | 32 | — | 256 | Khusus yang dioptimalkan EBS | 10.000 | 15 |

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r7g.4xlarge | 16 | — | 128 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 15 |
| db.r7g.2xlarge | 8 | — | 64 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 15 |
| db.r7g.xlarge | 4 | — | 32 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r7g.large | 2 | — | 16 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |

db.r6id — kelas instans yang dioptimalkan untuk memori dengan prosesor Intel Xeon Scalable generasi ke-3 dan penyimpanan SSD

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r6id.32xlarge | 128 | — | 1,024 | 4x1900 SSD NVMe | 40.000 | 50 |
| db.r6id.24xlarge | 96 | — | 768 | SSD 4x1425 NVMe | 30.000 | 37,5 |

db.r6gd — kelas instans yang dioptimalkan untuk memori dengan prosesor Graviton2 dan penyimpanan SSD AWS

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r6gd.16xlarge | 64 | — | 512 | 2 x 1900 SSD NVMe | 19.000 | 25 |
| db.r6gd.12xlarge | 48 | — | 384 | 2 x 1425 SSD NVMe | 13.500 | 20 |
| db.r6gd.8xlarge | 32 | — | 256 | 1 x 1900 SSD NVMe | 9.000 | 12 |
| db.r6gd.4xlarge | 16 | — | 128 | 1 x 950 SSD NVMe | 4,750 | Hingga 10 |
| db.r6gd.2xlarge | 8 | — | 64 | 1 x 474 SSD NVMe | Hingga 4.750 | Hingga 10 |
| db.r6gd.xlarge | 4 | — | 32 | 1 x 237 SSD NVMe | Hingga 4.750 | Hingga 10 |

db.r6g — kelas instance yang dioptimalkan untuk memori dengan prosesor Graviton2 AWS

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r6g.16xlarge | 64 | — | 512 | Khusus yang dioptimalkan EBS | 19.000 | 25 |
| db.r6g.12xlarge | 48 | — | 384 | Khusus yang dioptimalkan EBS | 13.500 | 20 |

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r6g.8xlarge | 32 | — | 256 | Khusus yang dioptimalkan EBS | 9.000 | 12 |
| db.r6g.4xlarge | 16 | — | 128 | Khusus yang dioptimalkan EBS | 4,750 | Hingga 10 |
| db.r6g.2xlarge | 8 | — | 64 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |
| db.r6g.xlarge | 4 | — | 32 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |
| db.r6g.large | 2 | — | 16 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |

db.r6i – kelas instans yang memorinya dioptimalkan dengan prosesor Intel Xeon Scalable generasi ke-3

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r6i.32xlarge | 128 | — | 1,024 | Khusus yang dioptimalkan EBS | 40.000 | 50 |

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r6i.24xlarge | 96 | — | 768 | Khusus yang dioptimalkan EBS | 30.000 | 37,5 |
| db.r6i.16xlarge | 64 | — | 512 | Khusus yang dioptimalkan EBS | 20.000 | 25 |
| db.r6i.12xlarge | 48 | — | 384 | Khusus yang dioptimalkan EBS | 15.000 | 18.75 |
| db.r6i.8xlarge | 32 | — | 256 | Khusus yang dioptimalkan EBS | 10.000 | 12,5 |
| db.r6i.4xlarge | 16 | — | 128 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r6i.2xlarge | 8 | — | 64 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r6i.xlarge | 4 | — | 32 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |
| db.r6i.large | 2 | — | 16 | Khusus yang dioptimalkan EBS | Hingga 10.000 | Hingga 12,5 |

db.r5 – kelas instans dengan memori yang dioptimalkan

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r5.24xlarge | 96 | 347 | 768 | Khusus yang dioptimalkan EBS | 19.000 | 25 |
| db.r5.16xlarge | 64 | 264 | 512 | Khusus yang dioptimalkan EBS | 13.600 | 20 |
| db.r5.12xlarge | 48 | 173 | 384 | Khusus yang dioptimalkan EBS | 9.500 | 12 |
| db.r5.8xlarge | 32 | 132 | 256 | Khusus yang dioptimalkan EBS | 6,800 | 10 |
| db.r5.4xlarge | 16 | 71 | 128 | Khusus yang dioptimalkan EBS | 4,750 | Hingga 10 |
| db.r5.2xlarge | 8 | 38 | 64 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |
| db.r5.xlarge | 4 | 19 | 32 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |
| db.r5.large | 2 | 10 | 16 | Khusus yang dioptimalkan EBS | Hingga 4.750 | Hingga 10 |

db.r4 – kelas instans yang memorinya dioptimalkan dengan prosesor Intel Xeon Scalable

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.r4.16xlarge | 64 | 195 | 488 | Khusus yang dioptimalkan EBS | 14.000 | 25 |
| db.r4.8xlarge | 32 | 99 | 244 | Khusus yang dioptimalkan EBS | 7.000 | 10 |
| db.r4.4xlarge | 16 | 53 | 122 | Khusus yang dioptimalkan EBS | 3.500 | Hingga 10 |
| db.r4.2xlarge | 8 | 27 | 61 | Khusus yang dioptimalkan EBS | 1.700 | Hingga 10 |
| db.r4.xlarge | 4 | 13,5 | 30,5 | Khusus yang dioptimalkan EBS | 850 | Hingga 10 |
| db.r4.large | 2 | 7 | 15.25 | Khusus yang dioptimalkan EBS | 425 | Hingga 10 |

## Spesifikasi perangkat keras untuk kelas instans performa yang dapat dibobol

Tabel berikut menunjukkan spesifikasi komputasi, memori, penyimpanan, dan bandwidth untuk kelas instance burstable-performance.

db.t4g — kelas instance berkinerja pecah yang ditenagai oleh prosesor Graviton2 AWS

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.t4g.large | 2 | — | 8 | Khusus yang dioptimalkan EBS | Hingga 2.780 | Hingga 5 |
| db.t4g.medium | 2 | — | 4 | Khusus yang dioptimalkan EBS | Hingga 2.085 | Hingga 5 |

db.t3 – kelas instans performa yang dapat melonjak

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.t3.large | 2 | Variab | 8 | Khusus yang dioptimalkan EBS | Hingga 2.048 | Hingga 5 |
| db.t3.medium | 2 | Variab | 4 | Khusus yang dioptimalkan EBS | Hingga 1.536 | Hingga 5 |
| db.t3.small | 2 | Variab | 2 | Khusus yang dioptimalkan EBS | Hingga 1.536 | Hingga 5 |

db.t2 – kelas instans performa yang dapat melonjak

| Kelas instans | vCPU | ECU | Memori (GiB) | Penyimpanan instans (GiB) | Maks. Bandwidth EBS (Mbps) | Bandwith jaringan (Gbps) |
|---|---|---|---|---|---|---|
| db.t2.medium | 2 | Variab | 4 | Khusus EBS | — | Sedang |
| db.t2.small | 1 | Variab | 2 | Khusus EBS | — | Rendah |

# Penyimpanan Amazon Aurora

Di bagian berikut ini, Anda dapat mempelajari tentang subsistem penyimpanan Aurora. Aurora menggunakan arsitektur penyimpanan terdistribusi dan bersama yang merupakan faktor penting dalam performa, skalabilitas, dan keandalan klaster Aurora.

Topik

- Gambaran umum penyimpanan Amazon Aurora
- Apa saja konten volume klaster
- Konfigurasi penyimpanan untuk klaster DB Amazon Aurora
- Cara penyimpanan Aurora berubah ukuran secara otomatis
- Cara penyimpanan data Aurora ditagih

## Gambaran umum penyimpanan Amazon Aurora

Data Aurora disimpan dalam volume cluster, yang merupakan volume virtual tunggal yang menggunakan solid state drive ()SSDs. Volume cluster terdiri dari salinan data di tiga Availability Zone dalam satu AWS Region. Karena data direplikasi secara otomatis di Zona ketersediaan, data Anda sangat durabel dengan kemungkinan kehilangan data yang kecil. Replikasi ini juga mempertahankan ketersediaan basis data Anda selama failover. Hal itu terjadi karena salinan data sudah ada di Zona Ketersediaan lain dan terus melayani permintaan data ke instans DB dalam klaster DB Anda. Jumlah replikasi bergantung pada jumlah instans DB dalam klaster Anda.

Aurora menggunakan penyimpanan lokal terpisah untuk file sementara yang tidak persisten. Ini termasuk file yang digunakan untuk tujuan seperti mengurutkan set data besar selama pemrosesan kueri, dan membuat indeks. Untuk informasi selengkapnya, lihat Batas penyimpanan sementara untuk Aurora MySQL dan Batas penyimpanan sementara untuk Aurora PostgreSQL.

## Apa saja konten volume klaster

Volume klaster Aurora berisi semua data pengguna, objek skema, dan metadata internal seperti tabel sistem dan log biner. Misalnya, Aurora menyimpan semua tabel, indeks, objek besar biner (BLOBs), prosedur tersimpan, dan sebagainya untuk cluster Aurora dalam volume cluster.

Arsitektur penyimpanan bersama Aurora membuat data Anda terpisah dari instans DB di dalam klaster. Misalnya, Anda dapat menambahkan instans DB dengan cepat karena Aurora tidak membuat salinan data tabel yang baru. Sebagai gantinya, instans DB terhubung ke volume bersama yang sudah berisi semua data Anda. Anda dapat menghapus instans DB dari suatu klaster tanpa menghapus data dasar dari klaster tersebut. Aurora hanya akan menghapus data jika Anda menghapus seluruh klaster tersebut.

## Konfigurasi penyimpanan untuk klaster DB Amazon Aurora

Amazon Aurora memiliki dua konfigurasi penyimpanan klaster DB:

- Aurora I/O-Optimized— Peningkatan kinerja harga dan prediktabilitas untuk I/O-intensive applications. You pay only for the usage and storage of your DB clusters, with no additional charges for read and write I/O operasi.

  Aurora I/O-Optimizedadalah pilihan terbaik ketika I/O pengeluaran Anda adalah 25% atau lebih dari total pengeluaran database Aurora Anda.

  Anda dapat memilih Aurora I/O-Optimized saat Anda membuat atau memodifikasi klaster DB dengan versi mesin DB yang mendukung konfigurasi klaster Aurora I/O-Optimized. Anda dapat beralih dari Aurora I/O-Optimized ke Aurora Standard kapan saja.
- Aurora Standard— Harga hemat biaya untuk banyak aplikasi dengan penggunaan moderat I/O . Selain penggunaan dan penyimpanan cluster DB Anda, Anda juga membayar tarif standar per 1 juta permintaan untuk I/O operasi.

  Aurora Standardadalah pilihan terbaik ketika I/O pengeluaran Anda kurang dari 25% dari total pengeluaran database Aurora Anda.

  Anda dapat beralih dari Aurora Standard ke Aurora I/O-Optimized sekali setiap 30 hari. Saat Anda beralih antara Aurora Standard dan opsi Aurora I/O-Optimized penyimpanan untuk instans non-NVMe-based DB, tidak ada waktu henti. Namun, untuk instans DB NVMe berbasis, beralih antara Aurora I/O-Optimized dan opsi Aurora Standard penyimpanan memerlukan restart mesin database, yang dapat menyebabkan periode waktu henti yang singkat.

Untuk informasi tentang Wilayah AWS dan dukungan versi, lihatDaerah yang Didukung dan engine Aurora DB untuk konfigurasi penyimpanan cluster.

Untuk informasi selengkapnya tentang harga konfigurasi penyimpanan Amazon Aurora, lihat Harga Amazon Aurora.

Untuk informasi tentang memilih konfigurasi penyimpanan saat membuat klaster DB, lihat Membuat klaster DB. Untuk informasi tentang memodifikasi konfigurasi penyimpanan untuk klaster DB, lihat Pengaturan untuk Amazon Aurora.

## Cara penyimpanan Aurora berubah ukuran secara otomatis

Volume klaster Aurora secara otomatis bertambah seiring peningkatan jumlah data dalam basis data Anda. Untuk informasi tentang ukuran volume cluster Aurora maksimum untuk setiap versi mesin, lihat. Batas ukuran Amazon Aurora Penskalaan penyimpanan otomatis ini dikombinasikan dengan subsistem penyimpanan beperforma tinggi dan sangat terdistribusi. Hal ini menjadikan Aurora sebagai pilihan yang baik untuk data perusahaan Anda yang penting ketika keandalan dan ketersediaan yang tinggi menjadi tujuan utama Anda.

Untuk menampilkan status volume, lihat Menampilkan status volume untuk klaster DB Aurora MySQL atau Menampilkan status volume untuk klaster DB Aurora PostgreSQL. Untuk cara menyeimbangkan biaya penyimpanan dengan prioritas lain, Penskalaan penyimpanan jelaskan cara memantau `AuroraVolumeBytesLeftTotal` metrik `VolumeBytesUsed` Amazon Aurora dan masuk. CloudWatch

Saat data Aurora dihapus, ruang yang dialokasikan untuk data tersebut dikosongkan. Contoh penghapusan data termasuk menghapus atau memotong tabel. Pengurangan otomatis penggunaan penyimpanan ini membantu Anda meminimalkan biaya penyimpanan.

> **Note**
>
> Perilaku pembatasan penyimpanan dan perubahan ukuran dinamis yang dibahas di sini berlaku untuk tabel persisten dan data lain yang disimpan dalam volume klaster.
> Untuk Aurora PostgreSQL, data tabel sementara disimpan dalam instans DB lokal.
> Untuk Aurora MySQL versi 2, data tabel sementara disimpan secara default dalam volume klaster untuk instans penulis dan dalam penyimpanan lokal untuk instans pembaca. Untuk informasi selengkapnya, lihat Mesin penyimpanan untuk tabel sementara di disk.

Untuk Aurora MySQL versi 3, data tabel sementara disimpan dalam instans DB lokal atau dalam volume klaster. Untuk informasi selengkapnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3.

Ukuran maksimum tabel sementara yang berada di penyimpanan lokal dibatasi oleh ukuran penyimpanan lokal maksimum instans DB. Ukuran penyimpanan lokal bergantung pada kelas instans yang Anda gunakan. Untuk informasi selengkapnya, lihat Batas penyimpanan sementara untuk Aurora MySQL dan Batas penyimpanan sementara untuk Aurora PostgreSQL.

Beberapa fitur penyimpanan, seperti ukuran maksimum volume klaster dan pengubahan ukuran otomatis saat data dihapus, bergantung pada versi Aurora klaster Anda. Untuk informasi selengkapnya, lihat Penskalaan penyimpanan. Anda juga dapat mempelajari cara menghindari masalah penyimpanan dan cara memantau penyimpanan yang dialokasikan dan mengosongkan ruang dalam klaster Anda.

## Cara penyimpanan data Aurora ditagih

Meskipun volume cluster Aurora dapat tumbuh hingga 256 tebibytes (TiB) untuk versi mesin tertentu, Anda hanya dikenakan biaya untuk ruang yang Anda gunakan dalam volume cluster Aurora. Pada versi Aurora sebelumnya, volume klaster dapat menggunakan kembali ruang yang dikosongkan saat Anda menghapus data, tetapi ruang penyimpanan yang dialokasikan tidak akan pernah berkurang. Saat data Aurora dihapus, seperti menghapus tabel atau basis data, keseluruhan ruang yang dialokasikan berkurang sebesar jumlah yang sebanding. Dengan demikian, Anda dapat mengurangi biaya penyimpanan dengan menghapus tabel, indeks, basis data, dan sebagainya yang tidak lagi Anda butuhkan.

**Tip**

Untuk versi terdahulu tanpa fitur pengubahan ukuran dinamis, pengaturan ulang penggunaan penyimpanan untuk suatu klaster memerlukan dump logis dan pemulihan ke klaster baru. Operasi tersebut dapat memakan waktu lama untuk volume data yang besar. Jika Anda mengalami situasi ini, pertimbangkan untuk meningkatkan klaster Anda ke versi yang mendukung pengubahan ukuran volume dinamis.

Untuk informasi tentang versi Aurora yang mendukung pengubahan ukuran dinamis, dan cara meminimalkan biaya penyimpanan dengan memantau penggunaan penyimpanan untuk klaster

Anda, lihat Penskalaan penyimpanan. Untuk informasi tentang penagihan penyimpanan cadangan Aurora, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. Untuk informasi harga tentang penyimpanan data Aurora, lihat Harga Amazon RDS for Aurora.

# Keandalan Amazon Aurora

Aurora dirancang untuk menjadi andal, durabel, dan toleran terhadap kesalahan. Anda dapat menentukan arsitektur klaster DB Aurora Anda untuk meningkatkan ketersediaan dengan melakukan hal-hal seperti menambahkan Replika Aurora dan menempatkannya di Zona Ketersediaan yang berbeda-beda, dan Aurora juga menyertakan beberapa fitur otomatis yang menjadikannya solusi basis data yang andal.

Topik

- Perbaikan penyimpanan otomatis
- Cache halaman yang dapat bertahan
- Pemulihan dari pengaktifan ulang yang tidak direncanakan

## Perbaikan penyimpanan otomatis

Karena Aurora menyimpan beberapa salinan data Anda di tiga Zona Ketersediaan, kemungkinan kehilangan data akibat kegagalan disk dapat diminimalkan secara signifikan. Aurora secara otomatis mendeteksi kegagalan dalam volume disk yang membentuk volume klaster. Saat sebuah segmen volume disk gagal, Aurora segera memperbaiki segmen tersebut. Saat Aurora memperbaiki segmen disk, Aurora menggunakan data dalam volume lain yang membentuk volume klaster untuk memastikan bahwa data dalam segmen yang diperbaiki sudah diperbarui. Akibatnya, Aurora menghindari kehilangan data dan mengurangi kebutuhan untuk melakukan point-in-time pemulihan untuk memulihkan dari kegagalan disk.

## Cache halaman yang dapat bertahan

Di Aurora, cache halaman instans DB dikelola dalam proses terpisah dari basis data, yang memungkinkan cache halaman bertahan secara terpisah dari basis data. (Cache halaman juga disebut kumpulan buffer InnoDB di Aurora My SQL dan cache buffer di Aurora Postgre.) SQL

Jika terjadi kegagalan basis data yang tidak terduga, cache halaman tetap berada di memori, sehingga halaman data saat ini tetap dalam kondisi “warm” di cache halaman saat basis data

diaktifkan kembali. Hal ini memberikan peningkatan performa dengan meniadakan kebutuhan kueri awal untuk mengeksekusi operasi I/O untuk melakukan “warming” cache halaman.

Untuk Aurora MySQL, perilaku cache halaman saat me-reboot dan gagal adalah sebagai berikut:

- Anda dapat me-reboot instance penulis tanpa me-reboot instance pembaca.
  - Jika instans pembaca tidak melakukan boot ulang saat instans penulis melakukan boot ulang, instans tersebut tidak kehilangan cache halamannya.
  - Jika instans pembaca melakukan boot ulang saat instans penulis melakukan boot ulang, instans tersebut kehilangan cache halamannya.
- Ketika instans pembaca melakukan boot ulang, halaman cache pada instans penulis dan pembaca sama-sama bertahan.
- Ketika klaster DB gagal, efeknya mirip dengan ketika instans penulis melakukan boot ulang. Pada instans penulis baru (sebelumnya instans pembaca) cache halaman bertahan, tetapi pada instans pembaca (sebelumnya instans penulis), cache halaman tidak bertahan.

Untuk Aurora PostgreSQL, Anda dapat menggunakan manajemen cache cluster untuk mempertahankan cache halaman dari instance pembaca yang ditunjuk yang menjadi instance penulis setelah failover. Untuk informasi selengkapnya, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL.

## Pemulihan dari pengaktifan ulang yang tidak direncanakan

Aurora dirancang untuk pulih dari pengaktifan ulang yang tidak direncanakan hampir seketika dan terus melayani data aplikasi Anda tanpa log biner. Aurora pulih secara asinkron pada thread paralel, sehingga basis data Anda terbuka dan tersedia segera setelah pengaktifan ulang yang tidak direncanakan.

Untuk informasi selengkapnya, silakan lihat Toleransi kesalahan untuk klaster DB Aurora dan Optimisasi untuk mengurangi waktu pengaktifan ulang basis data..

Berikut ini adalah pertimbangan untuk pencatatan biner dan pemulihan restart yang tidak direncanakan di Aurora My: SQL

- Mengaktifkan log biner di Aurora secara langsung memengaruhi waktu pemulihan setelah pengaktifan ulang yang tidak direncanakan, karena hal itu memaksa instans DB untuk melakukan pemulihan log biner.

- Jenis pencatatan log biner yang digunakan memengaruhi ukuran dan efisiensi pencatatan log. Untuk jumlah aktivitas basis data yang sama, beberapa format mencatat log untuk lebih banyak informasi dibandingkan yang lain dalam log biner. Pengaturan berikut untuk parameter `binlog_format` menghasilkan jumlah data log yang berbeda:
  - `ROW` – Data log paling banyak
  - `STATEMENT` – Data log paling sedikit
  - `MIXED` – Data log dalam jumlah sedang yang biasanya memberikan kombinasi integritas data dan performa yang terbaik

  Jumlah data log biner memengaruhi waktu pemulihan. Jika ada lebih banyak data yang dicatat lognya dalam log biner, instans DB harus memproses lebih banyak data selama pemulihan, yang meningkatkan waktu pemulihan.
- Untuk mengurangi overhead komputasi dan meningkatkan waktu pemulihan dengan pencatatan log biner, Anda dapat menggunakan binlog yang disempurnakan. Binlog yang disempurnakan mempersingkat waktu pemulihan basis data hingga 99%. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang disempurnakan untuk Aurora MySQL.
- Aurora tidak memerlukan log biner untuk mereplikasi data dalam cluster DB atau untuk melakukan point-in-time restore (). PITR
- Jika Anda tidak membutuhkan log biner untuk replikasi eksternal (atau log stream biner eksternal), kami sarankan Anda mengatur parameter `binlog_format` ke `OFF` untuk menonaktifkan log biner. Hal ini mengurangi waktu pemulihan.

Untuk informasi selengkapnya tentang pencatatan log biner dan replikasi Aurora, lihat Replikasi dengan Amazon Aurora. Untuk informasi selengkapnya tentang implikasi berbagai jenis SQL replikasi Saya, lihat Keuntungan dan kerugian replikasi berbasis pernyataan dan berbasis baris dalam dokumentasi Saya. SQL

# Keamanan Amazon Aurora

Keamanan untuk Amazon Aurora dikelola di tiga tingkat:

- Untuk mengontrol siapa yang dapat melakukan tindakan RDS pengelolaan Amazon pada cluster DB Aurora dan instans DB, Anda menggunakan (). AWS Identity and Access Management IAM Saat Anda terhubung AWS menggunakan IAM kredensyal, AWS akun Anda harus memiliki IAM kebijakan yang memberikan izin yang diperlukan untuk melakukan operasi pengelolaan AmazonRDS. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.

Jika Anda menggunakan IAM untuk mengakses RDS konsol Amazon, Anda harus terlebih dahulu masuk ke AWS Management Console dengan kredensyal pengguna Anda, dan kemudian pergi ke RDS konsol Amazon di https://console.aws.amazon.com /rds.

- Cluster Aurora DB harus dibuat di cloud pribadi virtual (VPC) berdasarkan layanan Amazon. VPC Untuk mengontrol perangkat dan EC2 instans Amazon mana yang dapat membuka koneksi ke titik akhir dan port instans DB untuk cluster Aurora DB di aVPC, Anda menggunakan grup keamanan. VPC Anda dapat membuat koneksi titik akhir dan port ini menggunakan Transport Layer Security (TLS) /Secure Sockets Layer (SSL). Selain itu, aturan firewall di perusahaan Anda dapat mengendalikan perangkat yang berjalan di perusahaan Anda untuk dapat membuka koneksi ke instans DB. Untuk informasi lebih lanjut tentang VPCs, lihat Amazon VPC dan Aurora.
- Untuk mengautentikasi login dan izin untuk klaster DB Amazon Aurora, Anda dapat menggunakan salah satu pendekatan berikut, atau kombinasi dari beberapa pendekatan.
  - Anda dapat mengambil pendekatan yang sama dengan instance DB yang berdiri sendiri dari My SQL atau Postgre. SQL

    Teknik untuk mengautentikasi login dan izin untuk instance DB yang berdiri sendiri dari My SQL atau PostgreSQL, seperti menggunakan SQL perintah atau memodifikasi tabel skema database, juga bekerja dengan Aurora. Untuk informasi selengkapnya, lihat Keamanan dengan Amazon Aurora MySQL atau Keamanan dengan Amazon Aurora PostgreSQL.
  - Anda dapat menggunakan otentikasi IAM database.

    Dengan otentikasi IAM database, Anda mengautentikasi ke cluster Aurora DB Anda dengan menggunakan pengguna atau IAM peran dan token otentikasi. Token autentikasi adalah nilai unik yang dihasilkan dengan menggunakan proses penandatanganan Signature Versi 4. Dengan menggunakan otentikasi IAM basis data, Anda dapat menggunakan kredensyal yang sama untuk mengontrol akses ke AWS sumber daya dan basis data Anda. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.
  - Anda dapat menggunakan otentikasi Kerberos untuk Aurora SQL Postgre dan Aurora My. SQL

    Anda dapat menggunakan Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster Aurora Postgre dan Aurora My SQL Anda. SQLDB Dalam hal ini, cluster DB Anda bekerja dengan AWS Directory Service for Microsoft Active Directory untuk mengaktifkan otentikasi Kerberos. AWS Directory Service for Microsoft Active Directory disebut juga AWS Managed Microsoft AD. Dengan menyimpan semua kredensial Anda di direktori yang sama, Anda dapat menghemat waktu dan tenaga. Anda memiliki tempat terpusat untuk menyimpan dan mengelola kredensial untuk beberapa klaster DB. Penggunaan direktori juga dapat meningkatkan profil

keamanan Anda secara keseluruhan. Untuk informasi selengkapnya, silakan lihat Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL dan Menggunakan otentikasi Kerberos untuk Aurora My SQL.

Untuk informasi tentang konfigurasi keamanan, lihat Keamanan dalam Amazon Aurora.

## Menggunakan SSL dengan cluster Aurora DB

Cluster Amazon Aurora DB mendukung koneksi Secure Sockets Layer (SSL) dari aplikasi yang menggunakan proses dan kunci publik yang sama dengan instans Amazon RDS DB. Untuk informasi selengkapnya, lihat Keamanan dengan Amazon Aurora MySQL, Keamanan dengan Amazon Aurora PostgreSQL, atau Menggunakan TLS/SSL dengan Aurora Serverless v1.

# Ketersediaan yang tinggi untuk Amazon Aurora

Arsitektur Amazon Aurora melibatkan pemisahan penyimpanan dan komputasi. Aurora memiliki beberapa fitur ketersediaan tinggi yang berlaku untuk data dalam klaster DB Anda. Data tetap aman meskipun beberapa atau semua instans DB di klaster tidak tersedia. Fitur ketersediaan tinggi lainnya berlaku untuk instans DB. Fitur-fitur ini membantu memastikan bahwa satu atau lebih instans DB siap menangani permintaan basis data dari aplikasi Anda.

Topik

- Ketersediaan data Aurora yang tinggi
- Ketersediaan yang tinggi untuk instans DB Aurora
- Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora
- Toleransi kesalahan untuk klaster DB Aurora
- Ketersediaan yang tinggi dengan Proksi Amazon RDS

## Ketersediaan data Aurora yang tinggi

Aurora menyimpan salinan data dalam klaster basis data di beberapa Zona Ketersediaan dalam satu Wilayah AWS. Aurora menyimpan salinan ini terlepas jika instans dalam klaster DB mencakup beberapa Zona Ketersediaan. Untuk informasi selengkapnya tentang Aurora, lihat Mengelola klaster DB Amazon Aurora.

Ketika data ditulis ke instans DB primer, Aurora secara sinkron mereplikasi data di seluruh Zona Ketersediaan ke enam simpul penyimpanan yang terkait dengan volume klaster Anda. Melakukan hal

tersebut akan memberikan redundansi data, menghilangkan I/O freeze, dan meminimalkan lonjakan latensi selama pencadangan sistem. Menjalankan instans DB dengan ketersediaan tinggi dapat meningkatkan ketersediaan selama pemeliharaan sistem terencana, dan membantu melindungi basis data Anda terhadap kegagalan dan gangguan Zona Ketersediaan. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.

## Ketersediaan yang tinggi untuk instans DB Aurora

Setelah Anda membuat instans primer (penulis), Anda dapat membuat hingga 15 Replika Aurora hanya baca. Replika Aurora juga dikenal sebagai instans pembaca. Aurora Replicas menggunakan replikasi asinkron untuk mendukung ketersediaan tinggi tanpa mempengaruhi kinerja instans primer.

Selama day-to-day operasi, Anda dapat menurunkan beberapa pekerjaan untuk aplikasi intensif baca dengan menggunakan instance pembaca untuk memproses kueri. `SELECT` Ketika masalah memengaruhi instans utama, salah satu dari instans pembaca ini mengambil alih sebagai instans utama. Mekanisme ini dikenal sebagai failover. Banyak fitur Aurora berlaku pada mekanisme failover. Misalnya, Aurora mendeteksi masalah basis data dan mengaktifkan mekanisme failover secara otomatis jika diperlukan. Aurora juga memiliki fitur yang mengurangi waktu failover untuk menyelesaikannya. Melakukan hal ini meminimalkan waktu tidak tersedianya basis data untuk menulis selama failover.

Aurora dirancang untuk pulih secepat mungkin, dan jalur tercepat menuju pemulihan sering kali dimulai ulang atau gagal ke instans DB yang sama. Restart lebih cepat dan melibatkan lebih sedikit overhead daripada failover.

Untuk menggunakan string koneksi yang tetap sama bahkan ketika failover menaikkan instans primer baru, Anda terhubung ke titik akhir klaster. Titik akhir klaster selalu mewakili instans utama saat ini di klaster. Untuk informasi selengkapnya tentang titik akhir klaster, lihat Koneksi titik akhir Amazon Aurora.

> **Tip**
>
> Dalam masing-masing Wilayah AWS, Availability Zones (AZs) mewakili lokasi yang berbeda satu sama lain untuk memberikan isolasi jika terjadi pemadaman. Kami menyarankan Anda mendistribusikan instans utama dan instans pembaca di cluster DB Anda melalui beberapa AZs untuk meningkatkan ketersediaan cluster DB Anda. Dengan begitu, masalah yang memengaruhi seluruh AZ tidak menyebabkan pemadaman untuk klaster Anda.
> Anda dapat mengatur cluster DB multi-AZ dengan membuat pilihan sederhana saat Anda membuat cluster. Anda dapat menggunakan AWS Management Console, AWS CLI, atau

Amazon RDS API. Anda juga dapat mengonversi cluster Aurora DB yang ada menjadi cluster DB multi-AZ dengan menambahkan instans DB pembaca baru dan menentukan AZ yang berbeda.

## Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora

Untuk ketersediaan tinggi di beberapa Wilayah AWS, Anda dapat mengatur database global Aurora. Setiap database global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh dunia. Wilayah AWS Aurora secara asinkron mereplikasi semua data dan pembaruan dari primer Wilayah AWS ke masing-masing Wilayah sekunder. Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

## Toleransi kesalahan untuk klaster DB Aurora

Klaster DB Aurora toleran terhadap kesalahan secara default. Volume cluster mencakup beberapa Availability Zones (AZs) dalam satu Wilayah AWS, dan setiap Availability Zone berisi salinan data volume cluster. Fungsionalitas ini berarti bahwa klaster DB Anda dapat menoleransi kesalahan dari Zona Ketersediaan tanpa kehilangan data dan hanya berupa gangguan layanan yang singkat.

Jika instans DB primer saat ini pada suatu klaster DB gagal, Aurora secara otomatis melakukan failover ke instans DB primer yang baru.

- Dengan menaikkan Replika Aurora yang sudah ada ke instans utama yang baru
- Dengan membuat instans primer baru

Jika klaster DB memiliki satu atau lebih Replika Aurora, maka Replika Aurora dipromosikan ke instans primer selama peristiwa failover. Peristiwa kegagalan mengakibatkan interupsi singkat, selama operasi baca dan tulis gagal dengan pengecualian. Namun, layanan biasanya dipulihkan dalam waktu kurang dari 60 detik, dan sering kali kurang dari 30 detik. Untuk meningkatkan ketersediaan klaster DB Anda, kami sarankan Anda membuat setidaknya satu Replika Aurora atau lebih di dua Zona Ketersediaan yang berbeda.

> **Tip**
>
> Di Aurora MySQL, Anda dapat meningkatkan ketersediaan selama failover dengan memiliki lebih dari satu instans DB pembaca dalam sebuah cluster. Di Aurora MySQL, Aurora hanya me-restart instance DB penulis dan instance pembaca yang gagal. Instans pembaca lain dalam klaster ini tetap tersedia selama failover untuk terus memproses kueri melalui koneksi ke titik akhir pembaca.
> Anda juga dapat meningkatkan ketersediaan selama failover dengan menggunakan Proksi RDS dengan klaster DB Aurora Anda. Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi dengan Proksi Amazon RDS.

Anda dapat menyesuaikan urutan di mana Replika Aurora Anda dinaikkan ke instans utama setelah kegagalan dengan menetapkan masing-masing replika sebagai prioritas. Prioritas berkisar dari 0 untuk prioritas tertinggi hingga 15 untuk prioritas terendah. Jika instans primer gagal, Amazon RDS mempromosikan Replika Aurora dengan prioritas yang paling tinggi untuk instans primer baru. Anda dapat mengubah prioritas dari Replika Aurora kapan saja. Memodifikasi prioritas tidak memicu failover.

Lebih dari satu Replika Aurora dapat memiliki prioritas yang sama, yang menghasilkan tingkat promosi. Jika dua atau lebih Replika Aurora memiliki prioritas yang sama, maka Amazon RDS menaikkan replika dengan ukuran paling besar. Jika dua atau lebih Replika Aurora memiliki prioritas yang sama, maka Amazon RDS menaikkan replika bebas dengan tingkat promosi yang sama.

> **Note**
>
> Beberapa faktor terlibat dalam mengidentifikasi target failover. Setelah lima upaya failover yang gagal, tingkatan promosi tidak lagi dipertimbangkan.

Jika klaster DB tidak berisi Replika Aurora, maka instans primer dibuat ulang di AZ yang sama selama peristiwa kegagalan. Peristiwa kegagalan mengakibatkan interupsi yang operasi baca dan tulisnya gagal dengan pengecualian. Layanan dipulihkan ketika instans primer baru dibuat, yang biasanya memakan waktu kurang dari 10 menit. Mempromosikan Replika Aurora ke instans primer jauh lebih cepat daripada membuat instans primer baru.

Misalkan instans primer di klaster Anda tidak tersedia karena pemadaman yang memengaruhi keseluruhan AZ. Dalam hal ini, cara untuk menjadikan instans primer baru online tergantung pada apakah klaster Anda menggunakan konfigurasi Multi-AZ:

- Jika Anda disediakan atau Aurora Serverless v2 cluster berisi instance pembaca apa pun di lain AZs, Aurora menggunakan mekanisme failover untuk mempromosikan salah satu instance pembaca tersebut menjadi instance utama yang baru.
- Jika Anda disediakan atau Aurora Serverless v2 cluster hanya berisi satu instans DB, atau jika instance utama dan semua instance pembaca berada di AZ yang sama, pastikan untuk membuat satu atau lebih instance DB baru secara manual di AZ lain.
- Jika klaster Anda menggunakan Aurora Serverless v1, Aurora secara otomatis membuat instance DB baru di AZ lain. Namun, proses ini melibatkan penggantian host, sehingga membutuhkan waktu lebih lama dari failover.

> **Note**
>
> Amazon Aurora juga mendukung replikasi dengan basis data MySQL eksternal, atau instans DB RDS MySQL. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

### Ketersediaan yang tinggi dengan Proksi Amazon RDS

Dengan Proksi RDS, Anda dapat membangun aplikasi yang dapat secara transparan menoleransi kegagalan basis data tanpa perlu menulis kode penanganan kegagalan yang kompleks. Proksi RDS secara otomatis merutekan lalu lintas ke instans basis data baru sekaligus mempertahankan koneksi aplikasi. Proksi RDS juga melewati cache Sistem Nama Domain (DNS) untuk mengurangi waktu failover hingga 66% untuk basis data Multi-AZ Aurora. Untuk informasi selengkapnya, lihat Proksi Amazon RDS untuk Aurora.

## Replikasi dengan Amazon Aurora

Ada beberapa opsi replikasi dengan Aurora. Setiap klaster DB Aurora memiliki replikasi default di antara beberapa instans DB dalam klaster yang sama. Anda juga dapat mengatur replikasi dengan klaster Aurora sebagai sumber atau target. Ketika mereplikasi data ke dalam atau ke luar dari klaster Aurora, Anda dapat memilih antara fitur default seperti basis data global Aurora atau mekanisme replikasi biasa untuk mesin MySQL atau PostgreSQL DB. Anda dapat memilih opsi yang sesuai

berdasarkan fitur mana yang memberikan kombinasi yang tepat antara ketersediaan, kepraktisan, dan performa yang tinggi untuk kebutuhan Anda. Bagian berikut ini menjelaskan bagaimana dan kapan harus memilih masing-masing teknik.

Topik


- Replika Aurora
- Replikasi dengan Aurora MySQL
- Replikasi dengan Aurora PostgreSQL


## Replika Aurora

Ketika Anda membuat instans DB kedua, ketiga, dan seterusnya dalam klaster DB yang disediakan Aurora, Aurora secara otomatis mengatur replikasi dari instans DB penulis ke semua instans DB lainnya. Instans DB lainnya ini bersifat hanya baca dan dikenal sebagai Replika Aurora. Kami juga menyebutnya sebagai instans pembaca ketika membahas cara-cara bagi Anda untuk menggabungkan instans DB penulis dan pembaca dalam sebuah klaster.

Replika Aurora memiliki dua tujuan utama. Anda dapat menerbitkan kueri ke replika ini untuk menskalakan operasi baca untuk aplikasi Anda. Anda biasanya melakukannya dengan menghubungkan ke titik akhir pembaca pada klaster. Dengan demikian, Aurora dapat menyebarkan beban untuk koneksi hanya baca ke sebanyak Replika Aurora yang Anda miliki di klaster. Replika Aurora juga membantu meningkatkan ketersediaan. Jika instans penulis dalam klaster menjadi tidak tersedia, Aurora secara otomatis mempromosikan salah satu instans pembaca untuk menggantikannya sebagai penulis baru.

Sebuah klaster DB Aurora dapat berisi hingga 15 Replika Aurora. Replika Aurora dapat didistribusikan di seluruh Availability Zone yang mencakup cluster DB dalam suatu Wilayah. AWS

Data dalam klaster DB Anda memiliki fitur ketersediaan tinggi dan keandalannya sendiri, yang terpisah dari instans DB di klaster. Jika Anda belum memahami fitur penyimpanan Aurora, lihat Gambaran umum penyimpanan Amazon Aurora. Volume klaster DB secara fisik terdiri atas beberapa salinan data untuk klaster DB. Instans primer dan Replika Aurora di klaster DB melihat data dalam volume klaster sebagai volume logis tunggal.

Akibatnya, semua Replika Aurora menghasilkan data yang sama untuk hasil kueri dengan sedikit lag replika. Lag ini biasanya kurang dari 100 milidetik setelah instans primer menulis pembaruan. Lag replika bervariasi bergantung pada laju perubahan basis data. Artinya, selama periode saat sejumlah besar operasi tulis terjadi untuk basis data, Anda mungkin melihat peningkatan lag replika.

> **Note**
>
> Aurora Replica restart secara otomatis, ketika kehilangan komunikasi dengan instans DB penulis selama lebih dari 60 detik dalam versi PostgreSQL Aurora berikut:
>
> - 14.6 dan versi yang lebih lama
> - 13.9 dan versi yang lebih lama
> - 12.13 dan versi yang lebih lama
> - Semua Aurora PostgreSQL 11 versi
>
> Dengan fitur ketersediaan baca, Replika Aurora tidak dimulai ulang secara otomatis. Untuk informasi selengkapnya tentang fitur ketersediaan baca dan versi yang berlaku, lihatMeningkatkan ketersediaan baca Aurora Replica.

Replika Aurora berfungsi dengan baik untuk penskalaan baca karena ditujukan sepenuhnya untuk operasi baca pada volume klaster Anda. Operasi tulis dikelola oleh instans primer. Karena volume klaster dibagikan di antara semua instans dalam klaster DB Aurora Anda, pekerjaan tambahan untuk mereplikasi salinan data untuk setiap Replika Aurora menjadi minimal.

Untuk meningkatkan ketersediaan, Anda dapat menggunakan Replika Aurora sebagai target failover. Artinya, jika instans primer gagal, Replika Aurora dipromosikan menjadi instans primer. Akan ada interupsi singkat saat permintaan baca dan tulis yang dibuat ke instans primer mengalami kegagalan dengan pengecualian.

Mempromosikan Replika Aurora melalui failover jauh lebih cepat daripada membuat ulang instans primer. Jika klaster DB Aurora Anda tidak menyertakan Replika Aurora, klaster DB Anda tidak akan tersedia selama instans DB Anda melakukan pemulihan dari kegagalan.

Ketika failover terjadi, beberapa Replika Aurora mungkin di-boot ulang, bergantung pada versi mesin DB. Misalnya, di Aurora MySQL, Aurora hanya me-restart instance DB penulis dan target failover selama failover. Untuk informasi selengkapnya tentang perilaku boot ulang versi mesin DB Aurora yang berbeda-beda, lihat Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora. Untuk informasi tentang apa yang terjadi pada cache halaman saat boot ulang atau failover, lihat Cache halaman yang dapat bertahan.

Untuk skenario ketersediaan tinggi, kami sarankan Anda membuat satu atau beberapa Replika Aurora. Replika ini harus berasal dari kelas instans DB yang sama dengan instans primer dan berada

di Zona Ketersediaan yang berbeda untuk klaster DB Aurora Anda. Untuk informasi selengkapnya tentang Replika Aurora sebagai target failover, lihat Toleransi kesalahan untuk klaster DB Aurora.

Anda tidak dapat membuat Replika Aurora terenkripsi untuk klaster DB Aurora yang tidak terenkripsi. Anda tidak dapat membuat Replika Aurora yang tidak terenkripsi untuk klaster DB Aurora terenkripsi.

> **Tip**
>
> Anda dapat menggunakan Replika Aurora dalam klaster Aurora sebagai satu-satunya bentuk replikasi untuk mempertahankan ketersediaan tinggi data Anda. Anda juga dapat menggabungkan replikasi Aurora default dengan jenis replikasi lainnya. Tindakan tersebut dapat membantu menambah tingkat ketersediaan tinggi dan distribusi geografis data Anda.

Untuk detail tentang cara membuat Replika Aurora, lihat Menambahkan Replika Aurora ke klaster DB.

## Replikasi dengan Aurora MySQL

Selain Replika Aurora, Anda memiliki opsi berikut untuk replikasi dengan Aurora MySQL:

- Aurora MySQL DB cluster di Wilayah yang berbeda. AWS
  - Anda dapat mereplikasi data di beberapa Wilayah menggunakan basis data global Aurora. Untuk detailnya, lihat Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora
  - Anda dapat membuat replika baca Aurora dari cluster DB MySQL Aurora di AWS Wilayah yang berbeda, dengan menggunakan replikasi log biner MySQL (binlog). Setiap klaster dapat memiliki hingga lima replika baca yang dibuat dengan cara ini, masing-masing di Wilayah berbeda.
- Dua klaster DB Aurora MySQL di Wilayah yang sama, dengan menggunakan replikasi log biner (binlog) MySQL.
- Instans DB RDS for MySQL sebagai data sumber dan klaster DB Aurora MySQL, dengan membuat replika baca Aurora dari instans DB RDS for MySQL. Biasanya, pendekatan ini digunakan untuk migrasi ke Aurora MySQL, bukan replikasi berkelanjutan.

Untuk informasi selengkapnya tentang replikasi dengan Aurora MySQL, lihat Replikasi dengan Amazon Aurora MySQL.

## Replikasi dengan Aurora PostgreSQL

Selain Replika Aurora, Anda memiliki opsi berikut untuk replikasi dengan Aurora PostgreSQL:

- Cluster DB primer Aurora di satu Wilayah dan hingga 10 cluster DB sekunder read-only di Wilayah yang berbeda dengan menggunakan database global Aurora. Aurora PostgreSQL tidak mendukung Replika Aurora lintas Wilayah. Namun, Anda dapat menggunakan database global Aurora untuk menskalakan kemampuan baca klaster PostgreSQL DB Aurora Anda ke lebih dari satu Wilayah dan untuk memenuhi tujuan ketersediaan. AWS Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.
- Dua klaster DB Aurora PostgreSQL di Wilayah yang sama, menggunakan fitur replika logis PostgreSQL.
- Instans DB RDS for PostgreSQL sebagai sumber data dan klaster DB Aurora PostgreSQL, dengan membuat replika baca Aurora dari instans DB PostgreSQL. Biasanya, pendekatan ini digunakan untuk migrasi ke Aurora PostgreSQL, bukan replikasi berkelanjutan.

Untuk informasi selengkapnya tentang replikasi dengan Aurora PostgreSQL, lihat Replikasi dengan Amazon Aurora PostgreSQL.

## DB untuk Aurora

Instans RDS yang disediakan Amazon di klaster Amazon Aurora ditagih berdasarkan komponen berikut:

- Jam instans DB (per jam) – Berdasarkan kelas instans DB dari instans DB (misalnya, db.t2.small atau db.m4.large). Harga terdaftar berdasarkan per jam, tetapi tagihan dihitung turun ke yang kedua dan menunjukkan waktu dalam bentuk desimal. RDSpenggunaan ditagih dalam kenaikan 1 detik, dengan minimal 10 menit. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB.
- Penyimpanan (per GiB per bulan) – Kapasitas penyimpanan yang telah Anda sediakan untuk instans DB Anda. Jika Anda menskalakan kapasitas penyimpanan yang telah Anda sediakan dalam bulan tertentu, tagihan Anda akan diprorata. Untuk informasi selengkapnya, lihat Penyimpanan Amazon Aurora.
- Input/output (I/O) requests (per 1 million requests) – Total number of storage I/Opermintaan yang telah Anda buat dalam siklus penagihan, untuk saja.

Untuk informasi selengkapnya tentang penagihan I/O Amazon Aurora, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

- Penyimpanan cadangan (per GiB per bulan) – Penyimpanan cadangan adalah penyimpanan yang terkait dengan cadangan basis data otomatis dan setiap snapshot basis data aktif yang telah Anda ambil. Meningkatkan periode retensi cadangan atau mengambil snapshot basis data tambahan akan meningkatkan penyimpanan cadangan yang digunakan oleh basis data Anda. Tagihan per detik tidak berlaku untuk penyimpanan cadangan (diukur dalam GB-bulan).

  Untuk informasi selengkapnya, lihat Mencadangkan dan memulihkan klaster DB Amazon Aurora.
- Transfer data (per GB) — Transfer data masuk dan keluar dari instans DB Anda dari atau ke internet dan AWS Wilayah lain. Untuk contoh yang berguna, lihat posting AWS blog Menjelajahi Biaya Transfer Data untuk Database AWS Terkelola.

Amazon RDS menyediakan opsi pembelian berikut untuk memungkinkan Anda mengoptimalkan biaya berdasarkan kebutuhan Anda:

- Instans Sesuai Permintaan – Bayar per jam untuk periode instans DB yang Anda gunakan. Harga terdaftar berdasarkan per jam, tetapi tagihan dihitung turun ke yang kedua dan menunjukkan waktu dalam bentuk desimal. RDSpenggunaan sekarang ditagih dalam kenaikan 1 detik, dengan minimal 10 menit.
- Instans terpesan – Memesan instans DB untuk jangka waktu satu tahun atau tiga tahun dan mendapatkan diskon yang signifikan dibandingkan dengan harga instans DB sesuai permintaan. Dengan penggunaan Instans Terpesan, Anda dapat meluncurkan, menghapus, memulai, atau menghentikan beberapa instans dalam satu jam dan mendapatkan keuntungan Instans Terpesan untuk semua instans.
- Aurora Serverless v2 – Aurora Serverless v2 menyediakan kapasitas sesuai permintaan di mana unit penagihan adalah unit kapasitas Aurora (ACU) jam, bukan jam instans DB. Aurora Serverless v2 kapasitas meningkat dan menurun, dalam rentang yang Anda tentukan, tergantung pada beban pada database Anda. Anda dapat mengkonfigurasi cluster di mana semua kapasitas berada Aurora Serverless v2. Atau Anda dapat mengonfigurasi kombinasi Aurora Serverless v2 dan instance yang disediakan sesuai permintaan atau cadangan. Untuk informasi tentang bagaimana Aurora Serverless v2 ACUsbekerja, lihatBagaimana Aurora Serverless v2 cara kerja.
- Aurora Postgre SQL Limitless Database — Aurora SQL Postgre Limitless Database adalah kemampuan penskalaan horizontal otomatis yang melampaui batas throughput penulisan dan penyimpanan instans DB tunggal. Limitless Database mendistribusikan beban kerja melalui

beberapa instance penulis Aurora, sambil mempertahankan kemudahan pengoperasian sebagai database tunggal. Limitless Database menyediakan kapasitas sesuai permintaan di mana unit penagihan adalah unit kapasitas Aurora (ACU) jam dalam grup pecahan DB. Untuk informasi tentang cara ACUs kerja Limitless Database, lihatMembuat cluster DB yang menggunakan Aurora SQL Postgre Limitless Database.

Untuk mengetahui informasi tentang harga Aurora, lihat Halaman harga Aurora.

Topik

- Instans Sesuai Permintaan untuk Aurora
- untuk Amazon Aurora

# Instans Sesuai Permintaan untuk Aurora

Penagihan instans DB Amazon RDS sesuai permintaan didasarkan pada kelas instans DB (misalnya, db.t3.smal atau db.m5.large). Untuk informasi tentang harga Amazon RDS, lihat Halaman harga Amazon RDS.

Penagihan dimulai untuk instans DB segera setelah instans DB tersedia. Harga dicantumkan per jam, tetapi tagihan dihitung turun menjadi detik dan menunjukkan waktu dalam bentuk desimal. Penggunaan Amazon RDS ditagih dalam setiap kenaikan satu detik, dengan minimum 10 menit. Dalam hal perubahan konfigurasi yang dapat ditagih, seperti penghitungan skala atau kapasitas penyimpanan, Anda dikenai biaya minimum 10 menit. Penagihan berlanjut hingga instans DB berakhir, yaitu pada saat Anda menghapus instans DB atau jika instans DB gagal.

Jika tidak ingin dikenai biaya lagi untuk instans DB, Anda harus menghentikan atau menghapusnya agar tidak ada tagihan untuk instans DB tambahan per jam. Untuk informasi selengkapnya tentang status instans DB yang ditagih, lihat Melihat status instans DB di klaster Aurora.

## Instans DB tang dihentikan

Ketika instans DB Anda dihentikan, Anda akan dikenai biaya untuk penyimpanan yang disediakan, termasuk IOPS yang Tersedia. Anda juga dikenai biaya untuk penyimpanan cadangan, termasuk penyimpanan untuk snapshot manual dan cadangan otomatis dalam periode retensi yang Anda tentukan. Anda tidak dikenai biaya untuk jam instans DB.

## Instans DB Multi-AZ

Penyiapan multi-AZ meningkatkan daya tahan dan ketersediaan data dengan secara otomatis menyediakan dan memelihara replika siaga sinkron di Availability Zone yang berbeda. Karena sumber daya tambahan dan ketersediaan yang meningkat, penerapan multi-AZ dihargai lebih tinggi daripada penerapan Single-AZ, dan dapat menelan biaya sekitar dua kali lipat karena instance siaga tambahan dan sumber daya terkait.

Pertimbangkan detail penting berikut tentang harga Multi-AZ:

- Biaya komputasi: Ditagih per jam instans DB untuk instans primer dan siaga.
- Biaya penyimpanan: Dibebankan per GB-bulan untuk penyimpanan yang disediakan untuk instans primer dan siaga.
- Biaya transfer data: Replikasi antara instans primer dan siaga termasuk dalam biaya, tetapi biaya transfer data lainnya mungkin berlaku berdasarkan penggunaan Anda.

Untuk memperkirakan biaya bulanan Anda secara akurat berdasarkan kasus penggunaan spesifik Anda dan Wilayah AWS, Anda dapat menggunakan AWS Kalkulator Harga. Alat ini memungkinkan Anda untuk memasukkan detail konfigurasi Anda dan memberikan rincian biaya yang komprehensif.

> **Note**
>
> Harga dapat berubah sewaktu-waktu. Lihat halaman Harga Amazon RDS untuk up-to-date informasi terbanyak.

# untuk Amazon Aurora

Dengan menggunakan instans DB terpesan, Anda dapat memesan instans DB untuk jangka waktu satu atau tiga tahun. Instans DB terpesan memberikan diskon yang signifikan dibandingkan harga instans DB sesuai permintaan. Instans DB terpesan bukan merupakan instans fisik, melainkan diskon penagihan yang diterapkan untuk penggunaan instans DB tertentu sesuai permintaan dalam akun Anda. Diskon untuk instans DB terpesan terikat dengan jenis instans dan Wilayah AWS.

Proses umum untuk menggunakan instans DB terpesan adalah: Pertama, dapatkan informasi tentang penawaran instans DB terpesan yang tersedia, kemudian beli penawaran instans DB terpesan, dan terakhir dapatkan informasi tentang instans DB terpesan yang ada.

Untuk informasi tentang pembelian instans DB cadangan dan melihat penagihan untuk instans DB cadangan, lihat bagian berikut.

•
•

## Gambaran umum instans DB terpesan

Saat membeli instans DB terpesan di Amazon RDS, Anda membeli komitmen untuk mendapatkan tarif diskon, pada jenis instans DB tertentu, selama durasi instans DB terpesan. Untuk menggunakan instans DB terpesan Amazon RDS, Anda membuat instans DB baru seperti yang Anda lakukan untuk instans sesuai permintaan.

Instans DB baru yang Anda buat harus memiliki spesifikasi yang sama dengan instans DB terpesan untuk hal berikut:

- Wilayah AWS
- Mesin DB (Nomor versi mesin DB tidak perlu cocok.)
- Jenis instans DB

Jika spesifikasi instans DB baru cocok dengan instans DB terpesan yang sudah ada untuk akun Anda, Anda akan ditagih dengan tarif diskon yang ditawarkan untuk instans DB terpesan. Jika tidak, instans DB ditagih dengan tarif sesuai permintaan.

Anda dapat memodifikasi instans DB yang Anda gunakan sebagai instans DB terpesan. Jika modifikasi sesuai dengan spesifikasi instans DB terpesan, sebagian atau seluruh diskon masih

akan berlaku untuk instans DB yang dimodifikasi. Jika modifikasi berada di luar spesifikasi, seperti mengubah kelas instans, diskon tidak lagi berlaku. Untuk informasi selengkapnya, lihat Instans DB terpesan berukuran fleksibel.

Topik

- Jenis penawaran
- Fleksibilitas konfigurasi klaster DB Aurora
- Instans DB terpesan berukuran fleksibel
- Contoh penagihan instans DB terpesan Aurora
- Menghapus instans DB terpesan

Untuk informasi selengkapnya tentang instans DB terpesan, termasuk harga, lihat Instans terpesan Amazon RDS.

Jenis penawaran

Instans DB terpesan tersedia dalam tiga jenis—Tidak Ada Di Muka, Di Muka Sebagian, dan Semua Di Muka—yang memungkinkan Anda mengoptimalkan biaya Amazon RDS berdasarkan perkiraan penggunaan Anda.

> **Note**
>
> Tidak semua kelas instans RDS mendukung semua jenis penawaran Instance Cadangan. Misalnya, beberapa kelas instance mungkin tidak menawarkan opsi No Upfront. Untuk mengonfirmasi ketersediaan, tinjau penawaran Instans Cadangan di AWS Management Console atau gunakan perintah. `describe-reserved-db-instances-offerings` AWS CLI

Tidak Ada Di Muka

Opsi ini menyediakan akses ke instans DB terpesan tanpa harus membayar di muka. Instans DB terpesan Tidak Ada Di Muka akan menagih tarif per jam yang didiskon untuk setiap jam dalam jangka waktu pemesanan, terlepas dari penggunaannya, dan tidak perlu membayar di muka. Opsi ini hanya tersedia sebagai pemesanan satu tahun.

Di Muka Sebagian

Opsi ini mengharuskan pembayaran di muka untuk sebagian instans DB terpesan. Sisa jam dalam jangka waktu pemesanan akan ditagih dengan tarif per jam yang didiskon, terlepas dari penggunaannya. Opsi ini adalah pengganti untuk opsi Pemanfaatan Berat sebelumnya.

Semua Di Muka

Pembayaran penuh dilakukan di awal jangka waktu pemesanan, tanpa biaya lain untuk sisa jangka waktu pemesanan, terlepas dari jumlah jam yang digunakan.

Jika Anda menggunakan penagihan gabungan, semua akun dalam organisasi akan dianggap sebagai satu akun. Artinya, semua akun dalam organisasi dapat menerima manfaat biaya per jam dari instans DB terpesan yang dibeli oleh akun lain. Untuk informasi selengkapnya tentang penagihan gabungan, lihat Instans DB terpesan Amazon RDS dalam Panduan AWS Billing and Cost Management.

Fleksibilitas konfigurasi klaster DB Aurora

Anda dapat menggunakan instans DB terpesan Aurora dengan kedua konfigurasi klaster DB:

- Aurora I/O-Optimized — Anda hanya membayar untuk penggunaan dan penyimpanan cluster DB Anda, tanpa biaya tambahan untuk operasi I/O baca dan tulis.
- Aurora Standard — Selain penggunaan dan penyimpanan cluster DB Anda, Anda juga membayar tarif standar per 1 juta permintaan untuk operasi I/O.

Aurora secara otomatis memperhitungkan perbedaan harga antara konfigurasi ini. Aurora I/O-Optimized mengkonsumsi 30% lebih banyak unit yang dinormalisasi per jam dari Aurora Standard.

Untuk informasi selengkapnya tentang konfigurasi penyimpanan klaster Aurora, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang harga untuk konfigurasi penyimpanan klaster Aurora, lihat Harga Amazon Aurora.

Instans DB terpesan berukuran fleksibel

Saat membeli instans DB terpesan, satu hal yang Anda tentukan adalah kelas instans, misalnya, db.r5.large. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB.

Jika Anda memiliki instans DB, dan perlu menskalakan kapasitasnya ke ukuran yang lebih besar, instans DB terpesan Anda secara otomatis akan diterapkan pada instans DB yang diskalakan. Artinya, instans DB terpesan Anda secara otomatis diterapkan ke semua ukuran kelas instans DB. Instans DB cadangan yang fleksibel dengan ukuran tersedia untuk instans DB dengan mesin database yang sama. Wilayah AWS Instans DB terpesan berukuran fleksibel hanya dapat diskalakan di jenis kelas instansnya. Misalnya, instans DB terpesan untuk db.r5.large dapat diterapkan ke db.r5.xlarge, tetapi tidak ke db.r6g.large, karena db.r5 dan db.r6g adalah jenis kelas instans yang berbeda.

Instans DB terpesan berukuran fleksibel tersedia untuk mesin basis data Amazon RDS berikut:

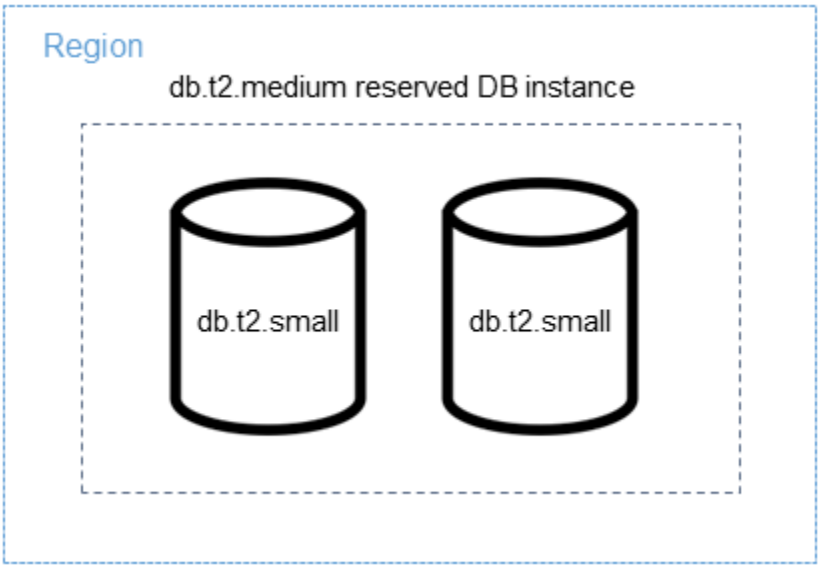
- Aurora MySQL
- Aurora PostgreSQL

Anda dapat membandingkan penggunaan untuk ukuran instans DB terpesan yang berbeda dengan menggunakan unit per jam yang dinormalkan. Misalnya, satu unit penggunaan di dua instans DB db.r3.large setara dengan delapan unit per jam penggunaan yang dinormalkan di satu db.r3.small. Tabel berikut menunjukkan jumlah unit per jam yang dinormalkan untuk setiap ukuran instans DB.

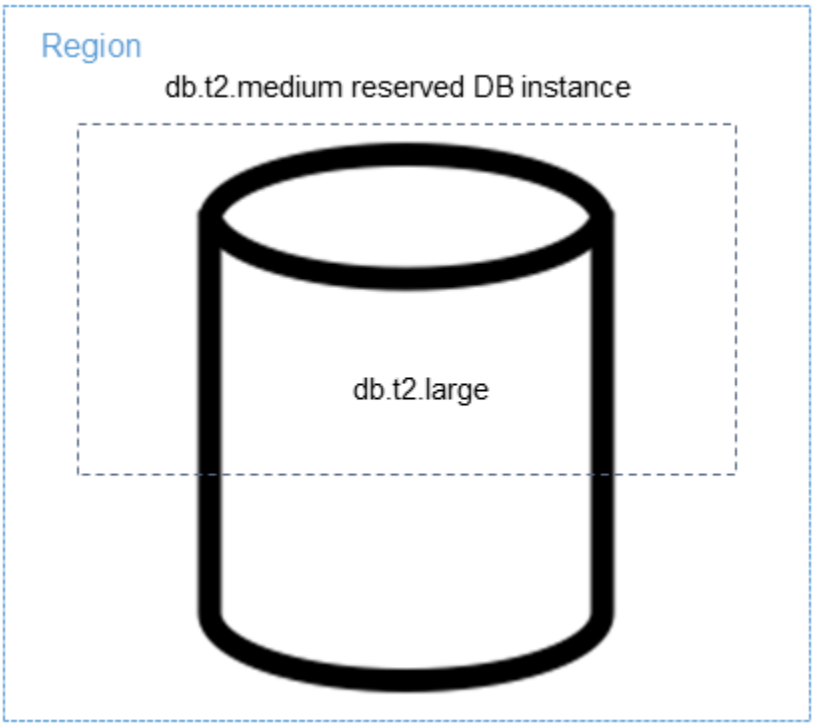
| Ukuran instans | Unit dinormalisasi per jam untuk satu instans DB, Aurora Standard | Unit dinormalisasi per jam untuk satu instans DB, Aurora I/O-Optimized | Unit dinormalisasi per jam untuk tiga instans DB (penulis dan dua pembaca), Aurora Standard | Unit dinormalisasi per jam untuk tiga instans DB (penulis dan dua pembaca), Aurora I/O-Optimized |
|---|---|---|---|---|
| small | 1 | 1.3 | 3 | 3.9 |
| medium | 2 | 2.6 | 6 | 7.8 |
| large | 4 | 5.2 | 12 | 15.6 |
| xlarge | 8 | 10.4 | 24 | 31.2 |
| 2xlarge | 16 | 20.8 | 48 | 62.4 |

| Ukuran instans | Unit dinormalisasi per jam untuk satu instans DB, Aurora Standard | Unit dinormalisasi per jam untuk satu instans DB, Aurora I/O-Optimized | Unit dinormalisasi per jam untuk tiga instans DB (penulis dan dua pembaca), Aurora Standard | Unit dinormalisasi per jam untuk tiga instans DB (penulis dan dua pembaca), Aurora I/O-Optimized |
|---|---|---|---|---|
| 4xlarge | 32 | 41,6 | 96 | 124,8 |
| 8xlarge | 64 | 83.2 | 192 | 249.6 |
| 12xlarge | 96 | 124,8 | 288 | 374.4 |
| 16xlarge | 128 | 166.4 | 384 | 499.2 |
| 24xlarge | 192 | 249.6 | 576 | 748.8 |
| 32xlarge | 256 | 332.8 | 768 | 998,4 |

Misalnya, Anda membeli instans DB terpesan `db.t2.medium`, dan Anda memiliki dua instans DB `db.t2.small` yang berjalan di akun di pada Wilayah AWS yang sama. Dalam hal ini, manfaat penagihan diterapkan sepenuhnya pada kedua instans.



Atau, jika Anda memiliki satu `db.t2.large` instans yang berjalan di akun Anda dalam hal yang sama Wilayah AWS, manfaat penagihan diterapkan ke 50 persen dari penggunaan instans DB.




> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.

Contoh penagihan instans DB terpesan Aurora

Contoh berikut menggambarkan harga untuk instans DB yang dicadangkan untuk cluster Aurora DB menggunakan keduanya Aurora Standard and Aurora I/O-Optimized Konfigurasi cluster DB.

Contoh menggunakan Aurora Standard

Harga untuk instans DB terpesan tidak memberikan diskon untuk biaya yang terkait dengan penyimpanan, cadangan, dan I/O. Contoh berikut menggambarkan total biaya per bulannya untuk instans DB terpesan:

- Kelas instans DB db.r5.large AZ Tunggal terpesan untuk Aurora MySQL di AS Timur (Virginia Utara) dengan biaya sebesar $0,19 per jam, atau $138,70 per bulan
- Penyimpanan Aurora dengan biaya sebesar $0,10 per GiB per bulan (dengan asumsi $45,60 per bulan untuk contoh ini)
- Aurora I/O dengan biaya $0,20 per 1 juta permintaan (dengan asumsi $20 per bulan untuk contoh ini)
- Penyimpanan cadangan Aurora dengan biaya sebesar $0,021 per GiB per bulan (dengan asumsi $30 per bulan untuk contoh ini)

Tambahkan semua biaya ini ($138,70 + $45,60 + $20 + $30) dengan instans DB terpesan, dan total biaya per bulannya adalah $234,30.

Jika Anda memilih untuk menggunakan instans DB sesuai permintaan dan bukannya instans DB terpesan, biaya kelas instans DB db.r5.large AZ Tunggal pada Aurora MySQL di AS Timur (Virginia Utara) adalah sebesar $0,29 per jam, atau $217,50 per bulan. Jadi, untuk instans DB sesuai permintaan, tambahkan semua opsi ini ($217,50 + $45,60 + $20 + $30), dan total biaya per bulannya adalah $313,10. Anda menghemat hampir $79 per bulan dengan menggunakan instans DB terpesan.

Contoh menggunakan Aurora Standard Cluster DB dengan dua instance pembaca

Untuk menggunakan instans terpesan untuk klaster DB Aurora, cukup beli satu instans terpesan untuk setiap instans DB dalam klaster.

Berdasarkan contoh pertama, Anda memiliki klaster DB MySQL Aurora dengan satu instans DB penulis dan dua Replika Aurora, dengan total tiga instans DB di klaster. Kedua Replika Aurora tidak dikenai biaya penyimpanan atau cadangan tambahan. Jika Anda membeli tiga instans DB terpesan Aurora MySQL db.r5.large, biaya Anda akan menjadi $234,30 (untuk instans DB penulis) + 2 * ($138,70 + $20 I/O per Replika Aurora), dengan total $551,70 per bulan.

Biaya sesuai permintaan yang sesuai untuk klaster DB MySQL Aurora dengan satu instans DB penulis dan dua Replika Aurora adalah $313,10 + 2 * ($217,50 + $20 I/O per instans) dengan total $788,10 per bulan. Anda menghemat $236,40 per bulan dengan menggunakan instans DB terpesan.

Contoh menggunakan Aurora I/O-Optimized

Anda dapat menggunakan kembali yang sudah ada Aurora Standard instans DB yang dicadangkan dengan Aurora I/O-Optimized. Untuk sepenuhnya menggunakan manfaat diskon instans cadangan Anda dengan Aurora I/O-Optimized, Anda dapat membeli 30% instans cadangan tambahan yang mirip dengan instans cadangan Anda saat ini.

Tabel berikut menunjukkan contoh cara memperkirakan instance cadangan tambahan saat menggunakan Aurora I/O-Optimized. Jika instans cadangan yang diperlukan adalah sebagian kecil, Anda dapat memanfaatkan fleksibilitas ukuran yang tersedia dengan instans cadangan untuk mendapatkan bilangan bulat. Dalam contoh ini, “saat ini” mengacu pada Aurora Standard contoh cadangan yang Anda miliki sekarang. Contoh cadangan tambahan adalah jumlah Aurora Standard instans cadangan yang harus Anda beli untuk mempertahankan diskon instans cadangan Anda saat menggunakan Aurora I/O-Optimized.

| Kelas instans DB | Saat ini Aurora Standard contoh yang dicadangkan | Instans cadangan diperlukan untuk Aurora I/O-Optimized | Instans terpesan tambahan yang diperlukan | Instans terpesan tambahan yang diperlukan, menggunakan fleksibilitas ukuran |
|---|---|---|---|---|
| db.r6g.large | 10 | 10 * 1,3 = 13 | 3 * db.r6g.large | 3 * db.r6g.large |
| db.r6g.4xlarge | 20 | 20 * 1,3 = 26 | 6 * db.r6g.4xlarge | 6 * db.r6g.4xlarge |
| db.r6g.12xlarge | 5 | 5 * 1,3 = 6,5 | 1,5 * db.r6g.12xlarge | Masing-masing satu db.r6g.12xlarge, r6g.4xlarge, dan r6g.2xlarge<br><br>(0,5 * db.r6g.12xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2xlarge) |
| db.r6i.24xlarge | 15 | 15 * 1,3 = 19,5 | 4,5 * db.r6i.24xlarge | 4 * db.r6i.24xlarge + 1 * db.r6i.12xlarge |

| Kelas instans DB | Saat ini Aurora Standard contoh yang dicadangkan | Instans cadangan diperlukan untuk Aurora I/O-Optimized | Instans terpesan tambahan yang diperlukan | Instans terpesan tambahan yang diperlukan, menggunakan fleksibilitas ukuran |
|---|---|---|---|---|
| | | | | (0,5 * db.r6i.24xlarge = 1 * db.r6i.12xlarge) |

Contoh menggunakan Aurora I/O-Optimized Cluster DB dengan dua instance pembaca

Anda memiliki klaster DB MySQL Aurora dengan satu instans DB penulis dan dua Replika Aurora, dengan total tiga instans DB di klaster. Mereka menggunakan Aurora I/O-Optimized Konfigurasi cluster DB. Untuk menggunakan instans DB terpesan untuk klaster ini, Anda perlu membeli empat instans DB terpesan dengan kelas instans DB yang sama. Tiga instans DB menggunakan Aurora I/O-Optimized mengkonsumsi 3,9 unit dinormalisasi per jam, dibandingkan dengan 3 unit dinormalisasi per jam untuk tiga instans DB yang menggunakan Aurora Standard. Namun, Anda menghemat biaya I/O bulanan untuk setiap instans DB.

> **Note**
>
> Harga dalam contoh ini adalah sampel harga dan mungkin tidak sama dengan harga sebenarnya. Untuk informasi harga Aurora, lihat Harga Amazon Aurora.

Menghapus instans DB terpesan

Syarat untuk instans DB terpesan mencakup komitmen satu tahun atau tiga tahun. Anda tidak dapat membatalkan instans DB terpesan. Namun, Anda dapat menghapus instans DB yang dicakup oleh diskon instans DB terpesan. Proses penghapusan instans DB yang dicakup oleh diskon instans DB terpesan sama dengan instans DB lainnya.

Anda ditagih untuk biaya di muka, terlepas dari apakah Anda menggunakan sumber daya atau tidak.

Jika Anda menghapus instans DB yang dicakup oleh diskon instans DB terpesan, Anda dapat meluncurkan instans DB lain dengan spesifikasi yang kompatibel. Dalam hal ini, Anda tetap mendapatkan tarif diskon selama jangka waktu pemesanan (satu atau tiga tahun).

Anda dapat menggunakan AWS Management Console, the AWS CLI, dan RDS API untuk bekerja dengan instans DB cadangan.

Konsol

Anda dapat menggunakan AWS Management Console untuk bekerja dengan instans DB cadangan seperti yang ditunjukkan dalam prosedur berikut.

Untuk mendapatkan harga dan informasi tentang penawaran instans DB yang tersedia

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans terpesan.
3. Pilih Beli Instans DB Terpesan.
4. Untuk Deskripsi produk, pilih mesin DB dan jenis lisensi.
5. Untuk Kelas instans DB, pilih kelas instans DB.
6. Untuk Opsi Deployment, pilih apakah Anda menginginkan deployment instans DB AZ Tunggal atau Multi-AZ.

   > **Note**
   >
   > Instans Amazon Aurora terpesan selalu memiliki opsi deployment yang ditetapkan ke Instans DB ber-AZ Tunggal. Namun, saat Anda membuat klaster DB Aurora, opsi deployment default adalah Buat Replika Aurora atau Pembaca di AZ (Multi-AZ) yang berbeda.
   > Anda harus membeli instans DB terpesan untuk setiap instans yang Anda akan gunakan, termasuk Replika Aurora. Oleh karena itu, untuk deployment Multi-AZ di Aurora, Anda harus membeli instans DB terpesan tambahan.

7. Untuk Jangka waktu, pilih jangka waktu untuk memesan instans DB.
8. Untuk Jenis penawaran, pilih jenis penawaran.

   Setelah memilih jenis penawaran, Anda dapat melihat informasi harga.

> **Important**
>
> Pilih Batalkan untuk menghindari pembelian instans DB terpesan dan biaya yang dikenakan.

Setelah Anda memiliki informasi tentang penawaran instans DB terpesan yang tersedia, Anda dapat menggunakan informasi tersebut untuk membeli penawaran sebagaimana ditunjukkan dalam prosedur berikut.

Untuk membeli instans DB terpesan

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans terpesan.
3. Pilih Beli instans DB terpesan.
4. Untuk Deskripsi produk, pilih mesin DB dan jenis lisensi.
5. Untuk Kelas instans DB, pilih kelas instans DB.
6. Untuk Deployment Multi-AZ, pilih apakah Anda menginginkan deployment instans DB AZ Tunggal atau Multi-AZ.

   > **Note**
   >
   > Instans Amazon Aurora terpesan selalu memiliki opsi deployment yang ditetapkan ke Instans DB ber-AZ Tunggal. Saat Anda membuat klaster DB Amazon Aurora dari instans DB terpesan, klaster DB secara otomatis dibuat sebagai Multi-AZ. Pastikan untuk membeli instans DB terpesan untuk setiap instans DB yang Anda akan gunakan, termasuk Replika Aurora.

7. Untuk Jangka waktu, pilih jangka waktu yang Anda inginkan untuk pemesanan instans DB.
8. Untuk Jenis penawaran, pilih jenis penawaran.

   Setelah memilih jenis penawaran, Anda dapat melihat informasi harga.
9. (Opsional) Anda dapat menetapkan pengidentifikasi Anda sendiri ke instans DB terpesan yang Anda beli untuk membantu Anda melacaknya. Untuk Id Terpesan, ketik pengidentifikasi untuk instans DB terpesan Anda.

10. Pilih Kirim.

    Instans DB terpesan Anda dibeli, lalu ditampilkan dalam daftar Instans terpesan.

Setelah membeli instans DB terpesan, Anda dapat memperoleh informasi tentang instans DB terpesan Anda seperti yang ditunjukkan dalam prosedur berikut.

Untuk mendapatkan informasi tentang instans DB cadangan untuk akun Anda AWS

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel Navigasi, pilih Instans terpesan.

   Instans DB terpesan untuk akun Anda akan muncul. Untuk melihat informasi terperinci tentang instans DB terpesan tertentu, pilih instans tersebut dalam daftar. Kemudian, Anda dapat melihat informasi terperinci tentang instans tersebut dalam panel detail di bagian bawah konsol.

AWS CLI

Anda dapat menggunakan AWS CLI untuk bekerja dengan instans DB cadangan seperti yang ditunjukkan dalam contoh berikut.

Example mendapatkan penawaran instans DB terpesan yang tersedia

Untuk mendapatkan informasi tentang penawaran instans DB cadangan yang tersedia, hubungi perintah. AWS CLI `describe-reserved-db-instances-offerings`

```
aws rds describe-reserved-db-instances-offerings
```

Panggilan ini menghasilkan output seperti yang berikut ini:

```
OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
```

```
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

Setelah memiliki informasi tentang penawaran instans DB terpesan yang tersedia, Anda dapat menggunakan informasi tersebut untuk membeli penawaran.

Untuk membeli instans DB cadangan, gunakan AWS CLI perintah [purchase-reserved-db-instances-offering](#)dengan parameter berikut:

- `--reserved-db-instances-offering-id` – ID penawaran yang ingin Anda beli. Lihat contoh sebelumnya untuk mendapatkan ID penawaran.
- `--reserved-db-instance-id` – Anda dapat menetapkan pengindentifikasi Anda sendiri ke instans DB terpesan yang Anda beli untuk membantu melacaknya.

Example membeli instans DB terpesan

Contoh berikut membeli penawaran instans DB pesanan dengan ID *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*, dan menetapkan pengidentifikasi *MyReservation*.

Untuk Linux, macOS, atau Unix:

```
aws rds purchase-reserved-db-instances-offering \
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

Untuk Windows:

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

Perintah ini menghasilkan output seperti yang berikut ini:

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State           Description  Offering Type
```

```
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql        Partial  Upfront
```

Setelah membeli instans DB terpesan, Anda dapat memperoleh informasi tentang instans DB terpesan Anda.

Untuk mendapatkan informasi tentang instans DB cadangan untuk AWS akun Anda, hubungi AWS CLI perintah `describe-reserved-db-instances`, seperti yang ditunjukkan pada contoh berikut.

Example mendapatkan instans DB terpesan Anda

```
aws rds describe-reserved-db-instances
```

Perintah ini menghasilkan output seperti yang berikut ini:

```
RESERVATION  ReservationId     Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS API

Anda dapat menggunakan RDS API untuk bekerja dengan instans DB cadangan:

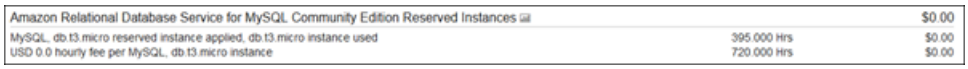
- Untuk mendapatkan informasi tentang penawaran instans DB cadangan yang tersedia, hubungi operasi Amazon RDSAPI. `DescribeReservedDBInstancesOfferings`
- Setelah memiliki informasi tentang penawaran instans DB terpesan yang tersedia, Anda dapat menggunakan informasi tersebut untuk membeli penawaran. Panggil `PurchaseReservedDBInstancesOffering`RDSAPIoperasi dengan parameter berikut:
  - `--reserved-db-instances-offering-id` – ID penawaran yang ingin Anda beli.
  - `--reserved-db-instance-id` – Anda dapat menetapkan pengindentifikasi Anda sendiri ke instans DB terpesan yang Anda beli untuk membantu melacaknya.
- Setelah membeli instans DB terpesan, Anda dapat memperoleh informasi tentang instans DB terpesan Anda. Panggil `DescribeReservedDBInstances`RDSAPIoperasi.

Anda dapat melihat penagihan untuk instans DB terpesan Anda di Dasbor Penagihan di AWS Management Console.

Untuk melihat penagihan instans DB terpesan

1. Masuk ke AWS Management Console.
2. Dari menu akun di kanan atas, pilih Dasbor Penagihan.
3. Pilih Detail Tagihan di kanan atas dasbor.
4. Di bagian Biaya Layanan AWS , perluas Layanan Basis Data Relasional.
5. Perluas Wilayah AWS lokasi instans DB cadangan Anda, misalnya US West (Oregon).

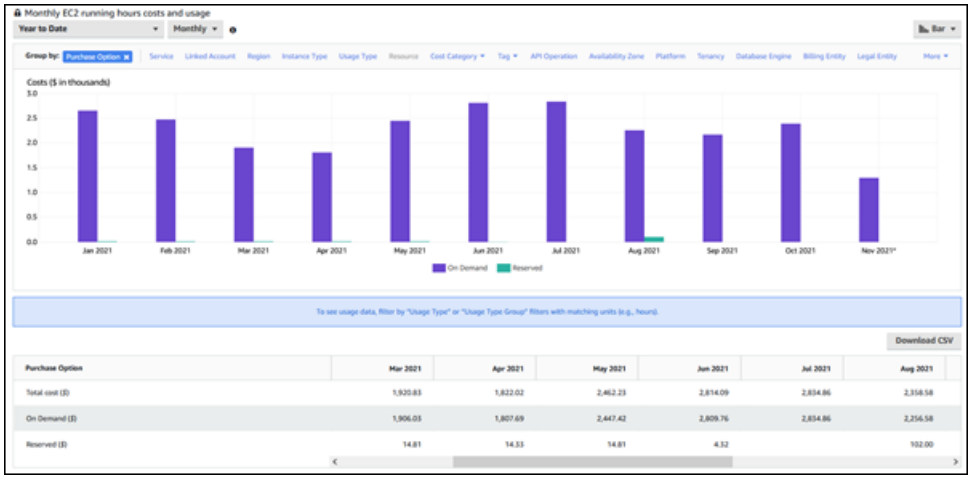
   Instans DB cadangan Anda dan biaya per jamnya untuk bulan berjalan ditampilkan di bawah Amazon Relational Database Service untuk ***`Database Engine`*** Instans  Cadangan.




   Instans DB terpesan dalam contoh ini dibeli dengan opsi Semua Di Muka, jadi tidak ada biaya per jam.
6. Pilih ikon Cost Explorer (grafik batang) di samping judul Instans Terpesan.

   Cost Explorer menampilkan grafik biaya dan penggunaan jam EC2 operasional Bulanan.
7. Kosongkan filter Grup Jenis Penggunaan di sebelah kanan grafik.
8. Pilih periode waktu dan unit waktu yang Anda inginkan untuk memeriksa biaya penggunaan.

   Contoh berikut menunjukkan biaya penggunaan untuk instans DB sesuai permintaan dan terpesan untuk tahun berjalan menurut bulan.




   Biaya instans DB terpesan dari Januari hingga Juni 2021 adalah biaya bulanan untuk instans Di Muka Sebagian, sedangkan biaya pada Agustus 2021 adalah biaya satu kali untuk instans Semua Di Muka.

Diskon instans terpesan untuk instans Di Muka Sebagian habis masa berlakunya pada Juni 2021, tetapi instans DB tidak dihapus. Setelah tanggal kedaluwarsa, biaya tersebut akan dikenakan berdasarkan tarif sesuai permintaan.

# Menyiapkan lingkungan Anda untuk Amazon Aurora

Sebelum Anda menggunakan Amazon Aurora untuk pertama kali, selesaikan tugas-tugas berikut.

Topik


- Mendaftar untuk Akun AWS
- Buat pengguna dengan akses administratif
- Memberikan akses programatis
- Menentukan persyaratan
- Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan


Jika Anda sudah memiliki Akun AWS, ketahui persyaratan Aurora Anda, dan lebih suka menggunakan default untuk grup keamanan IAM dan VPC, lewati saja. Memulai dengan Amazon Aurora

## Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

# Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

## Memberikan akses programatis

Pengguna membutuhkan akses terprogram jika mereka ingin berinteraksi dengan AWS luar. AWS Management Console Cara untuk memberikan akses terprogram tergantung pada jenis pengguna yang mengakses AWS.

Untuk memberi pengguna akses programatis, pilih salah satu opsi berikut.

| Pengguna mana yang membutuhkan akses programatis? | Untuk | Oleh |
| --- | --- | --- |
| Identitas tenaga kerja<br><br>(Pengguna yang dikelola di Pusat Identitas IAM) | Gunakan kredensi sementara untuk menandatangani permintaan terprogram ke AWS CLI,, AWS SDKs atau. AWS APIs | Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.<br><br>• Untuk AWS CLI, lihat Mengkonfigurasi yang akan AWS CLI digunakan AWS IAM Identity Center dalam Panduan AWS Command Line Interface Pengguna.<br>• Untuk AWS SDKs, alat, dan AWS APIs, lihat Autentikasi Pusat Identitas IAM di Panduan Referensi Alat AWS SDKs dan Alat. |

| Pengguna mana yang membutuhkan akses programatis? | Untuk | Oleh |
| --- | --- | --- |
| IAM | Gunakan kredensi sementara untuk menandatangani permintaan terprogram ke AWS CLI,, AWS SDKs atau. AWS APIs | Mengikuti petunjuk dalam Menggunakan kredensil sementara dengan AWS sumber daya di Panduan Pengguna IAM. |
| IAM | (Tidak direkomendasikan) Gunakan kredensi jangka panjang untuk menandata ngani permintaan terprogra m ke AWS CLI,, AWS SDKs atau. AWS APIs | Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.<br>• Untuk mengetahui AWS CLI, lihat Mengautentikasi menggunakan kredensil pengguna IAM di Panduan Pengguna.AWS Command Line Interface<br>• Untuk AWS SDKs dan alat, lihat Mengautentikasi menggunakan kredensil jangka panjang di Panduan Referensi Alat AWS SDKs dan Alat.<br>• Untuk AWS APIs, lihat Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM. |

## Menentukan persyaratan

Blok bangunan dasar Aurora adalah klaster DB. Klaster DB dapat berisi satu atau beberapa instans DB. Klaster DB menyediakan alamat jaringan yang disebut titik akhir klaster. Aplikasi Anda terhubung ke titik akhir klaster yang dibuka oleh klaster DB kapan pun aplikasi tersebut perlu mengakses basis

data yang dibuat dalam klaster DB tersebut. Informasi yang Anda tentukan saat membuat elemen konfigurasi kontrol klaster DB seperti memori, mesin dan versi basis data, konfigurasi jaringan, keamanan, serta periode pemeliharaan.

Sebelum membuat grup keamanan dan klaster DB, Anda harus mengetahui klaster DB dan kebutuhan jaringan Anda. Berikut beberapa hal penting yang perlu dipertimbangkan:

- Kebutuhan sumber daya – Apa saja kebutuhan memori dan prosesor untuk aplikasi atau layanan Anda? Anda akan menggunakan pengaturan ini saat menentukan kelas instans DB yang akan Anda gunakan saat membuat klaster DB. Untuk spesifikasi tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB.
- VPC, subnet, dan grup keamanan – Klaster DB Anda akan berada di cloud privat virtual (VPC). Aturan grup keamanan harus dikonfigurasi agar terhubung ke klaster DB. Daftar berikut menjelaskan aturan untuk setiap opsi VPC:
  - VPC default - Jika AWS akun Anda memiliki VPC default di Wilayah AWS , VPC tersebut dikonfigurasi untuk mendukung cluster DB. Jika Anda menentukan VPC default saat membuat klaster DB:
    - Anda harus membuat grup keamanan VPC yang mengizinkan koneksi dari aplikasi atau layanan ke klaster DB Aurora. Gunakan opsi Grup Keamanan di konsol VPC atau AWS CLI untuk membuat grup keamanan VPC. Untuk informasi, lihat Langkah 3: Buat grup keamanan VPC.
    - Anda harus menentukan grup subnet DB default. Jika ini adalah cluster DB pertama yang Anda buat di AWS Wilayah, Amazon RDS akan membuat grup subnet DB default saat membuat cluster DB.
  - VPC yang ditentukan pengguna – Jika Anda ingin menentukan VPC yang ditentukan pengguna ketika membuat klaster DB:
    - Anda harus membuat grup keamanan VPC yang mengizinkan koneksi dari aplikasi atau layanan ke klaster DB Aurora. Gunakan opsi Grup Keamanan di konsol VPC atau AWS CLI untuk membuat grup keamanan VPC. Untuk informasi, lihat Langkah 3: Buat grup keamanan VPC.
    - VPC harus memenuhi persyaratan tertentu untuk meng-hosting klaster DB, seperti memiliki setidaknya dua subnet, yang masing-masingnya berada di zona ketersediaan terpisah. Untuk informasi, lihat Amazon VPC dan Aurora.
    - Anda harus menentukan grup subnet DB yang menentukan subnet mana di VPC tersebut yang dapat digunakan oleh klaster DB. Untuk informasi, lihat bagian Grup Subnet DB di Bekerja dengan klaster DB dalam VPC.

- Ketersediaan tinggi: Apakah Anda memerlukan dukungan failover? Di Aurora, deployment Multi-AZ menciptakan instans primer dan Replika Aurora. Anda dapat mengonfigurasi instans primer dan Replika Aurora agar berada di Zona Ketersediaan yang berbeda untuk dukungan failover. Kami merekomendasikan deployment Multi-AZ untuk beban kerja produksi agar menjaga ketersediaan yang tinggi. Untuk tujuan pengembangan dan pengujian, Anda dapat menggunakan non-Multi-AZ penerapan. Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.
- Kebijakan IAM: Apakah AWS akun Anda memiliki kebijakan yang memberikan izin yang diperlukan untuk melakukan operasi Amazon RDS? Jika Anda terhubung AWS menggunakan kredensil IAM, akun IAM Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk melakukan operasi Amazon RDS. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.
- Port terbuka: TCP/IP Port apa yang akan didengarkan database Anda? Firewall di beberapa perusahaan mungkin memblokir koneksi ke port default untuk mesin basis data Anda. Jika firewall perusahaan Anda memblokir port default, pilih port lain untuk klaster DB baru. Perhatikan bahwa setelah membuat klaster DB yang mendengarkan di port yang Anda tentukan, Anda dapat mengubah port tersebut dengan mengubah klaster DB.
- AWS Wilayah: AWS Wilayah mana Anda ingin database Anda berada? Memiliki basis data yang dekat dengan aplikasi atau layanan web dapat mengurangi latensi jaringan. Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.

Setelah mendapatkan informasi yang Anda perlukan untuk membuat grup keamanan dan klaster DB, lanjutkan ke langkah berikutnya.

## Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan

Klaster DB Anda akan dibuat di VPC. Grup keamanan memberikan akses ke klaster DB dalam VPC. Grup tersebut bertindak sebagai firewall untuk klaster DB terkait, yang mengontrol lalu lintas masuk dan keluar pada tingkat klaster. Klaster DB dibuat secara default dengan satu firewall dan grup keamanan default yang mencegah akses ke klaster DB. Oleh karena itu, Anda harus menambahkan aturan ke grup keamanan yang memungkinkan Anda untuk terhubung ke klaster DB Anda. Gunakan informasi jaringan dan konfigurasi yang Anda tentukan pada langkah sebelumnya untuk membuat aturan yang memungkinkan akses ke klaster DB Anda.

Misalnya, jika memiliki aplikasi yang akan mengakses basis data pada klaster DB Anda dalam VPC, Anda harus menambahkan aturan TCP kustom yang menentukan rentang port dan alamat IP yang

akan digunakan aplikasi untuk mengakses basis data. Jika Anda memiliki aplikasi di EC2 instans Amazon, Anda dapat menggunakan grup keamanan VPC yang Anda siapkan untuk instans Amazon EC2.

Anda dapat mengonfigurasi konektivitas antara EC2 instans Amazon, kluster DB saat membuat cluster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance.

> **Tip**
>
> Anda dapat mengatur konektivitas jaringan antara EC2 instans Amazon dan kluster DB secara otomatis saat membuat cluster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance.

Untuk informasi tentang cara menghubungkan sumber daya di Amazon Lightsail ke kluster DB, lihat Menghubungkan sumber daya Lightsail untuk menggunakan peering VPC. Layanan AWS

Untuk informasi selengkapnya tentang membuat VPC untuk digunakan dengan Aurora, lihat Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 . Untuk informasi tentang skenario umum untuk mengakses instans DB, lihat Skenario untuk mengakses cluster DB di VPC.

Untuk membuat grup keamanan VPC

1. Masuk ke AWS Management Console dan buka konsol VPC Amazon di https://console.aws.amazon.com /vpc.

   > **Note**
   >
   > Pastikan Anda berada di konsol VPC, bukan konsol RDS.

2. Di sudut kanan atas AWS Management Console, pilih AWS Wilayah tempat Anda ingin membuat grup keamanan VPC dan kluster DB. Dalam daftar sumber daya Amazon VPC untuk Wilayah AWS tersebut, Anda akan melihat setidaknya satu VPC dan beberapa subnet. Jika tidak, Anda tidak memiliki VPC default di Wilayah itu AWS .
3. Di panel navigasi, pilih Security Groups (Grup Keamanan).
4. Pilih Buat grup keamanan.

   Halaman Membuat grup keamanan akan muncul.

5. Dalam Detail basic, masukkan Nama grup keamanan dan Deskripsi. Untuk VPC, pilih VPC tempat Anda ingin membuat klaster DB.
6. Di bagian Aturan masuk, pilih Tambahkan aturan.
   a. Untuk Tipe, pilih TCP kustom.
   b. Untuk Rentang port, ketik nilai port yang akan digunakan untuk klaster DB Anda.
   c. Untuk Sumber, pilih nama grup keamanan atau ketik rentang alamat IP (nilai CIDR) dari tempat Anda mengakses klaster DB. Jika Anda memilih IP Saya, pilihan ini mengizinkan akses ke klaster DB dari alamat IP yang terdeteksi di browser Anda.
7. Jika Anda perlu menambahkan lebih banyak alamat IP atau rentang port yang berbeda, pilih Tambahkan aturan dan masukkan informasi untuk aturan tersebut.
8. (Opsional) Dalam Aturan keluar, tambahkan aturan untuk lalu lintas keluar. Secara default, semua lalu lintas keluar akan diizinkan.
9. Pilih Buat grup keamanan.

Anda dapat menggunakan grup keamanan VPC yang baru Anda buat sebagai grup keamanan untuk klaster DB Anda saat membuatnya.

> **Note**
>
> Jika Anda menggunakan VPC default, grup subnet default yang mencakup semua subnet VPC akan dibuat untuk Anda. Saat membuat klaster DB, Anda dapat memilih VPC default dan menggunakan default untuk Grup Subnet DB.

Setelah menyelesaikan persyaratan pengaturan, Anda dapat membuat klaster DB menggunakan persyaratan dan grup keamanan Anda dengan mengikuti petunjuk di Membuat klaster DB Amazon Aurora. Untuk informasi tentang memulai dengan membuat klaster DB yang menggunakan mesin DB tertentu, lihat Memulai dengan Amazon Aurora.

# Memulai dengan Amazon Aurora

Pada bagian ini, Anda dapat mengetahui cara membuat dan terhubung ke klaster Aurora DB menggunakan Amazon RDS.

Prosedur berikut adalah tutorial yang menunjukkan dasar-dasar memulai Aurora. Bagian selanjutnya memperkenalkan konsep dan prosedur Aurora lanjutan, seperti berbagai jenis titik akhir dan cara menskalakan klaster Aurora ke atas dan ke bawah.

> **Important**
>
> Sebelum dapat membuat atau terhubung ke klaster DB, pastikan untuk menyelesaikan tugas dalam Menyiapkan lingkungan Anda untuk Amazon Aurora.

Topik

- Membuat dan terhubung ke klaster DB Aurora MySQL
- Membuat dan terhubung ke klaster DB Aurora PostgreSQL
- Tutorial: Membuat server web dan klaster DB Amazon Aurora

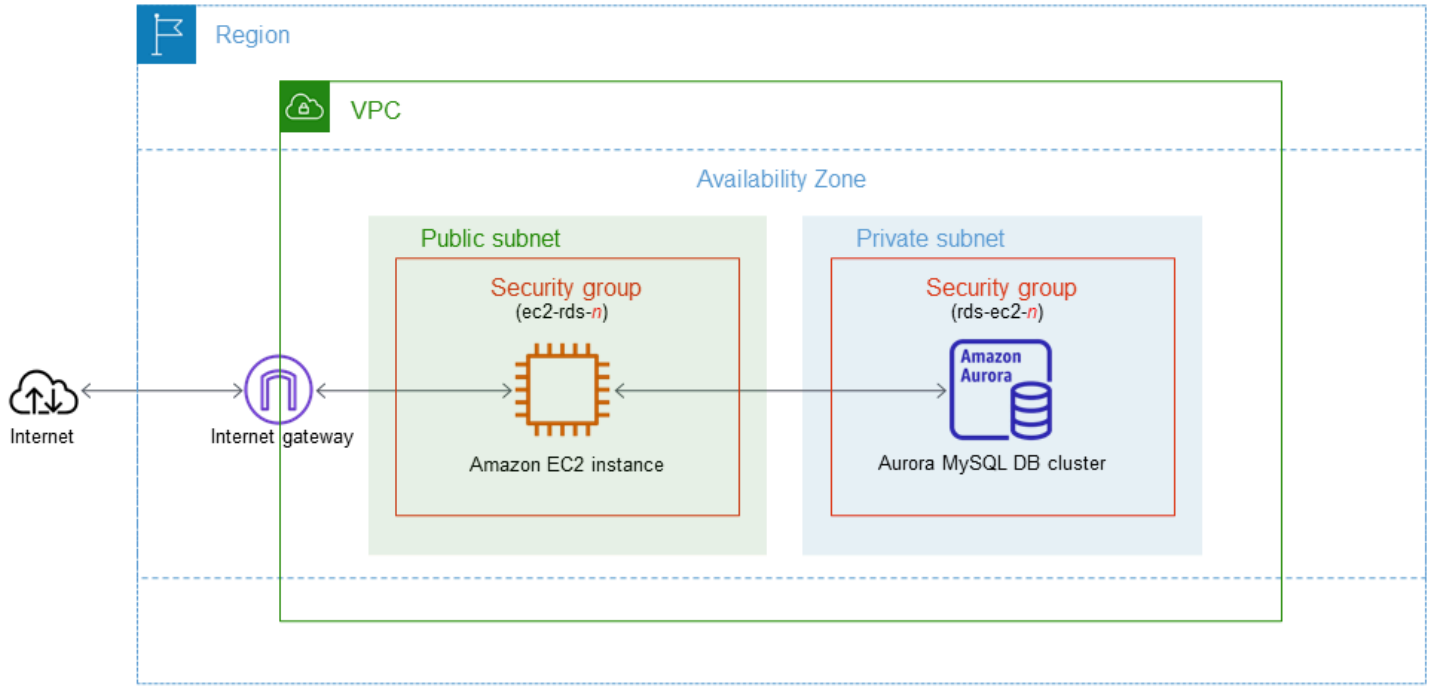
## Membuat dan terhubung ke klaster DB Aurora MySQL

Tutorial ini membuat EC2 instance dan cluster Aurora MySQL DB. Tutorial menunjukkan cara mengakses cluster DB dari EC2 instance menggunakan klien MySQL standar. Sebagai praktik terbaik, tutorial ini membuat klaster DB privat dalam cloud privat virtual (VPC). Dalam kebanyakan kasus, sumber daya lain dalam VPC yang sama, seperti EC2 instance, dapat mengakses cluster DB, tetapi sumber daya di luar VPC tidak dapat mengaksesnya.

Setelah Anda menyelesaikan tutorial, ada subnet publik dan privat di setiap Zona Ketersediaan di VPC Anda. Dalam satu Availability Zone, EC2 instance ada di subnet publik, dan instance DB ada di subnet pribadi.

> **Important**
>
> Tidak ada biaya untuk membuat AWS akun. Namun, dengan menyelesaikan tutorial ini, Anda mungkin dikenakan biaya untuk AWS sumber daya yang Anda gunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

Diagram berikut menunjukkan konfigurasi setelah tutorial selesai.




Tutorial ini memungkinkan Anda untuk membuat sumber daya Anda dengan menggunakan salah satu metode berikut:

1. Gunakan AWS Management Console - Langkah 1: Buat sebuah EC2 instance dan Langkah 2: Membuat klaster DB Aurora MySQL
2. Gunakan AWS CloudFormation untuk membuat instance database dan EC2 instance - (Opsional) Buat VPC, EC2 instance, dan cluster MySQL Aurora menggunakan AWS CloudFormation

Metode pertama menggunakan Easy create untuk membuat cluster Aurora MySQL DB pribadi dengan. AWS Management Console Di sini, Anda hanya menentukan jenis mesin DB, ukuran instans DB, dan pengidentifikasi cluster DB. Pembuatan mudah menggunakan pengaturan default untuk opsi konfigurasi lainnya.

Saat Anda menggunakan Standard create sebagai gantinya, Anda dapat menentukan lebih banyak opsi konfigurasi saat membuat cluster DB. Opsi ini mencakup pengaturan untuk ketersediaan, keamanan, cadangan, dan pemeliharaan. Untuk membuat klaster DB publik, Anda harus menggunakan Pembuatan standar. Untuk informasi, lihat the section called “Membuat klaster DB”.

Topik


- Prasyarat
- Langkah 1: Buat sebuah EC2 instance
- Langkah 2: Membuat klaster DB Aurora MySQL
- (Opsional) Buat VPC, EC2 instance, dan cluster MySQL Aurora menggunakan AWS CloudFormation
- Langkah 3: Membuat klaster DB Aurora MySQL
- Langkah 4: Hapus EC2 instance dan cluster DB
- (Opsional) Hapus EC2 instance dan cluster DB yang dibuat dengan CloudFormation
- (Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda


## Prasyarat

Sebelum memulai, selesaikan langkah-langkah di bagian berikut:

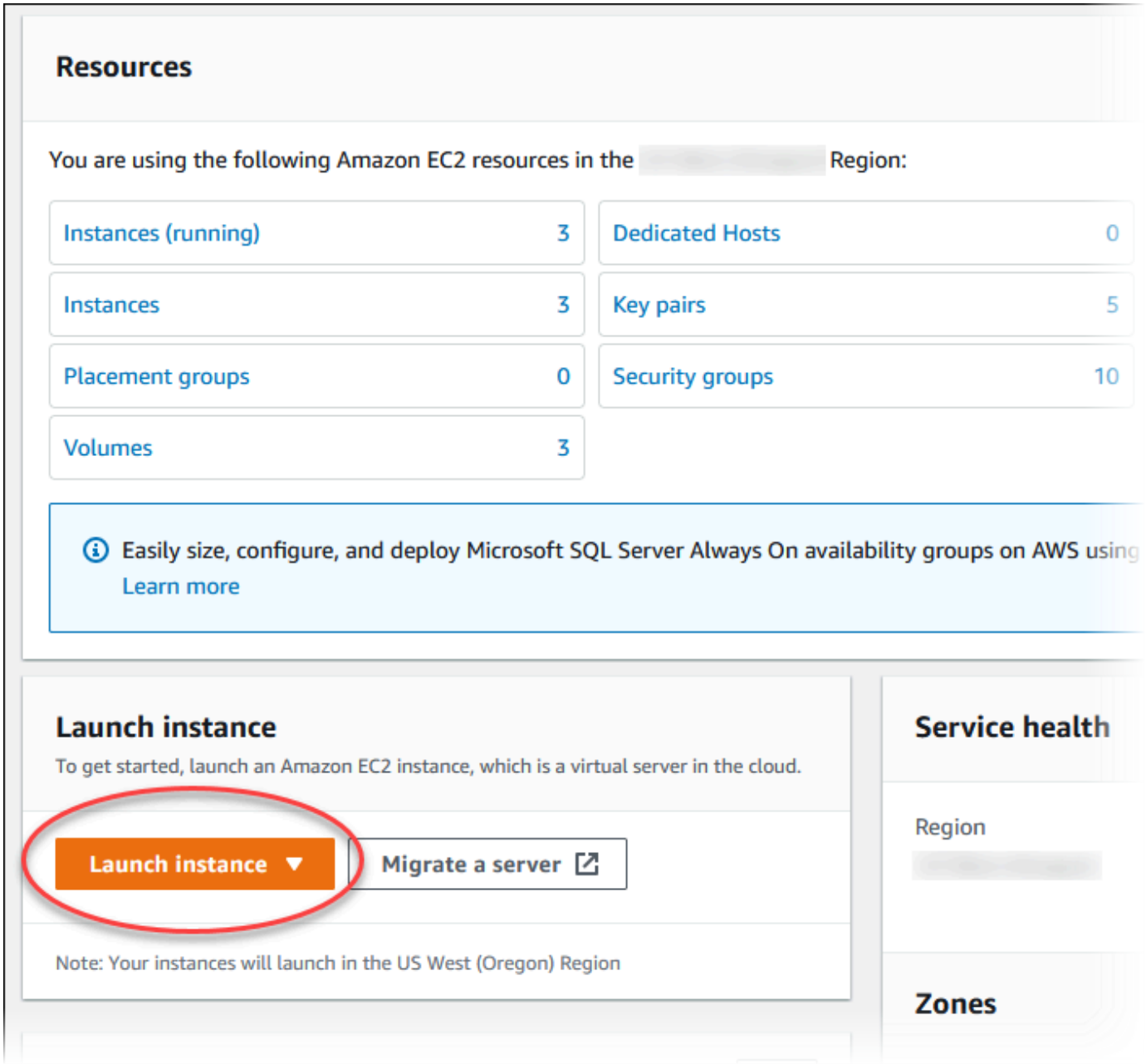
- Mendaftar untuk Akun AWS
- Buat pengguna dengan akses administratif

## Langkah 1: Buat sebuah EC2 instance

Buat EC2 instance Amazon yang akan Anda gunakan untuk terhubung ke database Anda.

Untuk membuat sebuah EC2 instance

1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda ingin membuat instance. EC2
3. Pilih EC2 Dasbor, lalu pilih Launch instance, seperti yang ditunjukkan pada gambar berikut.

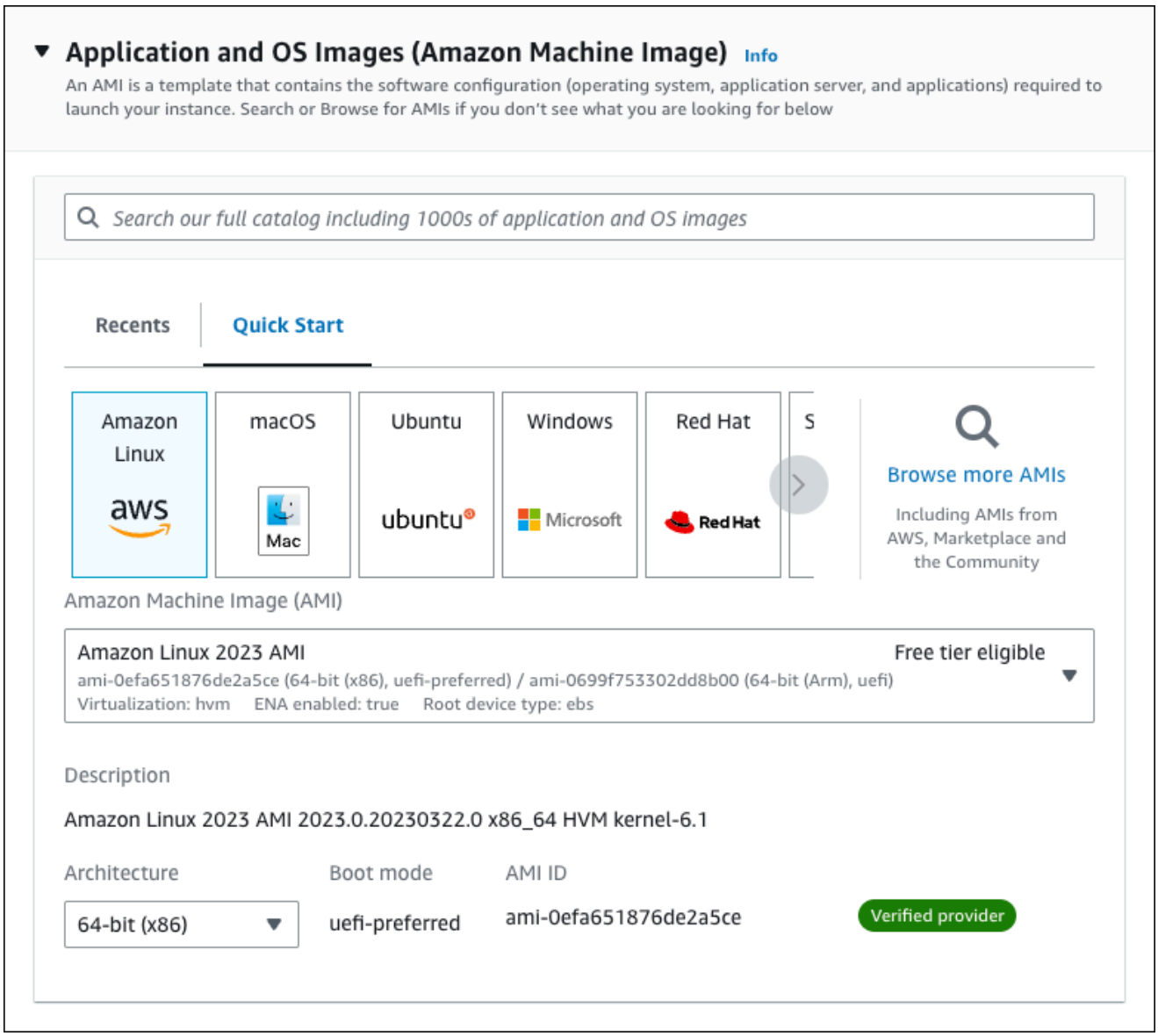



Halaman Meluncurkan instans akan terbuka.

4. Pilih pengaturan berikut di halaman Meluncurkan instans.

   a. Pada Nama dan tag, untuk Nama, masukkan **`ec2-database-connect`**.

   b. Pada Gambar Aplikasi dan OS (Amazon Machine Image), pilih Amazon Linux, lalu pilih AMI Amazon Linux 2023. Biarkan default untuk pilihan lainnya.




c. Pada Jenis instans, pilih t2.micro.

d. Pada Pasangan kunci (login), pilih Nama pasangan kunci untuk menggunakan pasangan kunci yang ada. Untuk membuat key pair baru untuk EC2 instance Amazon, pilih Create new key pair dan kemudian gunakan jendela Create key pair untuk membuatnya.

   Untuk informasi selengkapnya tentang membuat key pair baru, lihat Membuat key pair di Panduan EC2 Pengguna Amazon.

e. Untuk Izinkan lalu lintas SSH di pengaturan Jaringan, pilih sumber koneksi SSH ke instance. EC2

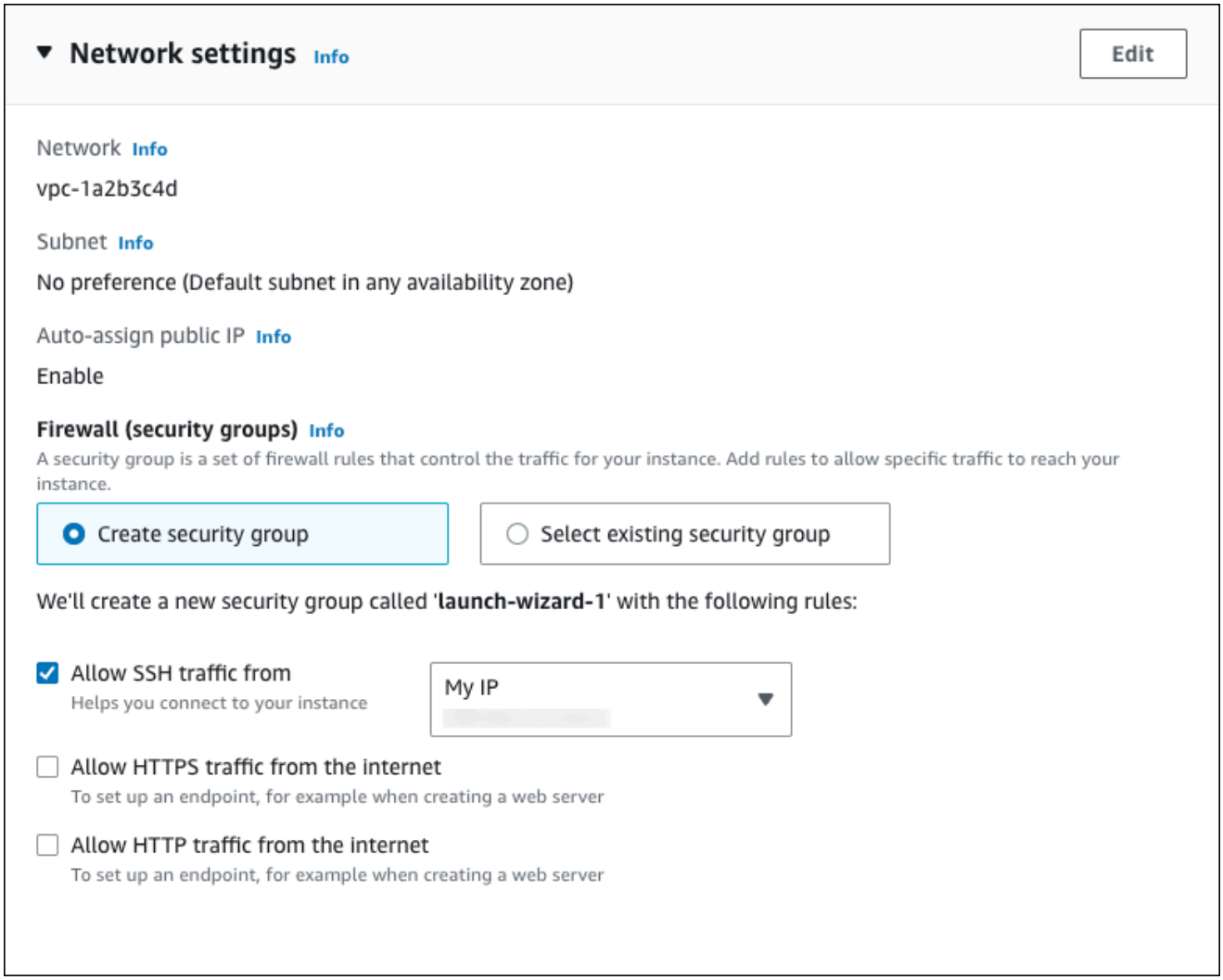
   Anda dapat memilih IP Saya jika alamat IP yang ditampilkan benar untuk koneksi SSH. Jika tidak, Anda dapat menentukan alamat IP yang akan digunakan untuk terhubung ke EC2

instance di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, di jendela atau tab browser yang berbeda, Anda dapat menggunakan layanan di https://checkip.amazonaws.com. Contoh alamat IP adalah 192.0.2.1/32.

Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, tentukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses EC2 instans publik Anda menggunakan SSH. Pendekatan ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, otorisasi hanya alamat IP tertentu atau rentang alamat untuk mengakses EC2 instans Anda menggunakan SSH.

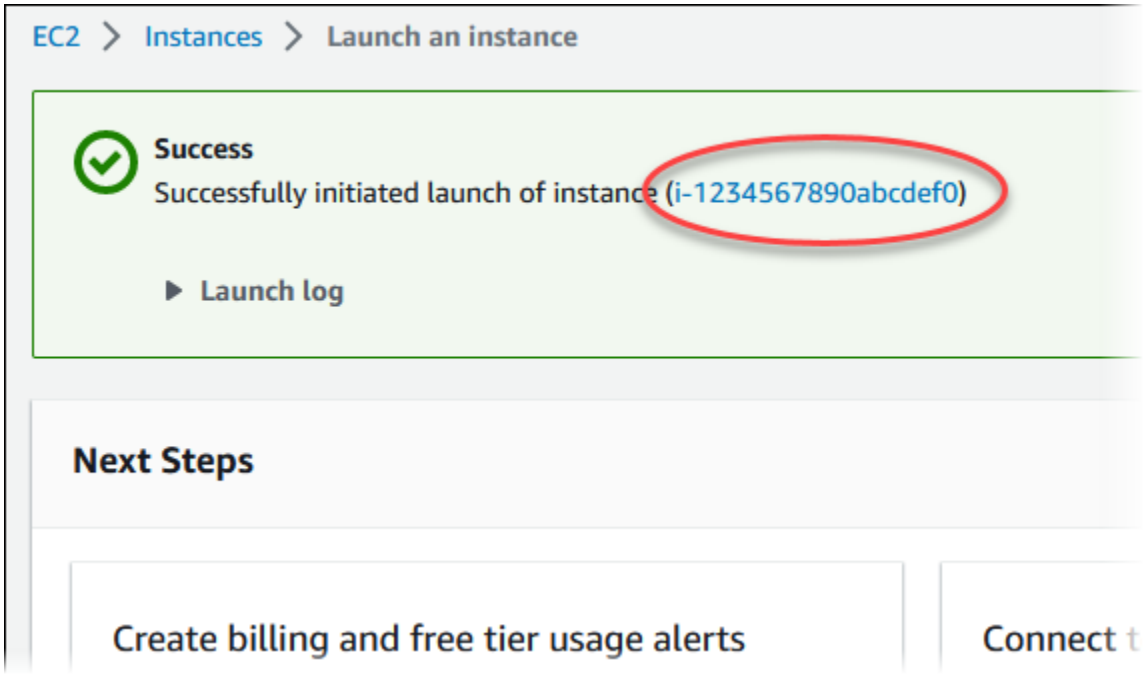
Gambar berikut menunjukkan contoh bagian Pengaturan jaringan.




f. Biarkan nilai default untuk bagian yang lainnya.

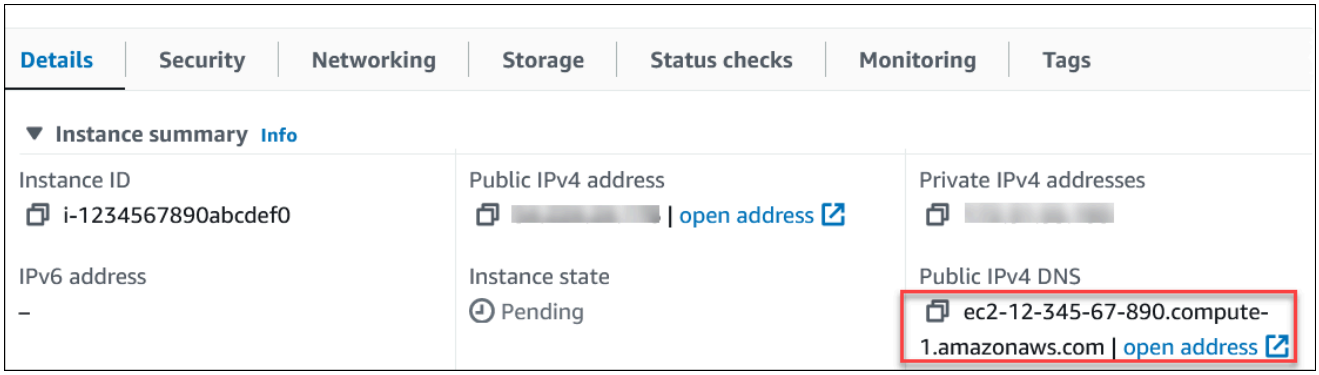
g. Tinjau ringkasan konfigurasi EC2 instans Anda di panel Summary, dan ketika Anda siap, pilih Launch instance.

5. Pada halaman Status Peluncuran, perhatikan pengenal untuk EC2 instance baru Anda, misalnya:`i-1234567890abcdef0`.

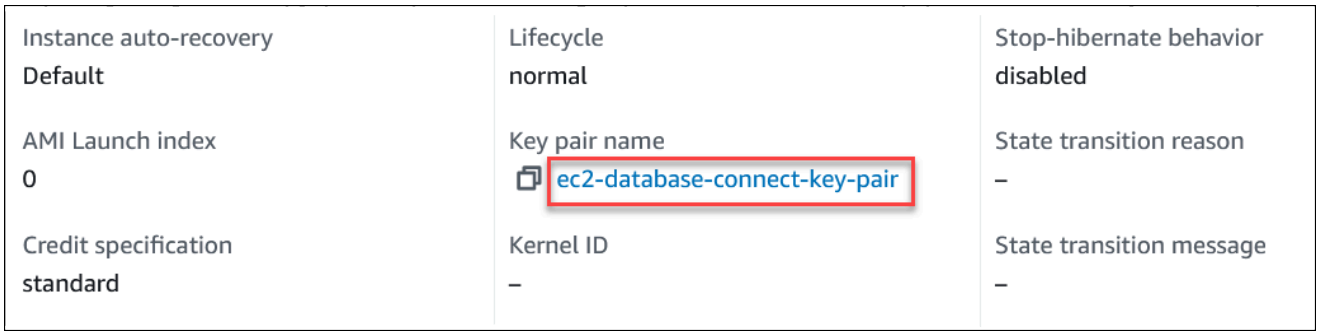



6. Pilih pengenal EC2 instance untuk membuka daftar EC2 instance, lalu pilih instance Anda EC2 .
7. Di tab Detail, catat nilai-nilai berikut, yang akan Anda butuhkan saat menghubungkan menggunakan SSH:

   a. Dalam ringkasan Instance, perhatikan nilai untuk  IPv4 DNS Publik.




   b. Di Detail instans, catat nilai untuk Nama pasangan kunci.



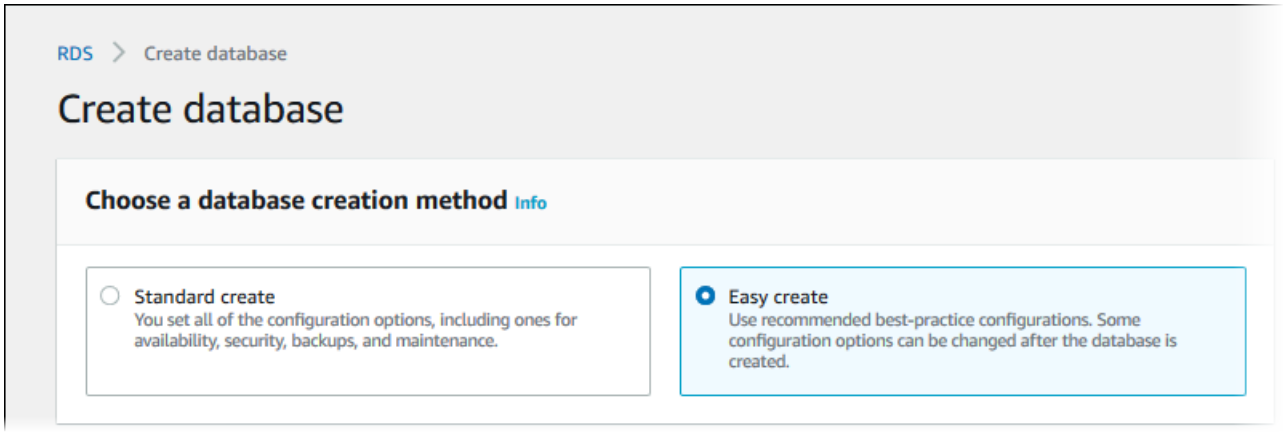
8. Tunggu hingga status Instance untuk EC2 instans Anda memiliki status Running sebelum melanjutkan.

## Langkah 2: Membuat klaster DB Aurora MySQL

Dalam contoh ini, Anda menggunakan Pembuatan mudah untuk membuat klaster DB Aurora MySQL dengan kelas instans DB db.r6g.large.

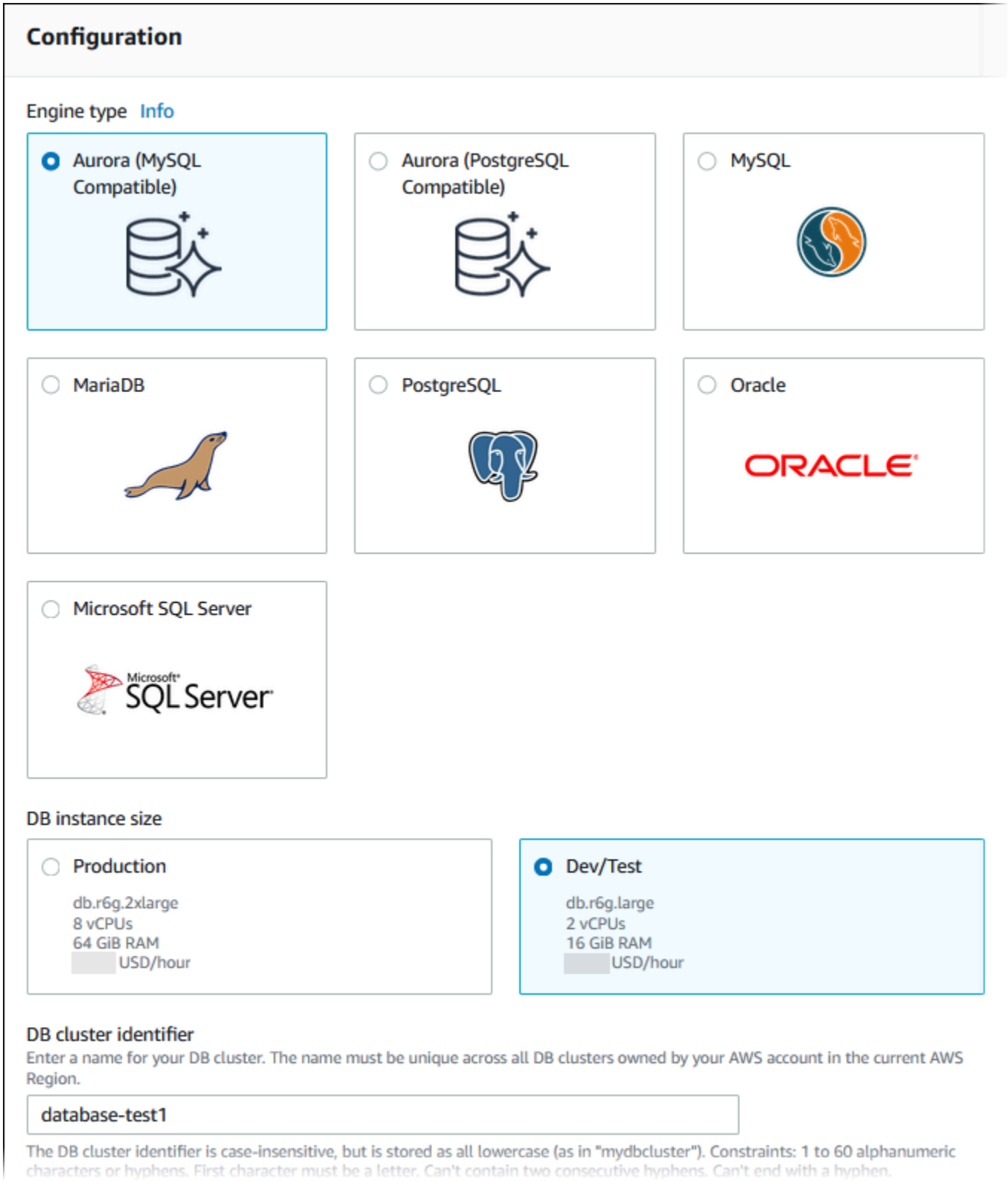
Untuk membuat klaster DB Aurora MySQL dengan Pembuatan mudah

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS di mana Anda ingin membuat cluster DB.
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data dan pastikan bahwa Pembuatan mudah dipilih.




5. Di Konfigurasi, pilih Aurora (Kompatibel dengan MySQL) untuk Jenis mesin.
6. Untuk Ukuran instans DB, pilih Dev/Tes.
7. Untuk pengidentifikasi klaster DB, masukkan **`database-test1`**.

   Tampilan halaman Membuat basis data seperti gambar berikut.



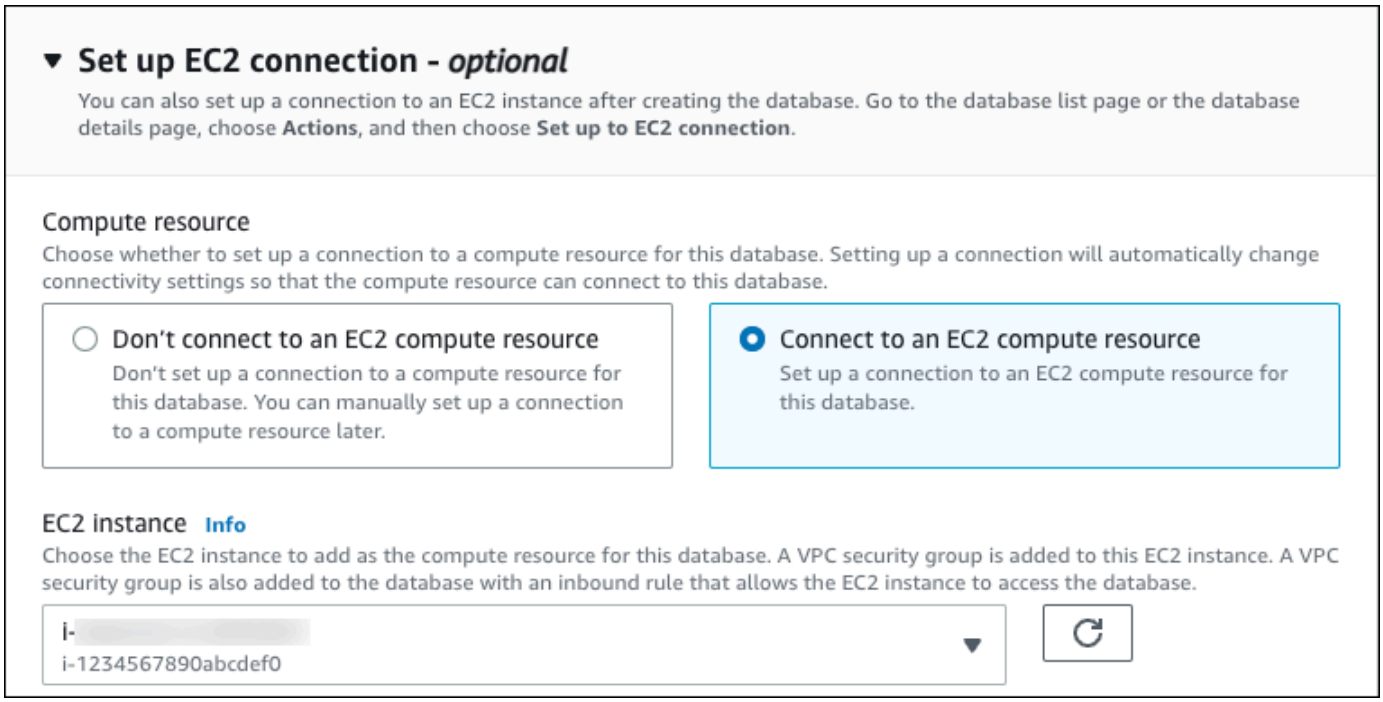

8. Untuk Nama pengguna utama, masukkan nama untuk pengguna utama, atau tetap gunakan nama default.

9. Untuk menggunakan kata sandi utama yang dibuat secara otomatis untuk klaster DB, pilih Buat kata sandi secara otomatis.

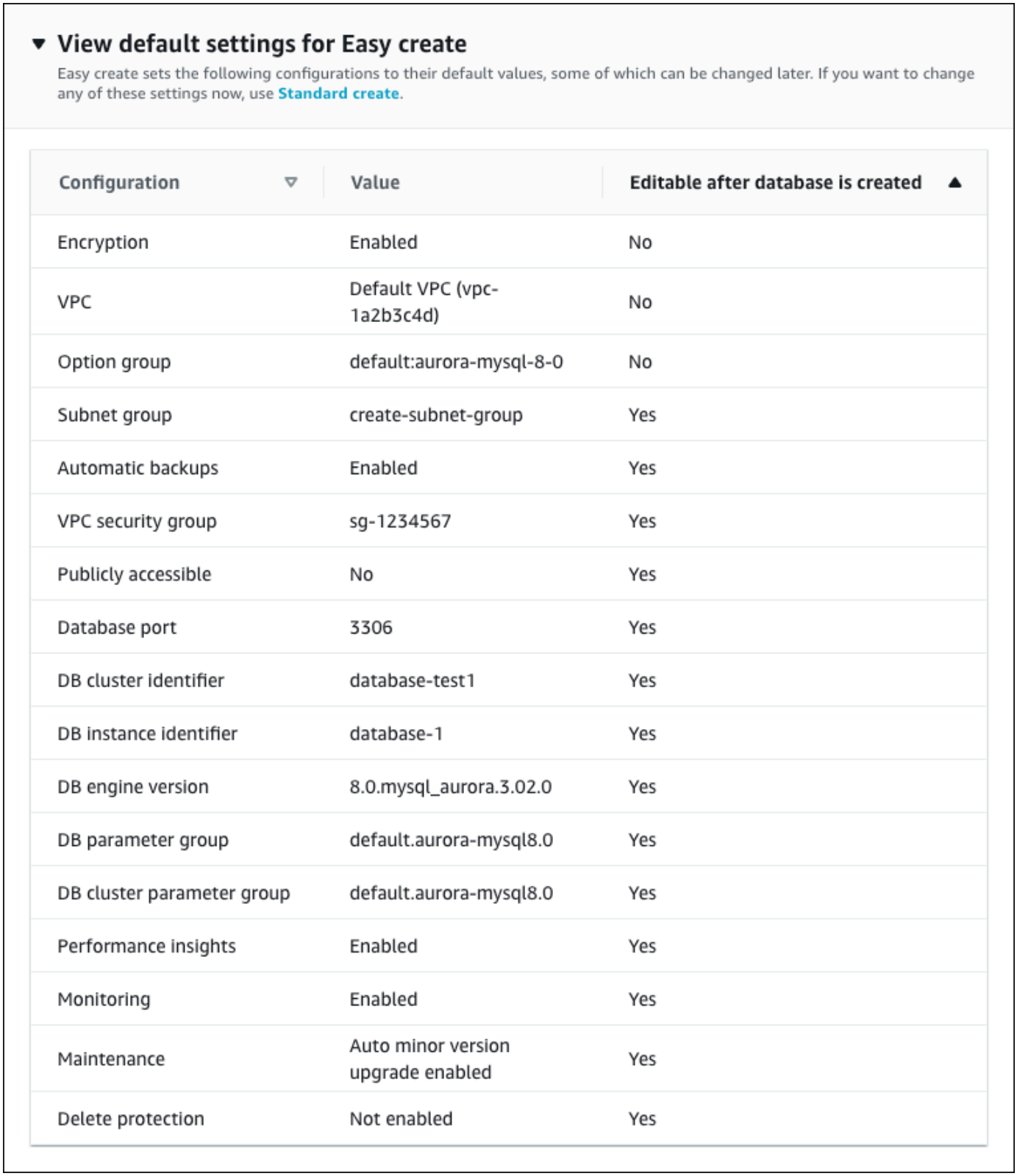
   Untuk memasukkan kata sandi utama Anda, hapus centang pada Buat kata sandi secara otomatis, lalu masukkan kata sandi yang sama dalam Kata sandi utama dan Konfirmasi kata sandi.

10. Untuk mengatur koneksi dengan EC2 instance yang Anda buat sebelumnya, buka Siapkan EC2 koneksi - opsional.

    Pilih Connect ke sumber daya EC2 komputasi. Pilih EC2 contoh yang Anda buat sebelumnya.




11. Buka Lihat pengaturan default untuk Pembuatan mudah.



**View default settings for Easy create**

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

| Configuration | Value | Editable after database is created |
|---|---|---|
| Encryption | Enabled | No |
| VPC | Default VPC (vpc-1a2b3c4d) | No |
| Option group | default:aurora-mysql-8-0 | No |
| Subnet group | create-subnet-group | Yes |
| Automatic backups | Enabled | Yes |
| VPC security group | sg-1234567 | Yes |
| Publicly accessible | No | Yes |
| Database port | 3306 | Yes |
| DB cluster identifier | database-test1 | Yes |
| DB instance identifier | database-1 | Yes |
| DB engine version | 8.0.mysql_aurora.3.02.0 | Yes |
| DB parameter group | default.aurora-mysql8.0 | Yes |
| DB cluster parameter group | default.aurora-mysql8.0 | Yes |
| Performance insights | Enabled | Yes |
| Monitoring | Enabled | Yes |
| Maintenance | Auto minor version upgrade enabled | Yes |
| Delete protection | Not enabled | Yes |

Anda dapat memeriksa pengaturan default yang digunakan dengan Pembuatan mudah. Kolom Dapat diedit setelah basis data dibuat menunjukkan opsi yang dapat Anda ubah setelah membuat basis data.

- Jika pengaturan memiliki Tidak di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB.
- Jika pengaturan memiliki Ya di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB, atau mengubah klaster DB setelah Anda membuatnya untuk mengubah pengaturan.

12. Pilih Buat basis data.

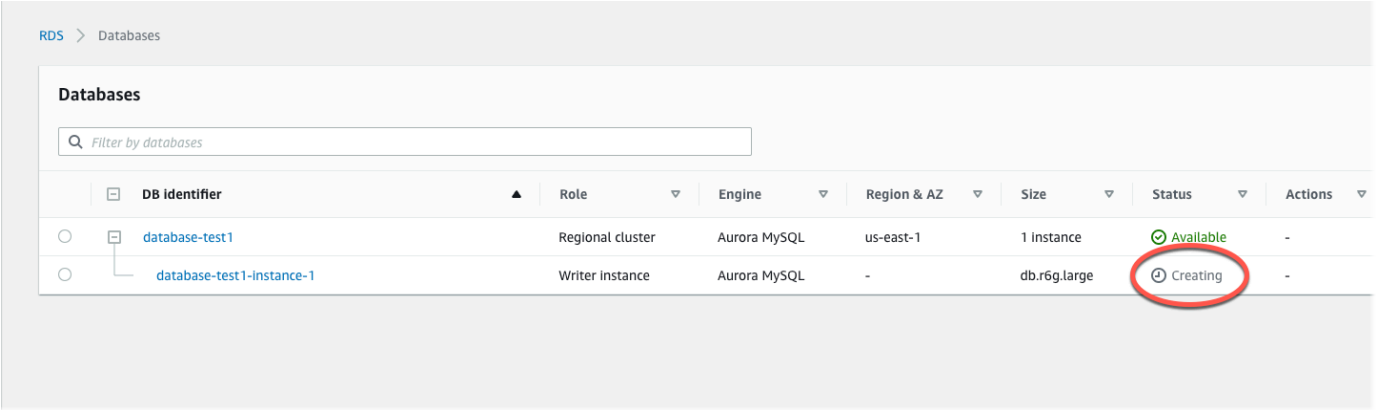
    Untuk melihat nama pengguna dan kata sandi utama untuk klaster DB, pilih Lihat detail kredensial.

    Anda dapat menggunakan nama pengguna dan kata sandi yang ditampilkan untuk terhubung ke klaster DB sebagai pengguna utama.

    > **Important**
    >
    > Anda tidak dapat melihat kata sandi pengguna utama lagi. Jika Anda tidak mencatatnya, Anda mungkin harus mengubahnya.
    > Jika perlu mengubah kata sandi pengguna utama setelah klaster DB tersedia, Anda dapat mengubah klaster DB untuk melakukannya. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

13. Dalam daftar Basis data, pilih nama klaster DB Aurora MySQL baru untuk menampilkan detailnya.

    Instans penulis memiliki status Membuat hingga klaster DB siap digunakan.




Saat status instans penulis berubah menjadi Tersedia, Anda dapat terhubung ke klaster DB. Tergantung pada kelas instans DB dan jumlah penyimpanan, diperlukan waktu hingga 20 menit sebelum klaster DB baru tersedia.

## (Opsional) Buat VPC, EC2 instance, dan cluster MySQL Aurora menggunakan AWS CloudFormation

Alih-alih menggunakan konsol untuk membuat VPC, EC2 instance, dan cluster DB MySQL Aurora, Anda dapat menggunakannya AWS CloudFormation untuk menyediakan sumber daya dengan memperlakukan infrastruktur sebagai kode. AWS Untuk membantu Anda mengatur AWS sumber daya Anda menjadi unit yang lebih kecil dan lebih mudah dikelola, Anda dapat menggunakan fungsionalitas tumpukan AWS CloudFormation bersarang. Untuk informasi selengkapnya, lihat Membuat tumpukan di AWS CloudFormation konsol dan Bekerja dengan tumpukan bersarang.

> **Important**
>
> AWS CloudFormation gratis, tetapi sumber daya yang CloudFormation menciptakan hidup. Anda dikenakan biaya penggunaan standar untuk sumber daya ini sampai Anda menghentikannya. Untuk informasi selengkapnya, lihat Harga Amazon Aurora.

Untuk membuat sumber daya Anda menggunakan AWS CloudFormation konsol, selesaikan langkah-langkah berikut:

- Langkah 1: Unduh CloudFormation template
- Langkah 2: Konfigurasikan sumber daya Anda menggunakan CloudFormation

## Unduh CloudFormation template

CloudFormation Template adalah file teks JSON atau YAMG yang berisi informasi konfigurasi tentang sumber daya yang ingin Anda buat di tumpukan. Template ini juga membuat VPC dan host benteng untuk Anda bersama dengan cluster Aurora.

Untuk mengunduh file template, buka tautan berikut, template Aurora MySQL. CloudFormation

Di halaman Github, klik tombol Unduh file mentah untuk menyimpan file YAMAL template.

## Konfigurasikan sumber daya Anda menggunakan CloudFormation

> **Note**
>
> Sebelum memulai proses ini, pastikan Anda memiliki pasangan Kunci untuk sebuah EC2 instance di Akun AWS. Untuk informasi selengkapnya, lihat pasangan EC2 kunci Amazon dan instans Linux.

Ketika Anda menggunakan AWS CloudFormation template, Anda harus memilih parameter yang benar untuk memastikan sumber daya Anda dibuat dengan benar. Ikuti langkah-langkah di bawah ini:

1. Masuk ke AWS Management Console dan buka AWS CloudFormation konsol di https://console.aws.amazon.com/cloudformation.
2. Pilih Buat tumpukan.
3. Di bagian Tentukan templat, pilih Unggah file templat dari komputer Anda, lalu pilih Berikutnya.
4. Di halaman Tentukan detail tumpukan, atur parameter berikut:
   a. Tetapkan nama Stack ke AurMySQLTestStack.
   b. Di bawah Parameter, atur Availability Zone dengan memilih dua zona ketersediaan.
   c. Di bawah konfigurasi Linux Bastion Host, untuk Key Name, pilih key pair untuk login ke EC2 instance Anda.
   d. Dalam pengaturan konfigurasi Linux Bastion Host, atur rentang IP yang Diizinkan ke alamat IP Anda. Untuk menyambung ke EC2 instans di VPC Anda menggunakan Secure Shell (SSH), tentukan alamat IP publik Anda menggunakan layanan di. https://checkip.amazonaws.com Contoh alamat IP adalah 192.0.2.1/32.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses EC2 instans publik Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, otorisasi hanya alamat IP tertentu atau rentang alamat untuk mengakses EC2 instans Anda menggunakan SSH.

e. Di bawah konfigurasi Database General, atur kelas instance Database ke db.r6g.large.

f. Tetapkan nama Database ke**`database-test1`**.

g. Untuk nama pengguna master Database, masukkan nama untuk pengguna master.

h. Atur Kelola kata sandi pengguna master DB dengan Secrets Manager `false` untuk tutorial ini.

i. Untuk kata sandi Database, tetapkan kata sandi pilihan Anda. Ingat kata sandi ini untuk langkah lebih lanjut dalam tutorial.

j. Setel penerapan Multi-AZ ke. `false`

k. Biarkan semua pengaturan lainnya sebagai nilai default. Klik Berikutnya untuk melanjutkan.

5. Di halaman Configure stack options, tinggalkan semua opsi default. Klik Berikutnya untuk melanjutkan.
6. Di halaman tumpukan Tinjauan, pilih Kirim setelah memeriksa database dan opsi host bastion Linux.

Setelah proses pembuatan tumpukan selesai, lihat tumpukan dengan nama BastionStackdan AMSNS untuk mencatat informasi yang Anda butuhkan untuk terhubung ke database. Untuk informasi selengkapnya, lihat Melihat data AWS CloudFormation tumpukan dan sumber daya di AWS Management Console.

## Langkah 3: Membuat klaster DB Aurora MySQL

Anda dapat menggunakan aplikasi klien SQL standar untuk menghubungkan ke klaster DB. Dalam contoh ini, Anda terhubung ke klaster DB Aurora MySQL menggunakan klien baris perintah mysql.

Untuk menghubungkan ke klaster DB Aurora MySQL

1. Temukan titik akhir (nama DNS) dan nomor port untuk instans penulis klaster DB Anda.

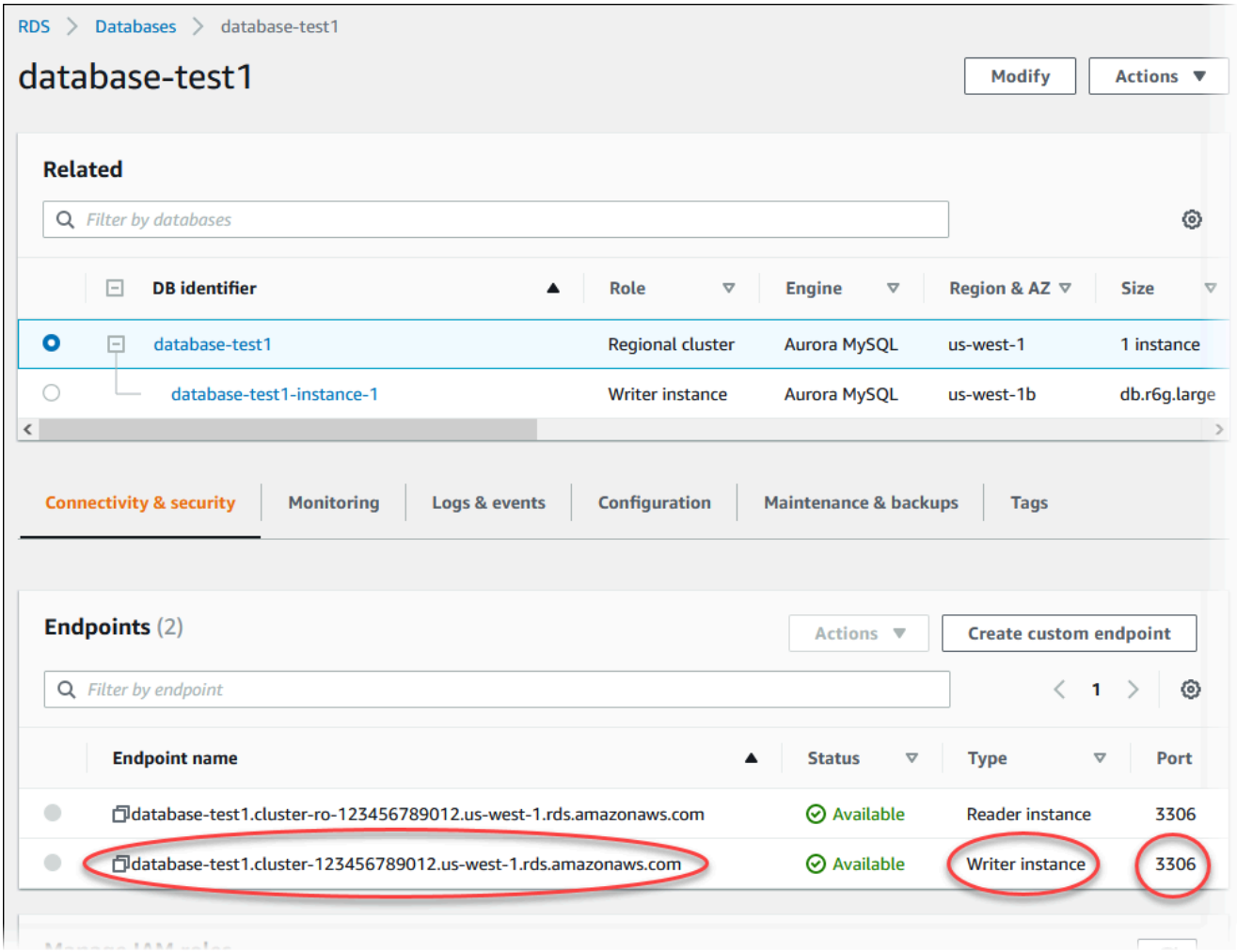
a. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

b. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS untuk klaster DB.

c. Di panel navigasi, pilih Basis data.

d. Pilih nama klaster DB Aurora MySQL untuk menampilkan detailnya.

e. Di tab Konektivitas & keamanan, salin titik akhir instans penulis. Perhatikan juga nomor port. Anda memerlukan titik akhir dan nomor port untuk terhubung ke klaster DB.




2. Hubungkan ke EC2 instans yang Anda buat sebelumnya dengan mengikuti langkah-langkah di Connect to Linux Anda di Panduan EC2 Pengguna Amazon.

   Kami menyarankan Anda terhubung ke EC2 instans Anda menggunakan SSH. Jika utilitas klien SSH diinstal di Windows, Linux, atau Mac, Anda dapat menghubungkan ke instans menggunakan format perintah berikut:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

Misalnya, anggaplah `ec2-database-connect-key-pair.pem` itu disimpan `/dir1` di Linux, dan IPv4 DNS publik untuk EC2 instance Anda adalah`ec2-12-345-678-90.compute-1.amazonaws.com`. Perintah SSH Anda akan tampak seperti berikut:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Dapatkan perbaikan bug terbaru dan pembaruan keamanan dengan memperbarui perangkat lunak pada EC2 instans Anda. Untuk melakukannya, gunakan perintah berikut.

> **Note**
>
> Opsi `-y` menginstal pembaruan tanpa meminta konfirmasi. Hilangkan opsi ini untuk memeriksa pembaruan sebelum menginstal.

```
sudo dnf update -y
```

4. Untuk menginstal klien baris perintah mysql dari MariaDB di Amazon Linux 2023, jalankan perintah berikut:

```
sudo dnf install mariadb105
```

5. Hubungkan ke klaster DB Aurora MySQL. Misalnya, masukkan perintah berikut. Tindakan ini memungkinkan Anda terhubung ke klaster DB Aurora MySQL menggunakan klien MySQL.

   Ganti titik akhir instans penulis dengan *`endpoint`*, dan ganti nama pengguna utama yang Anda gunakan dengan *`admin`*. Masukkan kata sandi utama yang Anda gunakan saat dimintai kata sandi.

```
mysql -h endpoint -P 3306 -u admin -p
```

   Setelah Anda memasukkan kata sandi untuk pengguna, Anda akan melihat output yang serupa dengan yang berikut ini.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 217
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Untuk informasi lebih lanjut tentang menghubungkan ke klaster DB Aurora MySQL, lihat Menghubungkan ke klaster DB Amazon Aurora MySQL. Jika Anda tidak dapat terhubung ke klaster DB Anda, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Untuk keamanan, ini adalah praktik terbaik untuk menggunakan koneksi terenkripsi. Hanya gunakan koneksi MySQL yang tidak terenkripsi saat klien dan server berada di VPC yang sama dan jaringan tepercaya. Untuk informasi tentang cara menggunakan koneksi terenkripsi, lihat Menghubungkan ke Aurora MySQL menggunakan SSL.

6. Jalankan perintah SQL.

   Misalnya, perintah SQL berikut menunjukkan tanggal dan waktu saat ini:

   ```
   SELECT CURRENT_TIMESTAMP;
   ```

## Langkah 4: Hapus EC2 instance dan cluster DB

Setelah Anda terhubung ke dan menjelajahi EC2 contoh contoh dan cluster DB yang Anda buat, hapus mereka sehingga Anda tidak lagi dikenakan biaya untuk mereka.

Jika Anda biasa AWS CloudFormation membuat sumber daya, lewati langkah ini dan lanjutkan ke langkah berikutnya.

Untuk menghapus EC2 instance

1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Di panel navigasi, pilih Instans.
3. Pilih EC2 instance, dan pilih Instance state, Terminate instance.

4. Pilih Akhiri saat diminta untuk konfirmasi.

Untuk informasi selengkapnya tentang menghapus EC2 instance, lihat Mengakhiri instance Anda di EC2 Panduan Pengguna Amazon.

Untuk menghapus klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data lalu pilih instans DB yang terkait dengan klaster DB.
3. Untuk Tindakan, pilih Hapus.
4. Hapus Buat snapshot akhir?.
5. Lengkapi pengakuan dan pilih Hapus.

Setelah semua instans DB yang terkait dengan klaster DB dihapus, klaster DB akan dihapus secara otomatis.

## (Opsional) Hapus EC2 instance dan cluster DB yang dibuat dengan CloudFormation

Jika Anda biasa AWS CloudFormation membuat sumber daya, hapus CloudFormation tumpukan setelah Anda terhubung dan jelajahi EC2 contoh contoh dan kluster DB, sehingga Anda tidak lagi dikenakan biaya untuk itu.

Untuk menghapus sumber CloudFormation daya

1. Buka AWS CloudFormation konsol.
2. Pada halaman Stacks di CloudFormation konsol, pilih tumpukan root (tumpukan tanpa nama VPCStack, BastionStack atau AMSNS).
3. Pilih Hapus.
4. Pilih Hapus tumpukan saat diminta konfirmasi.

Untuk informasi selengkapnya tentang menghapus tumpukan CloudFormation, lihat Menghapus tumpukan di AWS CloudFormation konsol di AWS CloudFormation Panduan Pengguna.

## (Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda

Anda juga dapat menghubungkan klaster DB Aurora MySQL ke sumber daya komputasi nirserver Lambda. Fungsi Lambda memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola infrastruktur. Fungsi Lambda juga memungkinkan Anda untuk otomatis merespons permintaan eksekusi kode pada skala apa pun, mulai dari belasan peristiwa dalam sehari hingga ratusan per detik. Lihat informasi yang lebih lengkap di Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora.

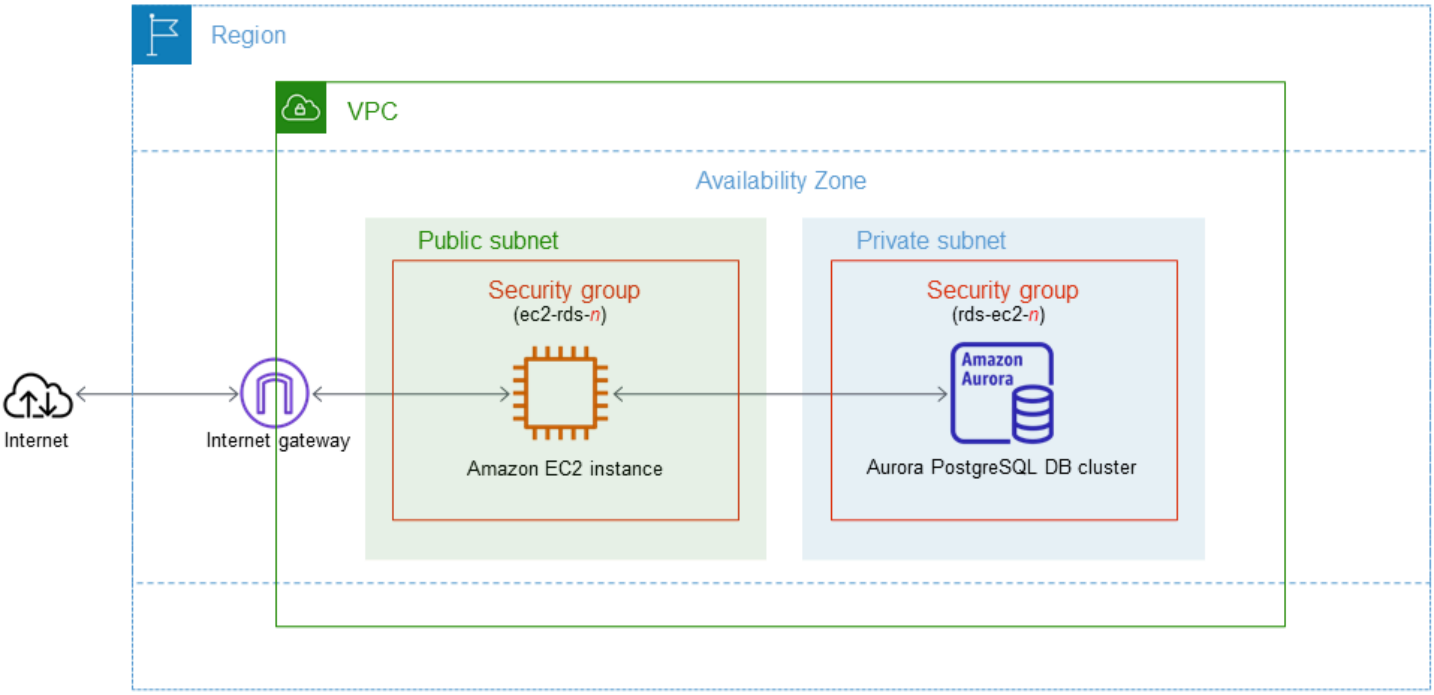
# Membuat dan terhubung ke klaster DB Aurora PostgreSQL

Tutorial ini membuat EC2 instance dan cluster Aurora PostgreSQL DB. Tutorial menunjukkan cara mengakses cluster DB dari EC2 instance menggunakan klien PostgreSQL standar. Sebagai praktik terbaik, tutorial ini membuat klaster DB privat dalam cloud privat virtual (VPC). Dalam kebanyakan kasus, sumber daya lain dalam VPC yang sama, seperti EC2 instance, dapat mengakses cluster DB, tetapi sumber daya di luar VPC tidak dapat mengaksesnya.

Setelah Anda menyelesaikan tutorial, ada subnet publik dan privat di setiap Zona Ketersediaan di VPC Anda. Dalam satu Availability Zone, EC2 instance ada di subnet publik, dan instance DB ada di subnet pribadi.

> **Important**
>
> Tidak ada biaya untuk membuat AWS akun. Namun, dengan menyelesaikan tutorial ini, Anda mungkin dikenakan biaya untuk AWS sumber daya yang Anda gunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

Diagram berikut menunjukkan konfigurasi setelah tutorial selesai.




Tutorial ini memungkinkan Anda untuk membuat sumber daya Anda dengan menggunakan salah satu metode berikut:

1. Gunakan AWS Management Console - Langkah 1: Buat sebuah EC2 instance dan Langkah 2: Membuat klaster DB Aurora PostgreSQL
2. Gunakan AWS CloudFormation untuk membuat instance database dan EC2 instance - (Opsional) Buat VPC, EC2 instance, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation

Metode pertama menggunakan Easy create untuk membuat cluster Aurora PostgreSQL DB pribadi dengan. AWS Management Console Di sini, Anda hanya menentukan jenis mesin DB, ukuran instans DB, dan pengidentifikasi cluster DB. Pembuatan mudah menggunakan pengaturan default untuk opsi konfigurasi lainnya.

Saat Anda menggunakan Standard create sebagai gantinya, Anda dapat menentukan lebih banyak opsi konfigurasi saat membuat cluster DB. Opsi ini mencakup pengaturan untuk ketersediaan, keamanan, cadangan, dan pemeliharaan. Untuk membuat klaster DB publik, Anda harus menggunakan Pembuatan standar. Untuk informasi, lihat the section called “Membuat klaster DB”.

Topik

- Prasyarat
- Langkah 1: Buat sebuah EC2 instance

- Langkah 2: Membuat klaster DB Aurora PostgreSQL
- (Opsional) Buat VPC, EC2 instance, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation
- Langkah 3: Membuat klaster DB Aurora PostgreSQL
- Langkah 4: Hapus EC2 instance dan cluster DB
- (Opsional) Hapus EC2 instance dan cluster DB yang dibuat dengan CloudFormation
- (Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda


## Prasyarat

Sebelum memulai, selesaikan langkah-langkah di bagian berikut:

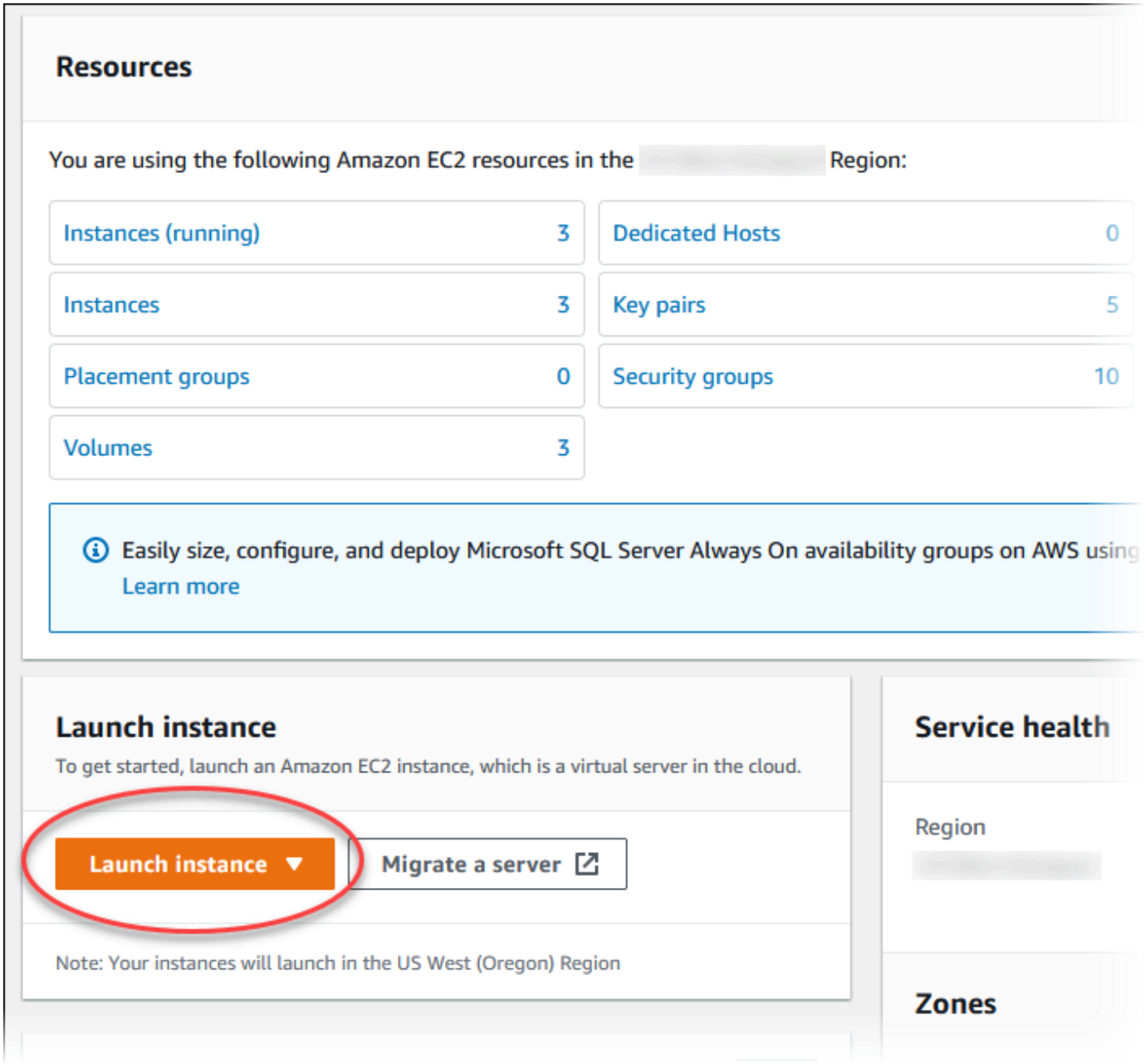
- Mendaftar untuk Akun AWS
- Buat pengguna dengan akses administratif

## Langkah 1: Buat sebuah EC2 instance

Buat EC2 instance Amazon yang akan Anda gunakan untuk terhubung ke database Anda.

Untuk membuat sebuah EC2 instance

1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda ingin membuat instance. EC2
3. Pilih EC2 Dasbor, lalu pilih Launch instance, seperti yang ditunjukkan pada gambar berikut.

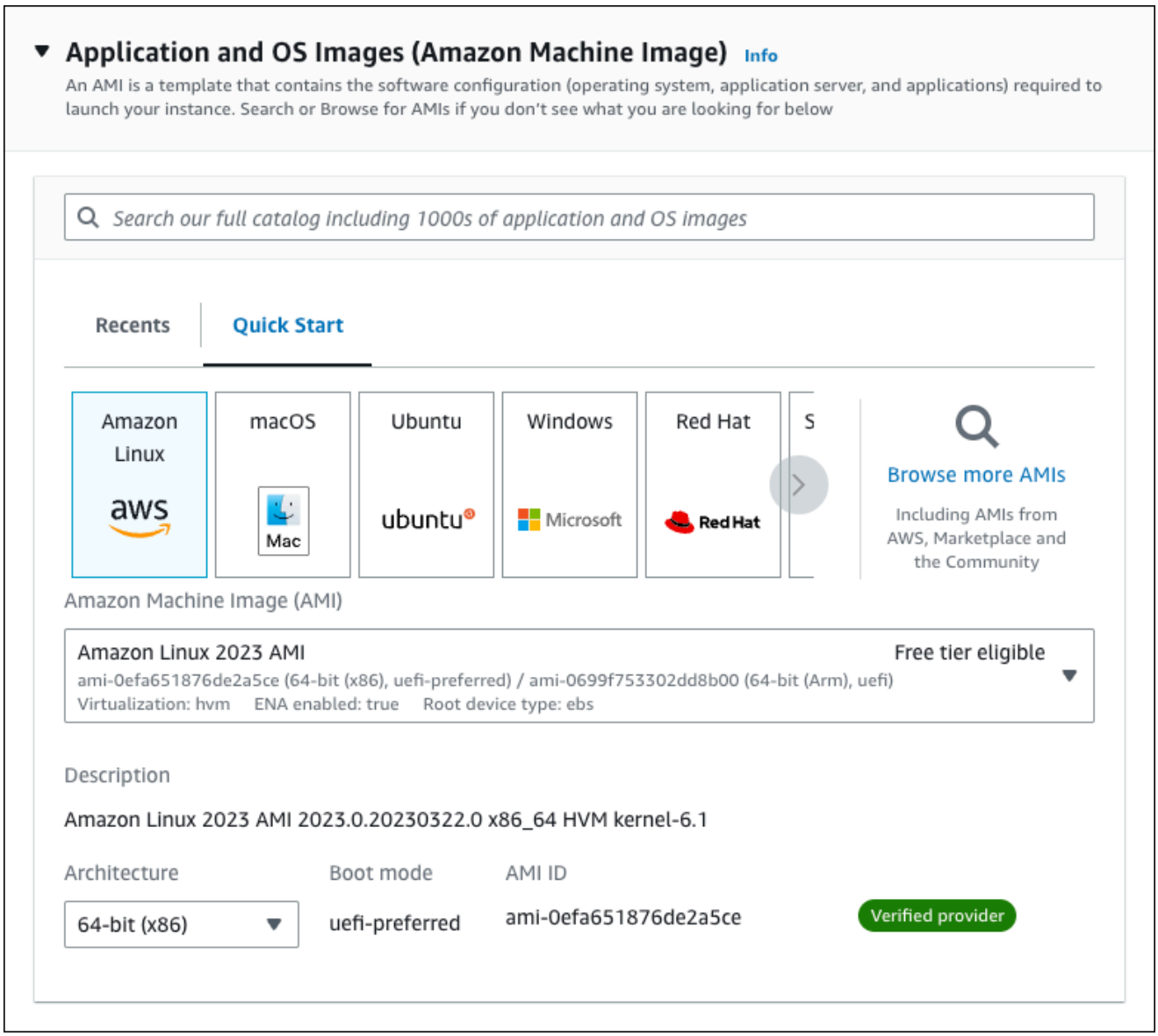



Halaman Meluncurkan instans akan terbuka.

4. Pilih pengaturan berikut di halaman Meluncurkan instans.

   a. Pada Nama dan tag, untuk Nama, masukkan **`ec2-database-connect`**.

   b. Pada Gambar Aplikasi dan OS (Amazon Machine Image), pilih Amazon Linux, lalu pilih AMI Amazon Linux 2023. Biarkan default untuk pilihan lainnya.




c. Pada Jenis instans, pilih t2.micro.

d. Pada Pasangan kunci (login), pilih Nama pasangan kunci untuk menggunakan pasangan kunci yang ada. Untuk membuat key pair baru untuk EC2 instance Amazon, pilih Create new key pair dan kemudian gunakan jendela Create key pair untuk membuatnya.

   Untuk informasi selengkapnya tentang membuat key pair baru, lihat Membuat key pair di Panduan EC2 Pengguna Amazon.

e. Untuk Izinkan lalu lintas SSH di pengaturan Jaringan, pilih sumber koneksi SSH ke instance. EC2

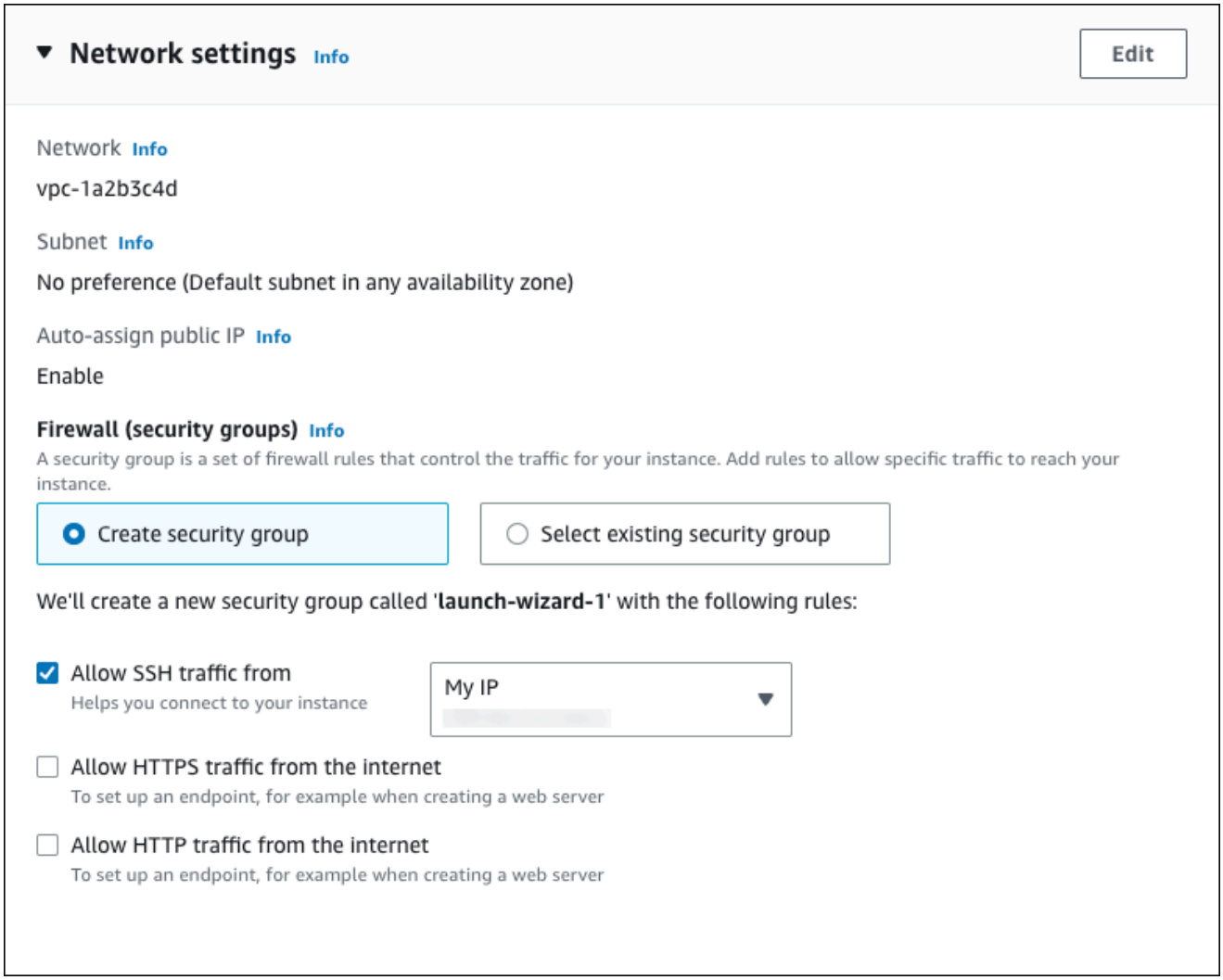
   Anda dapat memilih IP Saya jika alamat IP yang ditampilkan benar untuk koneksi SSH. Jika tidak, Anda dapat menentukan alamat IP yang akan digunakan untuk terhubung ke EC2

instance di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, di jendela atau tab browser yang berbeda, Anda dapat menggunakan layanan di https://checkip.amazonaws.com. Contoh alamat IP adalah 192.0.2.1/32.

Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, tentukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses EC2 instans publik Anda menggunakan SSH. Pendekatan ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, otorisasi hanya alamat IP tertentu atau rentang alamat untuk mengakses EC2 instans Anda menggunakan SSH.

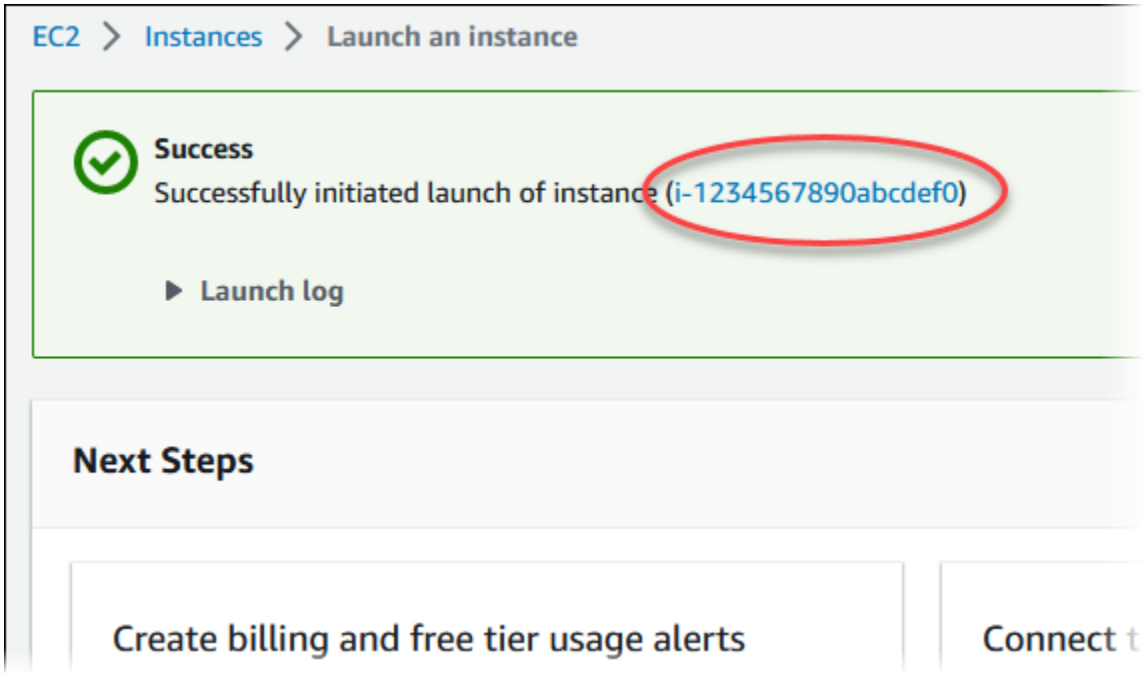
Gambar berikut menunjukkan contoh bagian Pengaturan jaringan.




f. Biarkan nilai default untuk bagian yang lainnya.

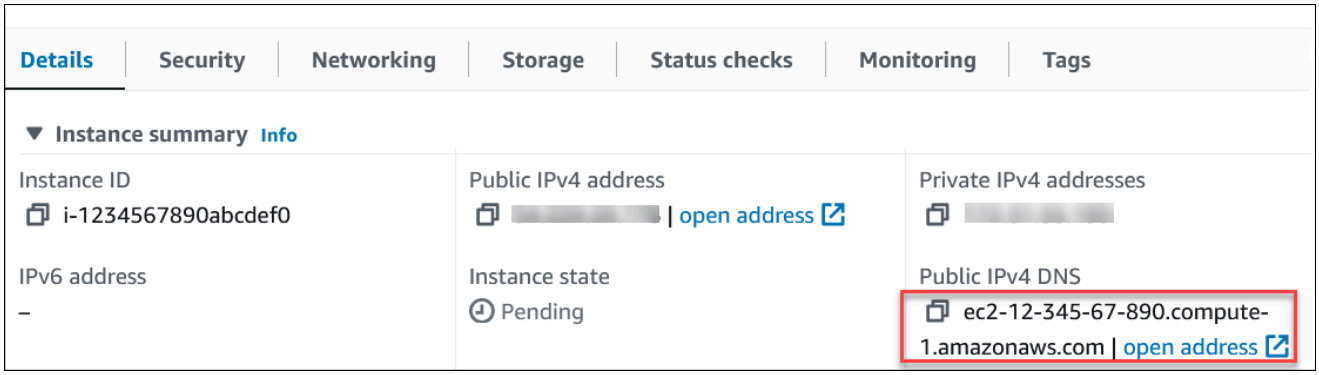
g. Tinjau ringkasan konfigurasi EC2 instans Anda di panel Summary, dan ketika Anda siap, pilih Launch instance.

5. Pada halaman Status Peluncuran, perhatikan pengenal untuk EC2 instance baru Anda, misalnya:`i-1234567890abcdef0`.

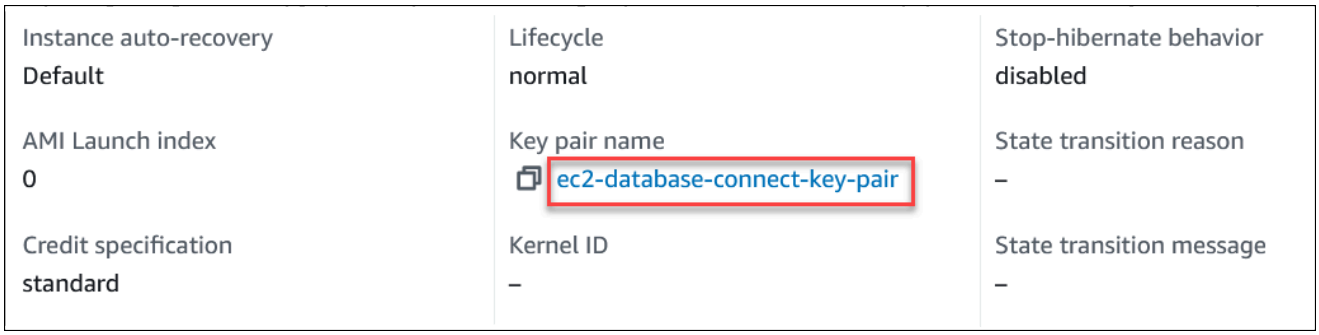



6. Pilih pengenal EC2 instance untuk membuka daftar EC2 instance, lalu pilih instance Anda EC2 .
7. Di tab Detail, catat nilai-nilai berikut, yang akan Anda butuhkan saat menghubungkan menggunakan SSH:

   a. Dalam ringkasan Instance, perhatikan nilai untuk  IPv4 DNS Publik.




   b. Di Detail instans, catat nilai untuk Nama pasangan kunci.



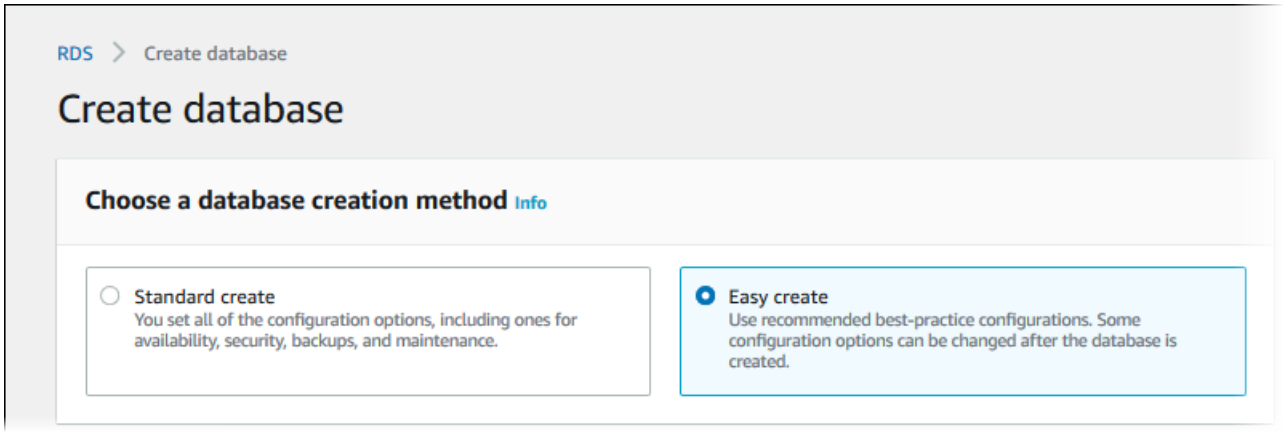
8. Tunggu hingga status Instance untuk EC2 instans Anda memiliki status Running sebelum melanjutkan.

## Langkah 2: Membuat klaster DB Aurora PostgreSQL

Dalam contoh ini, Anda menggunakan Pembuatan mudah untuk membuat klaster DB Aurora PostgreSQL dengan kelas instans DB db.t4g.large.

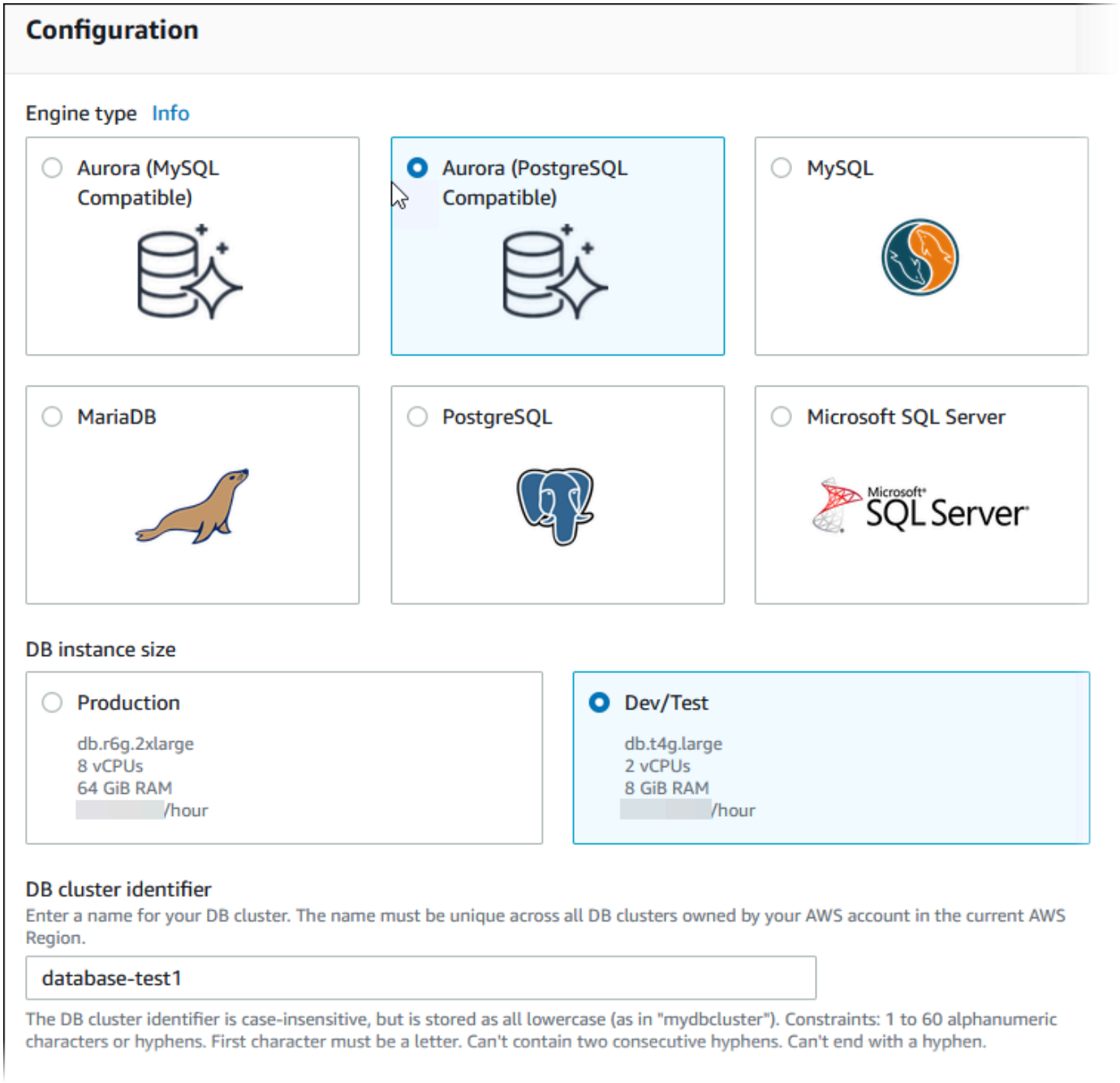
Untuk membuat klaster DB Aurora PostgreSQL dengan Pembuatan mudah

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS tempat Anda akan membuat klaster DB.
3. Di panel navigasi, pilih Basis data.
4. Pilih Buat basis data dan pastikan bahwa Pembuatan mudah dipilih.




5. Di Konfigurasi, pilih Aurora (Kompatibel dengan PostgreSQL) untuk Jenis mesin.
6. Untuk Ukuran instans DB, pilih Dev/Tes.
7. Untuk pengidentifikasi klaster DB, masukkan **`database-test1`**.

   Tampilan halaman Membuat basis data seperti gambar berikut.

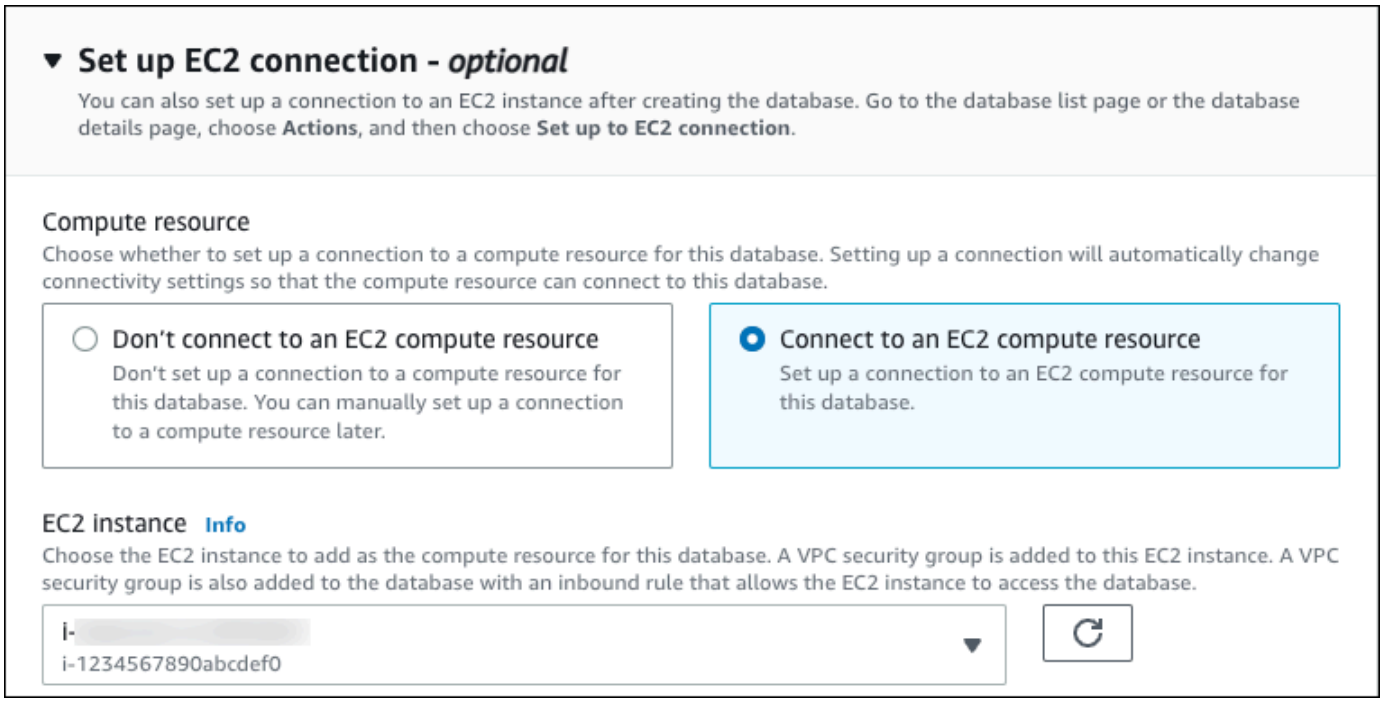



8. Untuk Nama pengguna utama, masukkan nama untuk pengguna, atau tetap gunakan nama default (**`postgres`**).
9. Untuk menggunakan kata sandi utama yang dibuat secara otomatis untuk klaster DB, pilih Buat kata sandi secara otomatis.

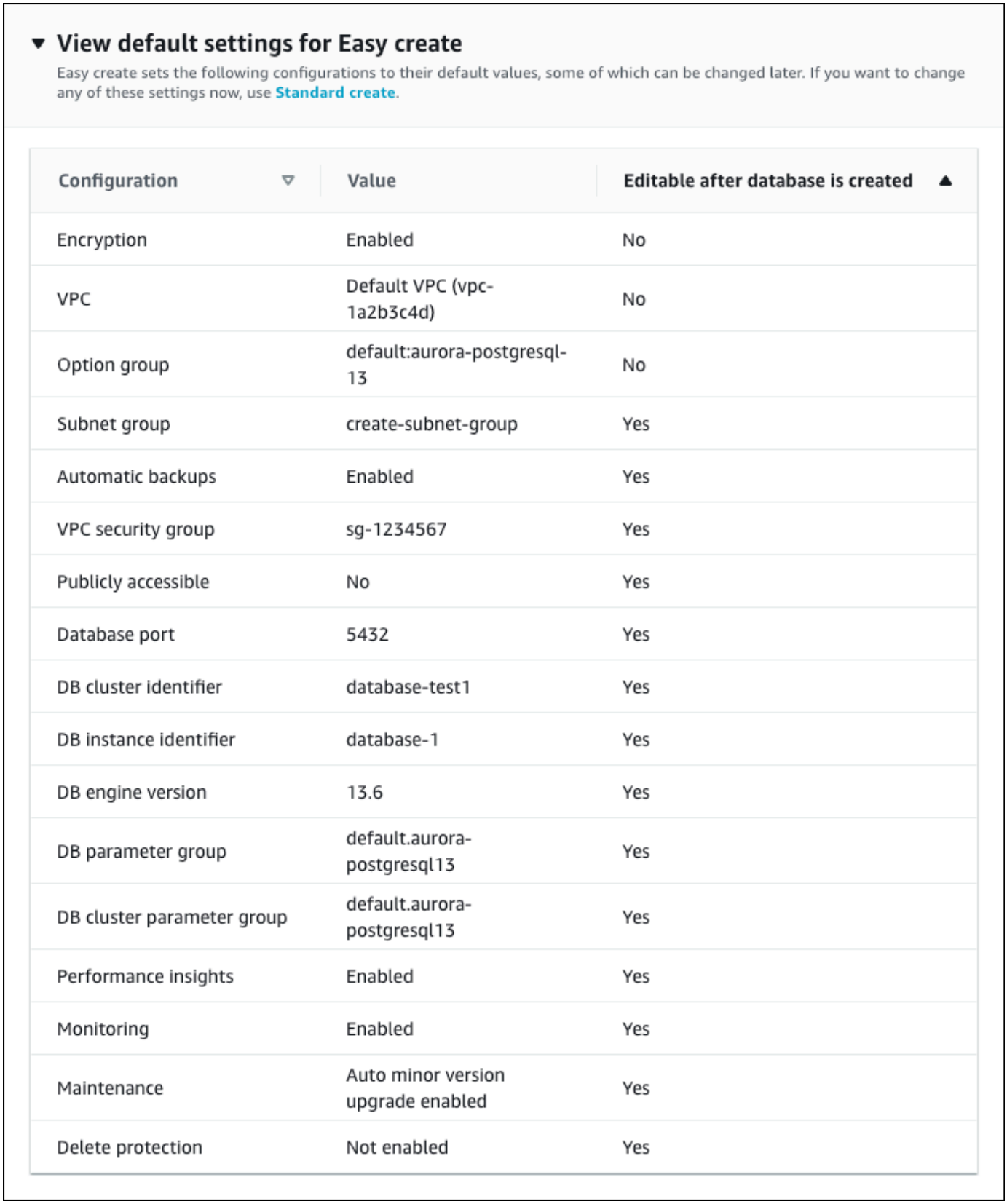
   Untuk memasukkan kata sandi utama Anda, hapus centang pada Buat kata sandi secara otomatis, lalu masukkan kata sandi yang sama dalam Kata sandi utama dan Konfirmasi kata sandi.

10. Untuk mengatur koneksi dengan EC2 instance yang Anda buat sebelumnya, buka Siapkan EC2 koneksi - opsional.

    Pilih Connect ke sumber daya EC2 komputasi. Pilih EC2 contoh yang Anda buat sebelumnya.




11. Buka Lihat pengaturan default untuk Pembuatan mudah.



▼ View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use Standard create.

| Configuration | Value | Editable after database is created |
|---|---|---|
| Encryption | Enabled | No |
| VPC | Default VPC (vpc-1a2b3c4d) | No |
| Option group | default:aurora-postgresql-13 | No |
| Subnet group | create-subnet-group | Yes |
| Automatic backups | Enabled | Yes |
| VPC security group | sg-1234567 | Yes |
| Publicly accessible | No | Yes |
| Database port | 5432 | Yes |
| DB cluster identifier | database-test1 | Yes |
| DB instance identifier | database-1 | Yes |
| DB engine version | 13.6 | Yes |
| DB parameter group | default.aurora-postgresql13 | Yes |
| DB cluster parameter group | default.aurora-postgresql13 | Yes |
| Performance insights | Enabled | Yes |
| Monitoring | Enabled | Yes |
| Maintenance | Auto minor version upgrade enabled | Yes |
| Delete protection | Not enabled | Yes |

Anda dapat memeriksa pengaturan default yang digunakan dengan Pembuatan mudah. Kolom Dapat diedit setelah basis data dibuat menunjukkan opsi yang dapat Anda ubah setelah membuat basis data.

- Jika pengaturan memiliki Tidak di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB.
- Jika pengaturan memiliki Ya di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB, atau mengubah klaster DB setelah Anda membuatnya untuk mengubah pengaturan.

12. Pilih Buat basis data.

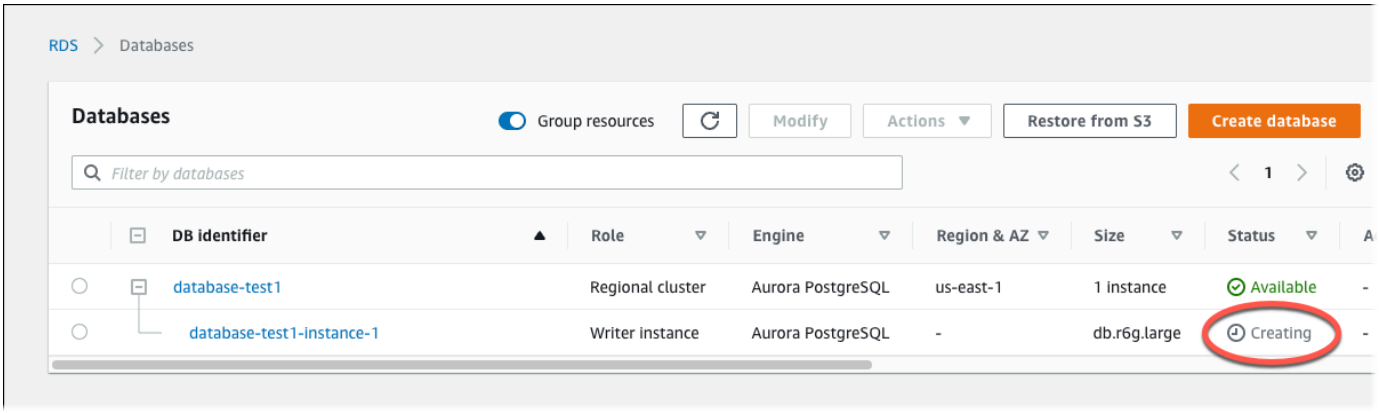
    Untuk melihat nama pengguna dan kata sandi utama untuk klaster DB, pilih Lihat detail kredensial.

    Anda dapat menggunakan nama pengguna dan kata sandi yang ditampilkan untuk terhubung ke klaster DB sebagai pengguna utama.

    > **Important**
    >
    > Anda tidak dapat melihat kata sandi pengguna utama lagi. Jika Anda tidak mencatatnya, Anda mungkin harus mengubahnya.
    > Jika perlu mengubah kata sandi pengguna utama setelah klaster DB tersedia, Anda dapat mengubah klaster DB untuk melakukannya. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

13. Dalam daftar Basis data, pilih nama klaster DB Aurora PostgreSQL baru untuk menampilkan detailnya.

    Instans penulis memiliki status Membuat hingga klaster DB siap digunakan.




Saat status instans penulis berubah menjadi Tersedia, Anda dapat terhubung ke klaster DB. Tergantung pada kelas instans DB dan jumlah penyimpanan, diperlukan waktu hingga 20 menit sebelum klaster DB baru tersedia.

## (Opsional) Buat VPC, EC2 instance, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation

Alih-alih menggunakan konsol untuk membuat klaster DB VPC, EC2 instance, dan Aurora PostgreSQL, Anda dapat menggunakannya untuk menyediakan sumber daya dengan memperlakukan infrastruktur sebagai kode. AWS CloudFormation AWS Untuk membantu Anda mengatur AWS sumber daya Anda menjadi unit yang lebih kecil dan lebih mudah dikelola, Anda dapat menggunakan fungsionalitas tumpukan AWS CloudFormation bersarang. Untuk informasi selengkapnya, lihat Membuat tumpukan di AWS CloudFormation konsol dan Bekerja dengan tumpukan bersarang.

> **Important**
>
> AWS CloudFormation gratis, tetapi sumber daya yang CloudFormation menciptakan hidup. Anda dikenakan biaya penggunaan standar untuk sumber daya ini sampai Anda menghentikannya. Untuk informasi selengkapnya, lihat Harga Amazon Aurora.

Untuk membuat sumber daya menggunakan AWS CloudFormation konsol, selesaikan langkah-langkah berikut:

- Langkah 1: Unduh CloudFormation template
- Langkah 2: Konfigurasikan sumber daya Anda menggunakan CloudFormation

## Unduh CloudFormation template

CloudFormation Template adalah file teks JSON atau YAMG yang berisi informasi konfigurasi tentang sumber daya yang ingin Anda buat di tumpukan. Template ini juga membuat VPC dan host benteng untuk Anda bersama dengan cluster Aurora.

Untuk mengunduh file template, buka tautan berikut, template PostgreSQL Aurora. CloudFormation

Di halaman Github, klik tombol Unduh file mentah untuk menyimpan file YAMAL template.

## Konfigurasikan sumber daya Anda menggunakan CloudFormation

> **Note**
>
> Sebelum memulai proses ini, pastikan Anda memiliki pasangan Kunci untuk sebuah EC2 instance di Anda Akun AWS. Untuk informasi selengkapnya, lihat pasangan EC2 kunci Amazon dan instans Linux.

Ketika Anda menggunakan AWS CloudFormation template, Anda harus memilih parameter yang benar untuk memastikan sumber daya Anda dibuat dengan benar. Ikuti langkah-langkah di bawah ini:

1. Masuk ke AWS Management Console dan buka AWS CloudFormation konsol di https://console.aws.amazon.com/cloudformation.
2. Pilih Buat tumpukan.
3. Di bagian Tentukan templat, pilih Unggah file templat dari komputer Anda, lalu pilih Berikutnya.
4. Di halaman Tentukan detail tumpukan, atur parameter berikut:
   a. Tetapkan nama Stack ke AurPostgreSQLTestStack.
   b. Di bawah Parameter, atur Availability Zone dengan memilih dua zona ketersediaan.
   c. Di bawah konfigurasi Linux Bastion Host, untuk Key Name, pilih key pair untuk login ke EC2 instance Anda.
   d. Dalam pengaturan konfigurasi Linux Bastion Host, atur rentang IP yang Diizinkan ke alamat IP Anda. Untuk menyambung ke EC2 instans di VPC Anda menggunakan Secure Shell (SSH), tentukan alamat IP publik Anda menggunakan layanan di. https://checkip.amazonaws.com Contoh alamat IP adalah 192.0.2.1/32.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses EC2 instans publik Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, otorisasi hanya alamat IP tertentu atau rentang alamat untuk mengakses EC2 instans Anda menggunakan SSH.

e. Di bawah konfigurasi Database General, atur kelas instance Database ke db.t4g.large.

f. Tetapkan nama Database ke**`database-test1`**.

g. Untuk nama pengguna master Database, masukkan nama untuk pengguna master.

h. Atur Kelola kata sandi pengguna master DB dengan Secrets Manager `false` untuk tutorial ini.

i. Untuk kata sandi Database, tetapkan kata sandi pilihan Anda. Ingat kata sandi ini untuk langkah lebih lanjut dalam tutorial.

j. Setel penerapan Multi-AZ ke. `false`

k. Biarkan semua pengaturan lainnya sebagai nilai default. Klik Berikutnya untuk melanjutkan.

5. Di halaman Configure stack options, tinggalkan semua opsi default. Klik Berikutnya untuk melanjutkan.
6. Di halaman tumpukan Tinjauan, pilih Kirim setelah memeriksa database dan opsi host bastion Linux.

Setelah proses pembuatan tumpukan selesai, lihat tumpukan dengan nama BastionStackdan APGNS untuk mencatat informasi yang Anda butuhkan untuk terhubung ke database. Untuk informasi selengkapnya, lihat Melihat data AWS CloudFormation tumpukan dan sumber daya di AWS Management Console.

## Langkah 3: Membuat klaster DB Aurora PostgreSQL

Anda dapat menggunakan aplikasi klien PostgreSQL standar untuk menghubungkan ke klaster DB. Dalam contoh ini, Anda terhubung ke klaster DB Aurora PostgreSQL menggunakan klien baris perintah psql.

Untuk menghubungkan ke klaster DB Aurora PostgreSQL

1. Temukan titik akhir (nama DNS) dan nomor port untuk instans penulis klaster DB Anda.

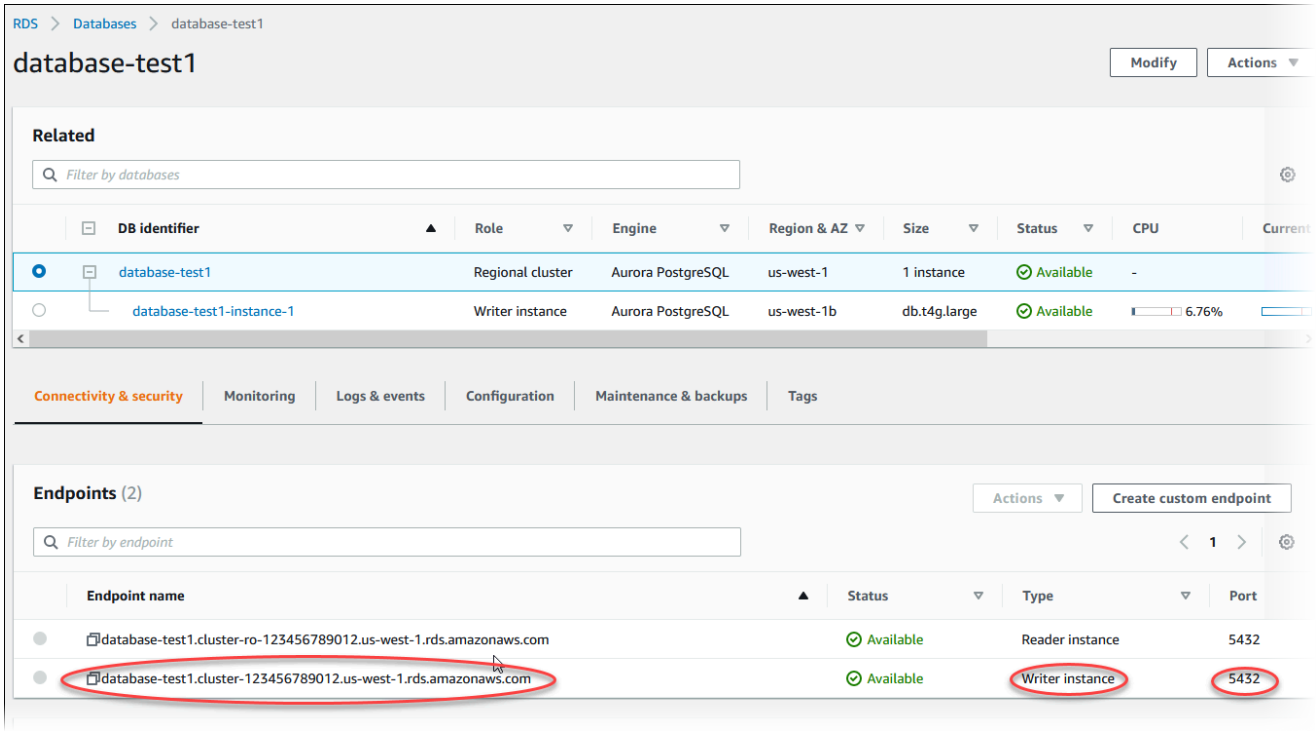
a. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

b. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS untuk cluster DB.

c. Di panel navigasi, pilih Basis data.

d. Pilih nama klaster DB Aurora PostgreSQL untuk menampilkan detailnya.

e. Di tab Konektivitas & keamanan, salin titik akhir instans penulis. Perhatikan juga nomor port. Anda memerlukan titik akhir dan nomor port untuk terhubung ke klaster DB.




2. Hubungkan ke EC2 instans yang Anda buat sebelumnya dengan mengikuti langkah-langkah di Connect to Linux Anda di Panduan EC2 Pengguna Amazon.

   Kami menyarankan Anda terhubung ke EC2 instans Anda menggunakan SSH. Jika utilitas klien SSH diinstal di Windows, Linux, atau Mac, Anda dapat menghubungkan ke instans menggunakan format perintah berikut:

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   Misalnya, asumsikan bahwa `ec2-database-connect-key-pair.pem` disimpan `/dir1` di Linux, dan IPv4 DNS publik untuk EC2 instance Anda

adalah`ec2-12-345-678-90.compute-1.amazonaws.com`. Perintah SSH Anda akan tampak seperti berikut:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Dapatkan perbaikan bug terbaru dan pembaruan keamanan dengan memperbarui perangkat lunak pada EC2 instans Anda. Untuk melakukannya, gunakan perintah berikut.

> **Note**
>
> Opsi `-y` menginstal pembaruan tanpa meminta konfirmasi. Hilangkan opsi ini untuk memeriksa pembaruan sebelum menginstal.

```
sudo dnf update -y
```

4. Instal klien baris perintah psql dari PostgreSQL di Amazon Linux 2023, menggunakan perintah berikut:

```
sudo dnf install postgresql15
```

5. Hubungkan ke klaster DB Aurora PostgreSQL. Misalnya, masukkan perintah berikut. Tindakan ini memungkinkan Anda terhubung ke klaster DB Aurora PostgreSQL menggunakan klien psql.

   Ganti titik akhir instans penulis untuk *`endpoint`*, ganti nama basis data `--dbname` yang ingin Anda hubungkan untuk *`postgres`*, dan ganti nama pengguna utama yang Anda gunakan untuk *`postgres`*. Masukkan kata sandi utama yang Anda gunakan saat dimintai kata sandi.

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

   Setelah Anda memasukkan kata sandi untuk pengguna, Anda akan melihat output yang serupa dengan yang berikut ini.

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

```
postgres=>
```

Untuk informasi lebih lanjut tentang menghubungkan ke klaster DB Aurora PostgreSQL, lihat Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.. Jika Anda tidak dapat terhubung ke klaster DB Anda, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Untuk keamanan, ini adalah praktik terbaik untuk menggunakan koneksi terenkripsi. Hanya gunakan koneksi PostgreSQL yang tidak terenkripsi saat klien dan server berada di VPC yang sama dan jaringan tepercaya. Untuk informasi tentang cara menggunakan koneksi terenkripsi, lihat Mengamankan data Aurora PostgreSQL dengan SSL/TLS.

6. Jalankan perintah SQL.

   Misalnya, perintah SQL berikut menunjukkan tanggal dan waktu saat ini:

   ```
   SELECT CURRENT_TIMESTAMP;
   ```

## Langkah 4: Hapus EC2 instance dan cluster DB

Setelah Anda terhubung ke dan menjelajahi EC2 contoh contoh dan cluster DB yang Anda buat, hapus mereka sehingga Anda tidak lagi dikenakan biaya untuk mereka.

Jika Anda biasa AWS CloudFormation membuat sumber daya, lewati langkah ini dan lanjutkan ke langkah berikutnya.

Untuk menghapus EC2 instance

1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Di panel navigasi, pilih Instans.
3. Pilih EC2 instance, dan pilih Instance state, Terminate instance.
4. Pilih Akhiri saat diminta untuk konfirmasi.

Untuk informasi selengkapnya tentang menghapus EC2 instance, lihat Mengakhiri instance Anda di EC2 Panduan Pengguna Amazon.

Untuk menghapus klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data lalu pilih instans DB yang terkait dengan klaster DB.
3. Untuk Tindakan, pilih Hapus.
4. Pilih Hapus.

   Setelah semua instans DB yang terkait dengan klaster DB dihapus, klaster DB akan dihapus secara otomatis.

## (Opsional) Hapus EC2 instance dan cluster DB yang dibuat dengan CloudFormation

Jika Anda biasa AWS CloudFormation membuat sumber daya, hapus CloudFormation tumpukan setelah Anda terhubung dan jelajahi EC2 contoh contoh dan kluster DB, sehingga Anda tidak lagi dikenakan biaya untuk itu.

Untuk menghapus sumber CloudFormation daya

1. Buka AWS CloudFormation konsol.
2. Pada halaman Stacks di CloudFormation konsol, pilih tumpukan root (tumpukan tanpa nama VPCStack, BastionStack atau APGNS).
3. Pilih Hapus.
4. Pilih Hapus tumpukan saat diminta konfirmasi.

Untuk informasi selengkapnya tentang menghapus tumpukan CloudFormation, lihat Menghapus tumpukan di AWS CloudFormation konsol di AWS CloudFormation Panduan Pengguna.

## (Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda

Anda juga dapat menghubungkan cluster DB PostgreSQL Aurora Anda ke sumber daya komputasi tanpa server Lambda. Fungsi Lambda memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola infrastruktur. Fungsi Lambda juga memungkinkan Anda untuk otomatis merespons permintaan eksekusi kode pada skala apa pun, mulai dari belasan peristiwa dalam sehari hingga

ratusan per detik. Lihat informasi yang lebih lengkap di Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora.

# Tutorial: Membuat server web dan klaster DB Amazon Aurora

Tutorial ini menunjukkan cara menginstal server web Apache dengan PHP dan membuat basis data MariaDB, MySQL, atau PostgreSQL. Server web berjalan di instans Amazon EC2 menggunakan Amazon Linux 2023, dan Anda dapat memilih antara klaster DB Aurora MySQL atau Aurora PostgreSQL. Baik instans Amazon EC2 maupun klaster DB berjalan di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC.

**Important**

Tidak ada biaya untuk membuat akun AWS. Namun, dengan menyelesaikan tutorial ini, Anda mungkin akan dikenaki biaya untuk sumber daya AWS yang digunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

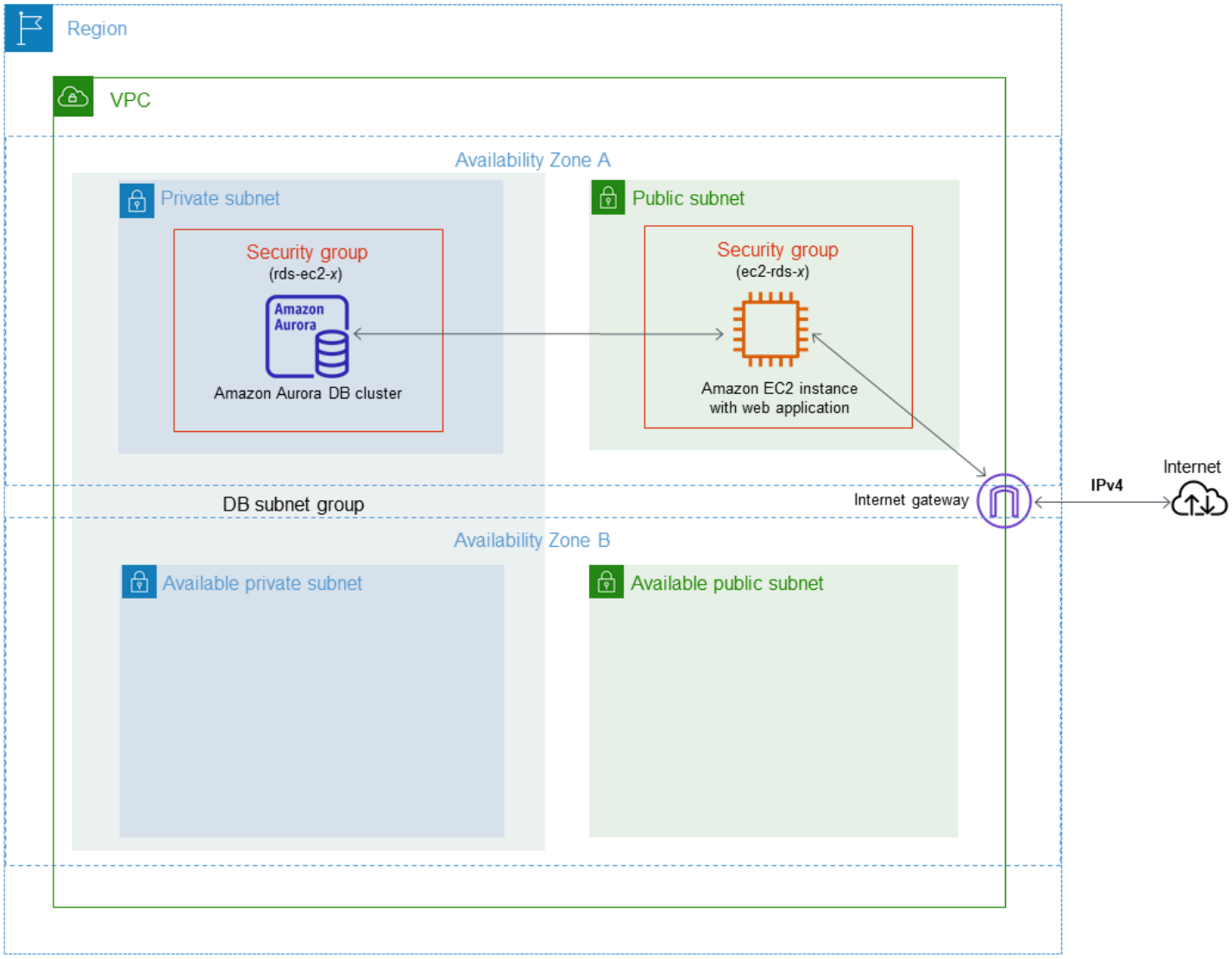
**Note**

Tutorial ini berfungsi dengan Amazon Linux 2023 dan mungkin tidak berfungsi untuk versi Linux lainnya.

Dalam tutorial berikut, Anda membuat instans EC2 yang menggunakan VPC, subnet, dan grup keamanan default untuk Akun AWS Anda. Tutorial ini menunjukkan cara membuat klaster DB dan secara otomatis menyiapkan konektivitas dengan instans EC2 yang Anda buat. Tutorial kemudian menunjukkan cara menginstal server web pada instans EC2. Anda menghubungkan server web ke klaster DB di VPC menggunakan titik akhir penulis klaster DB.

1. Luncurkan EC2 instance untuk terhubung dengan Anda
2. Membuat klaster DB Amazon Aurora
3. Menginstal server web di instans EC2 Anda

Diagram berikut menunjukkan konfigurasi setelah tutorial selesai.



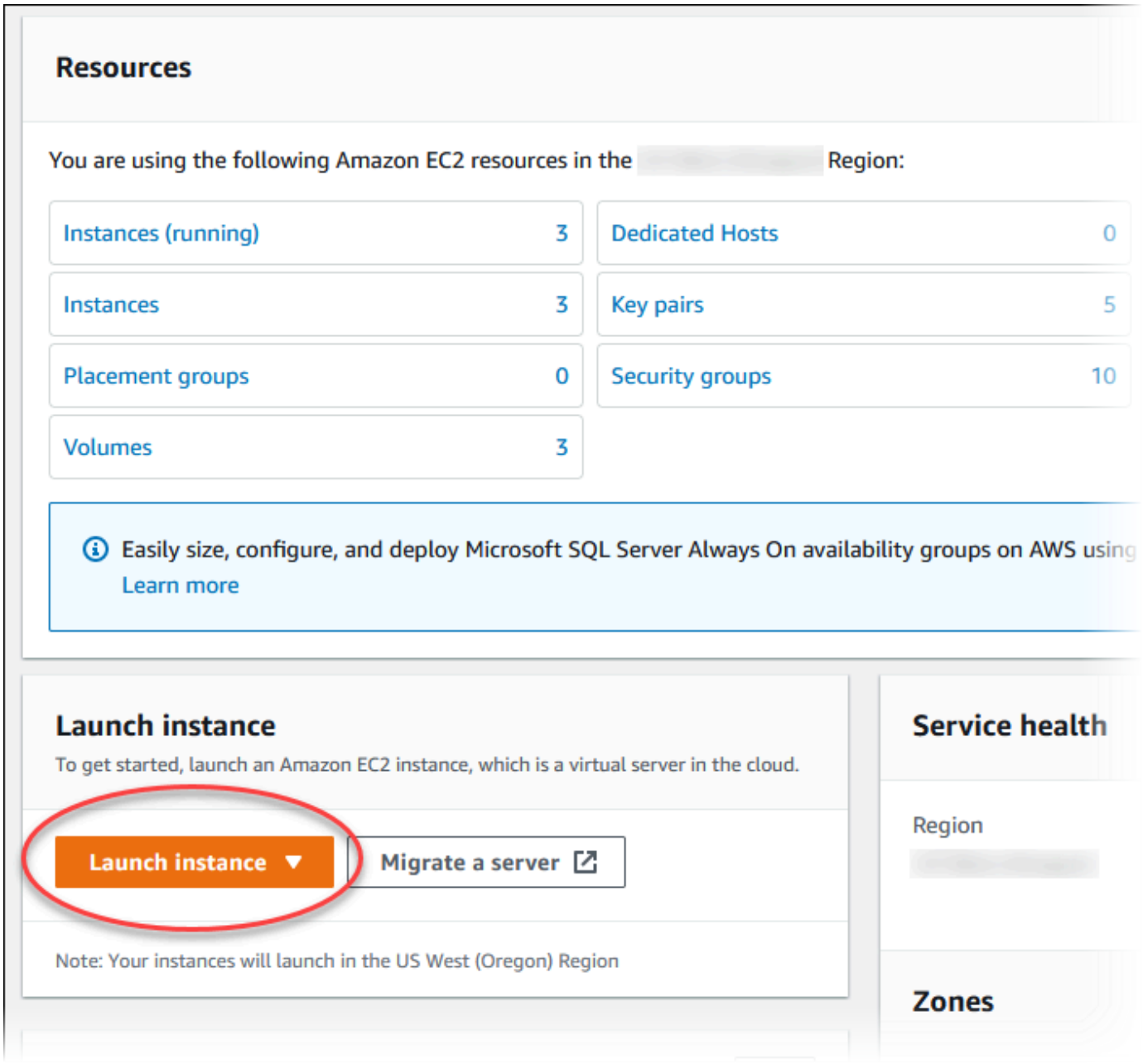

> **Note**
>
> Setelah Anda menyelesaikan tutorial, ada subnet publik dan privat di setiap Zona Ketersediaan di VPC Anda. Tutorial ini menggunakan VPC default untuk Akun AWS Anda dan secara otomatis menyiapkan konektivitas antara instans EC2 dan klaster DB. Jika Anda lebih suka mengonfigurasi VPC baru untuk skenario ini, selesaikan tugas di Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .

## Luncurkan EC2 instance untuk terhubung dengan Anda

Buat EC2 instans Amazon di subnet publik AndaVPC.

Untuk meluncurkan sebuah EC2 instance

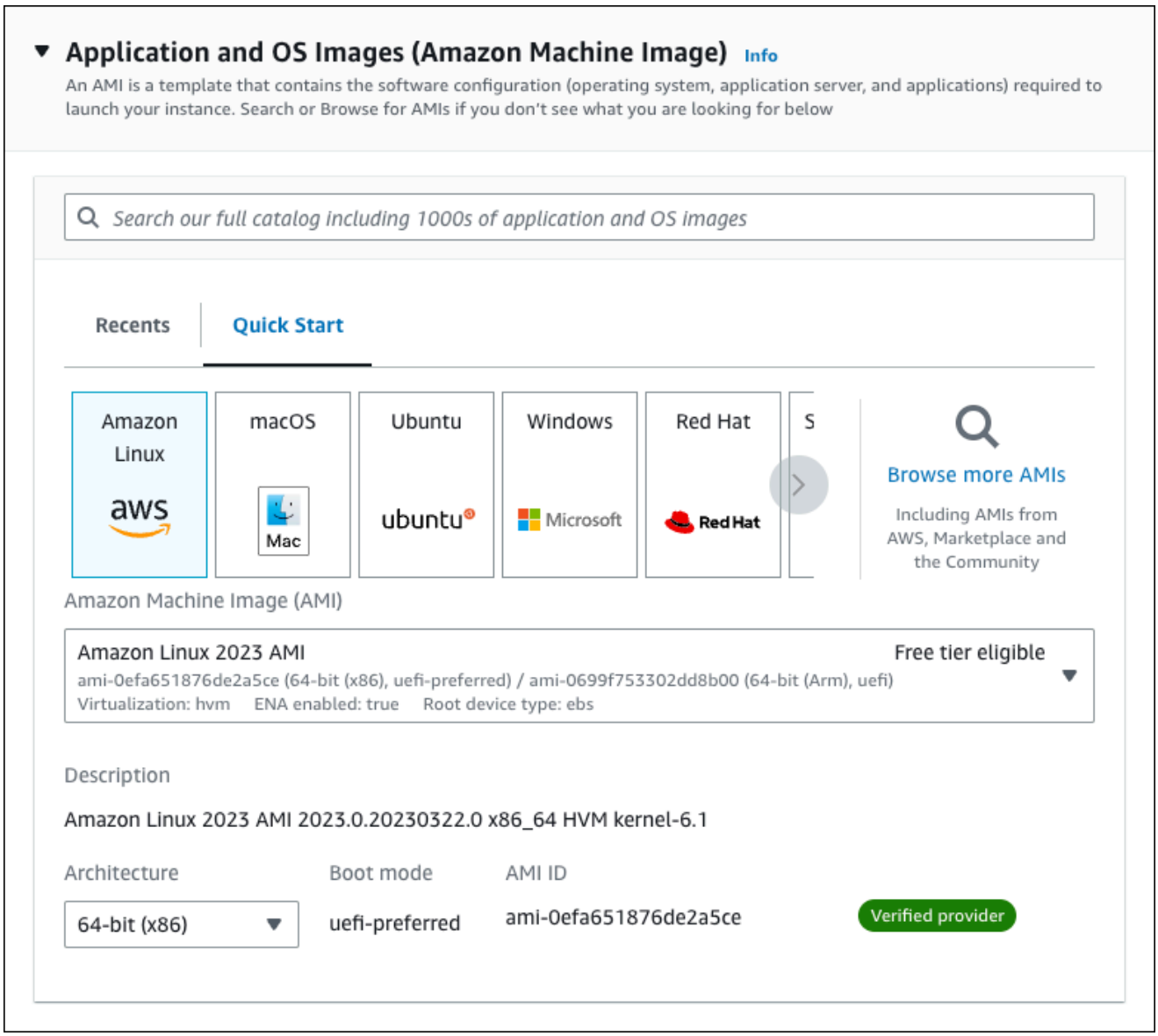
1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS tempat Anda ingin membuat instance. EC2
3. Pilih EC2Dasbor, lalu pilih Launch instance, seperti yang ditunjukkan berikut.




4. Pilih pengaturan berikut di halaman Luncurkan instans.

a. Di bagian Nama dan tag, untuk Nama, masukkan **`tutorial-ec2-instance-web-server`**.

b. Di bawah Gambar Aplikasi dan OS (Gambar Mesin Amazon), pilih Amazon Linux, lalu pilih Amazon Linux 2023 AMI. Biarkan default untuk pilihan lain.




c. Di bagian Jenis instans, pilih t2.micro.

d. Pada Pasangan kunci (login), pilih Nama pasangan kunci untuk menggunakan pasangan kunci yang ada. Untuk membuat key pair baru untuk EC2 instance Amazon, pilih Create new key pair dan kemudian gunakan jendela Create key pair untuk membuatnya.

   Untuk informasi selengkapnya tentang membuat key pair baru, lihat Membuat key pair di Panduan EC2 Pengguna Amazon.

e. Di bagian Pengaturan jaringan, atur nilai-nilai ini dan biarkan nilai-nilai lainnya sebagai default:

- Untuk Izinkan SSH lalu lintas dari, pilih sumber SSH koneksi ke EC2 instance.

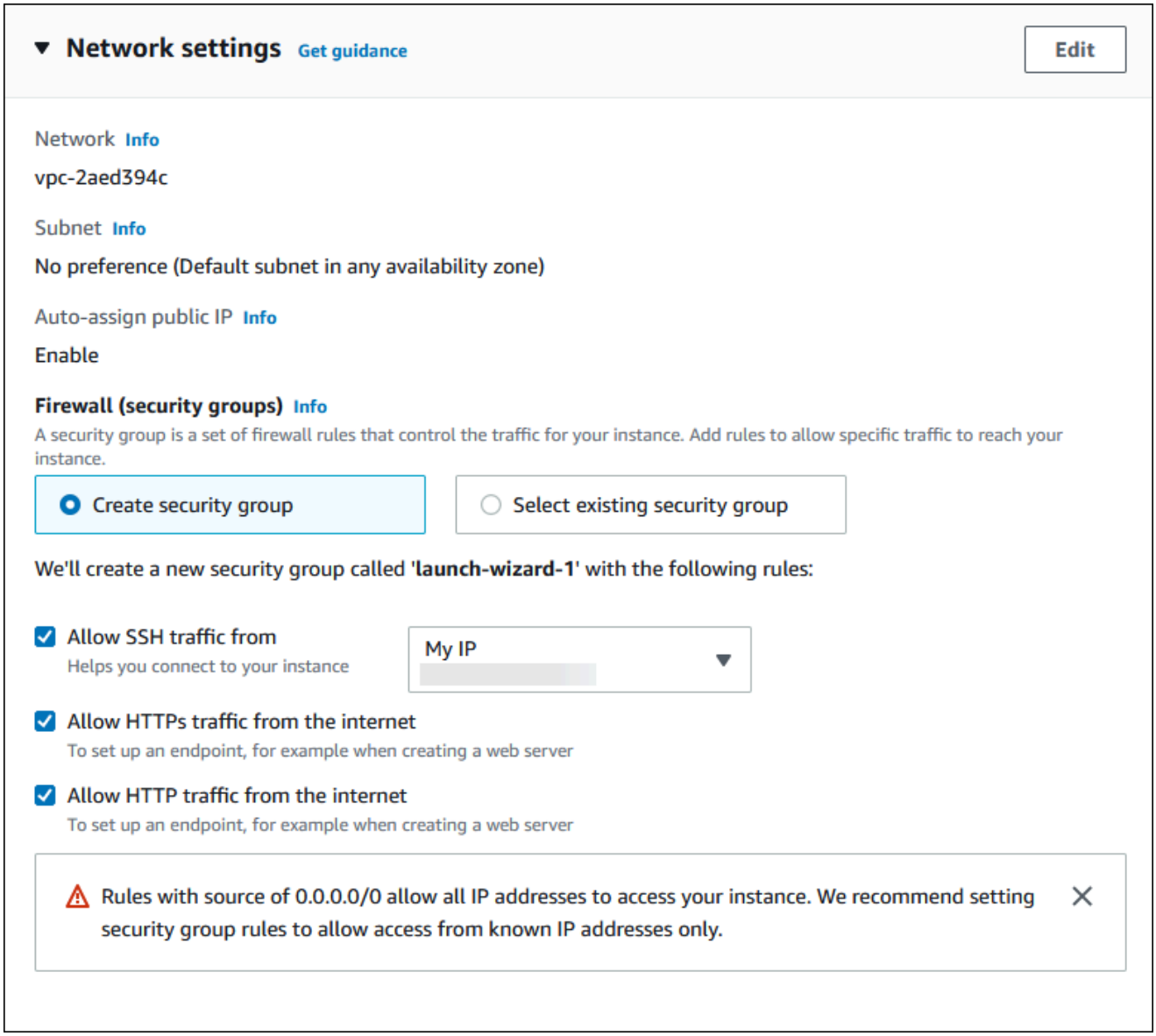
  Anda dapat memilih IP Saya jika alamat IP yang ditampilkan benar untuk SSH koneksi.

  Jika tidak, Anda dapat menentukan alamat IP yang akan digunakan untuk menyambung ke EC2 instance dalam VPC menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, Anda dapat membuka layanan di https://checkip.amazonaws.com di jendela atau tab browser lain. Contoh alamat IP adalah `203.0.113.25/32`.

  Dalam banyak kasus, Anda mungkin terhubung melalui penyedia layanan internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, tentukan rentang alamat IP yang digunakan oleh komputer klien.

  > **Warning**
  >
  > Jika Anda menggunakan `0.0.0.0/0` untuk SSH akses, Anda memungkinkan semua alamat IP untuk mengakses instans publik Anda menggunakanSSH. Pendekatan ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, otorisasi hanya alamat IP tertentu atau rentang alamat untuk mengakses instans Anda menggunakan. SSH

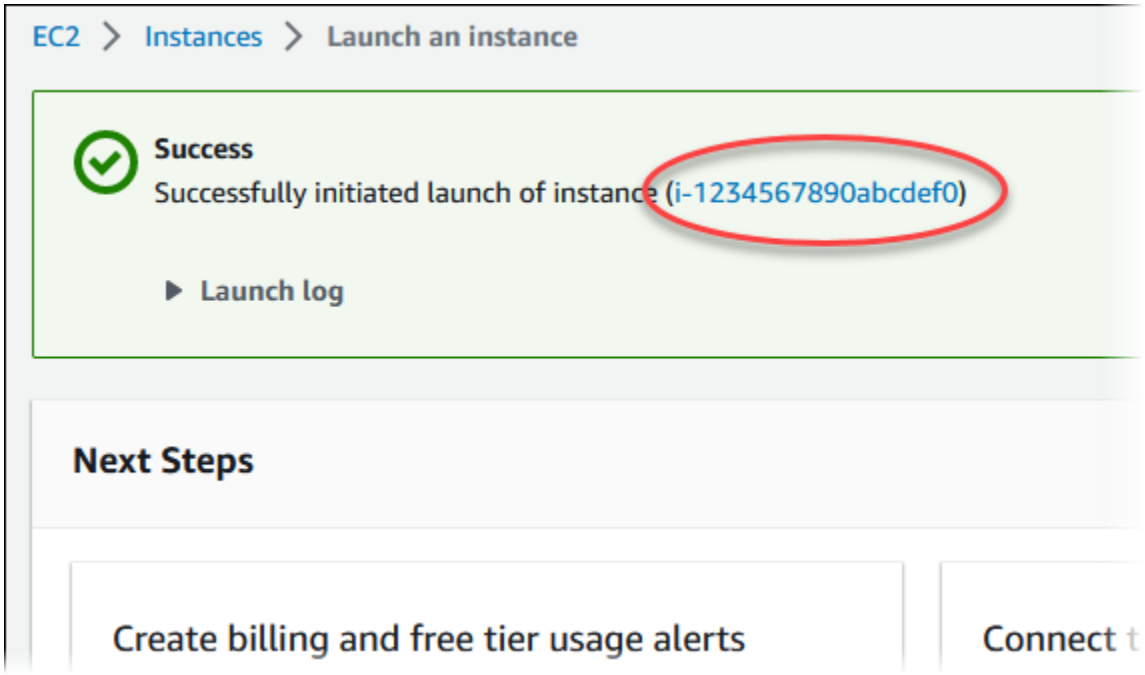
- Aktifkan Izinkan HTTPs lalu lintas dari internet.
- Aktifkan Izinkan HTTP lalu lintas dari internet.




f. Biarkan nilai default untuk bagian yang lainnya.

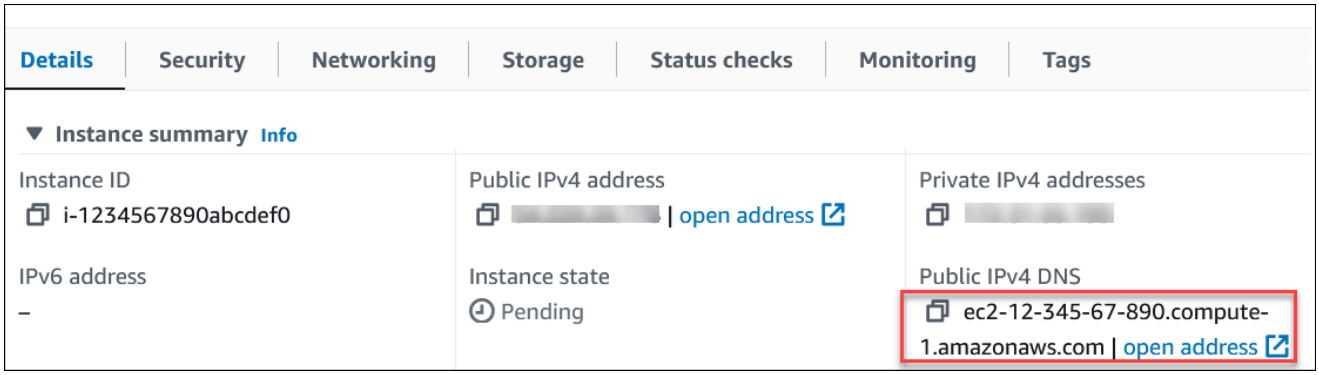
g. Tinjau ringkasan konfigurasi instans di panel Ringkasan, dan ketika Anda siap, pilih Luncurkan instans.

5. Pada halaman Status Peluncuran, perhatikan pengenal untuk EC2 instance baru Anda, misalnya:`i-1234567890abcdef0`.

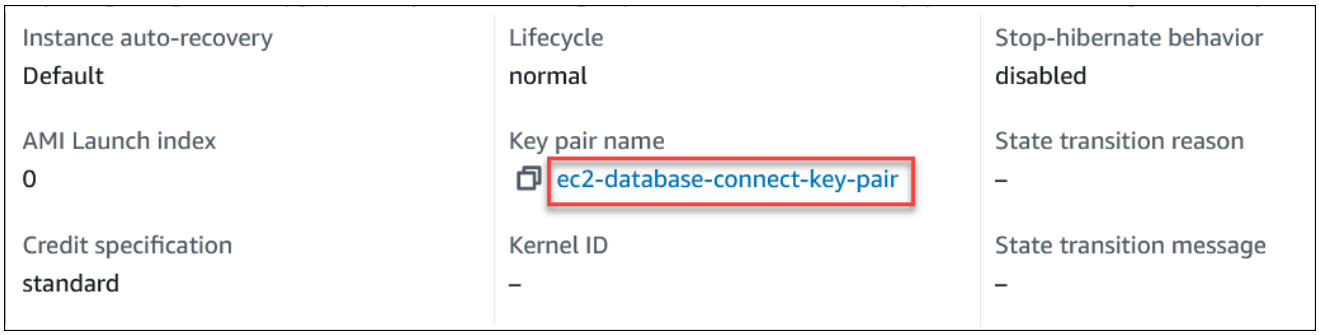



6. Pilih pengenal EC2 instance untuk membuka daftar EC2 instance, lalu pilih instance AndaEC2.
7. Di tab Detail, perhatikan nilai-nilai berikut, yang Anda butuhkan saat Anda terhubung menggunakanSSH:

   a. Dalam ringkasan Instance, perhatikan nilai untuk Public IPv4 DNS.




   b. Di Detail instans, catat nilai untuk Nama pasangan kunci.



8. Tunggu hingga status instans untuk instans Anda Berjalan sebelum melanjutkan.
9. Selesaikan Membuat klaster DB Amazon Aurora.

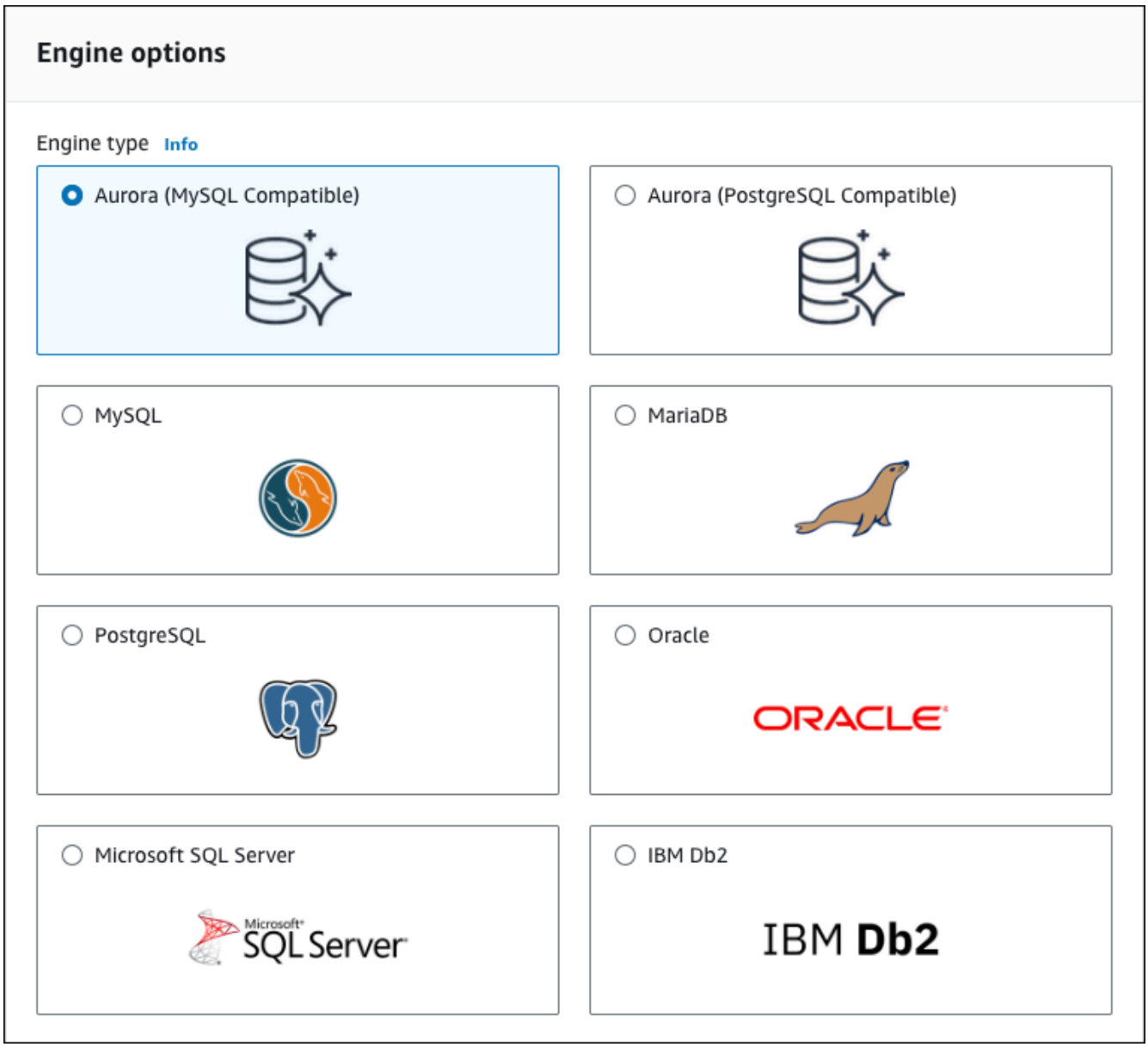
## Membuat klaster DB Amazon Aurora

Buat klaster DB Amazon Aurora MySQL atau Aurora PostgreSQL DB yang mempertahankan data yang digunakan oleh aplikasi web.

Aurora MySQL

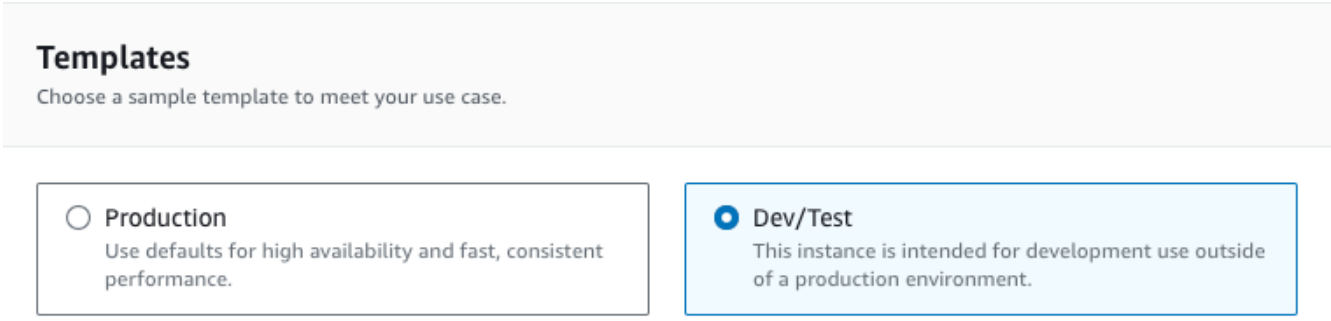
Untuk membuat klaster DB Aurora MySQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pastikan Wilayah AWS sama dengan tempat Anda membuat instans EC2.
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.
5. Di halaman Buat basis data, pilih Pembuatan standar.
6. Untuk Opsi mesin, pilih Aurora (Kompatibel dengan MySQL).



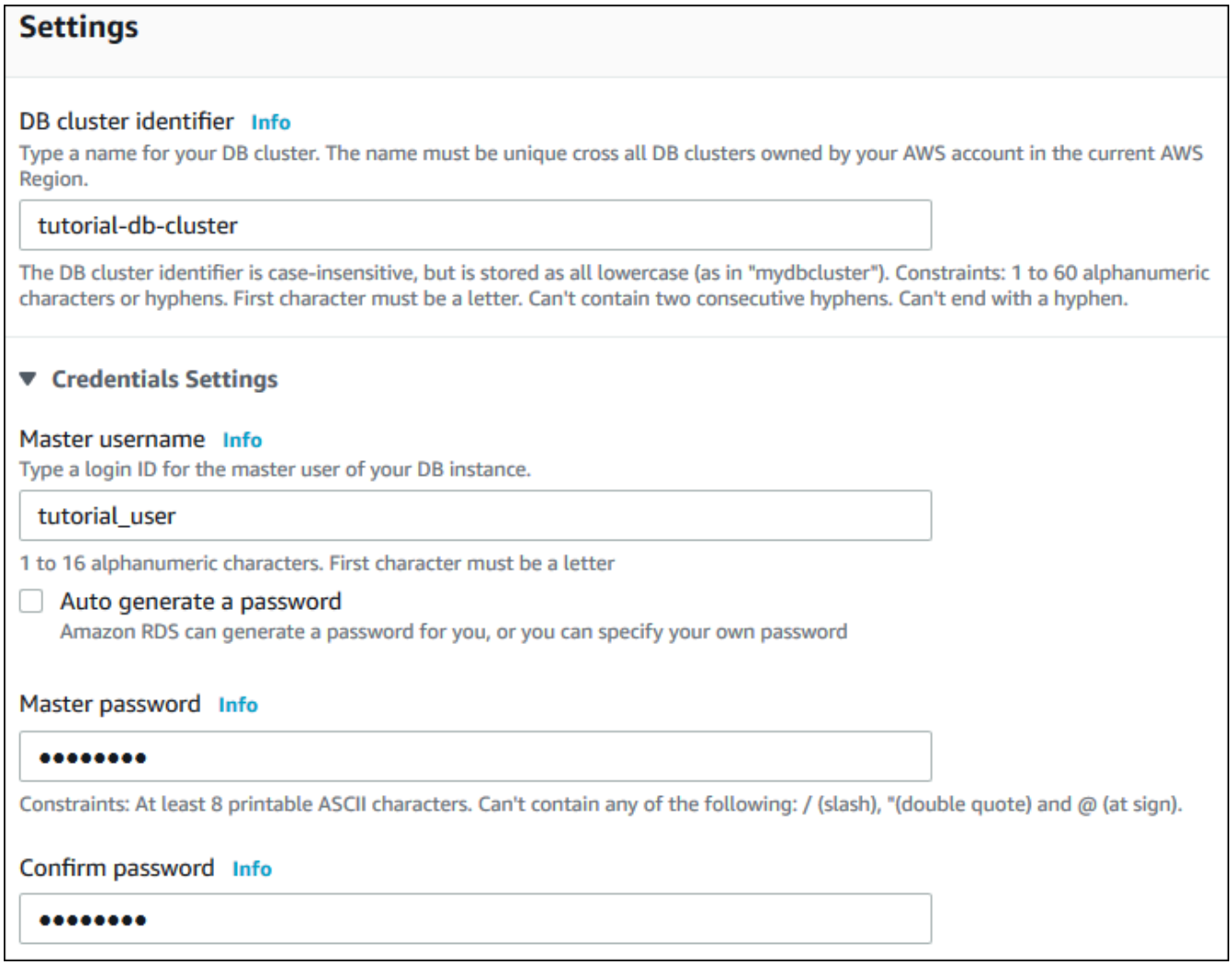

Pertahankan nilai default untuk Versi dan opsi mesin lain.

7. Di bagian Templat, pilih Dev/Test.



8. Di bagian Pengaturan, atur nilai-nilai ini:

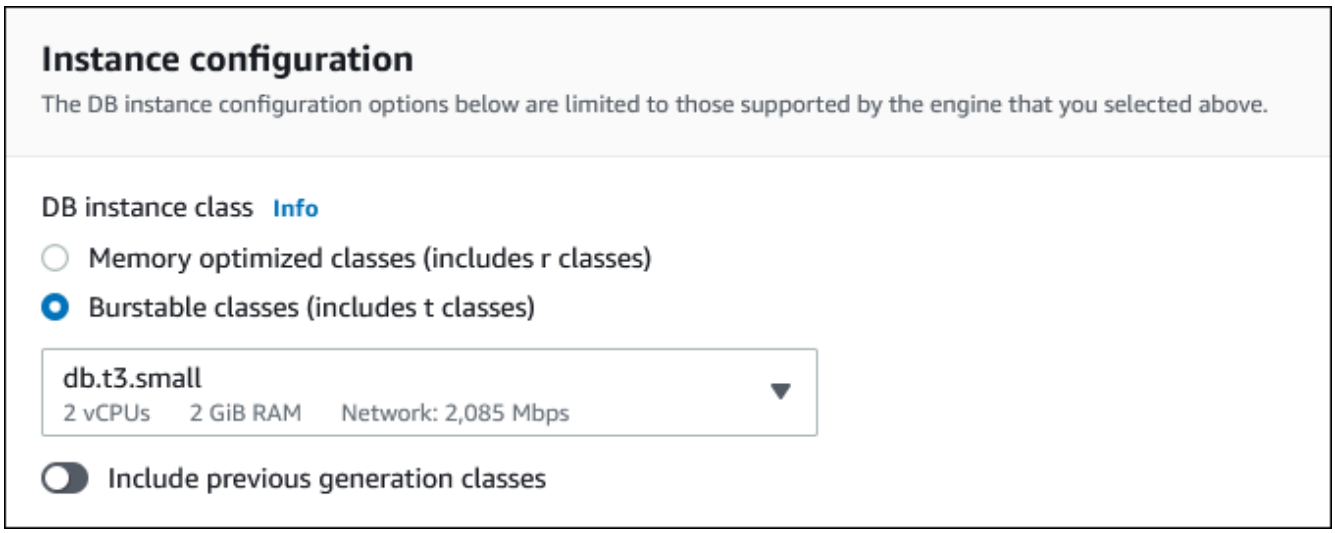
   - Pengidentifikasi klaster DB – Ketikkan **`tutorial-db-cluster`**.
   - Nama pengguna utama – Ketikkan **`tutorial_user`**.
   - Buat kata sandi secara otomatis – Biarkan opsi nonaktif.
   - Kata sandi utama – Ketikkan kata sandi.
   - Konfirmasi kata sandi – Ketik ulang kata sandi.



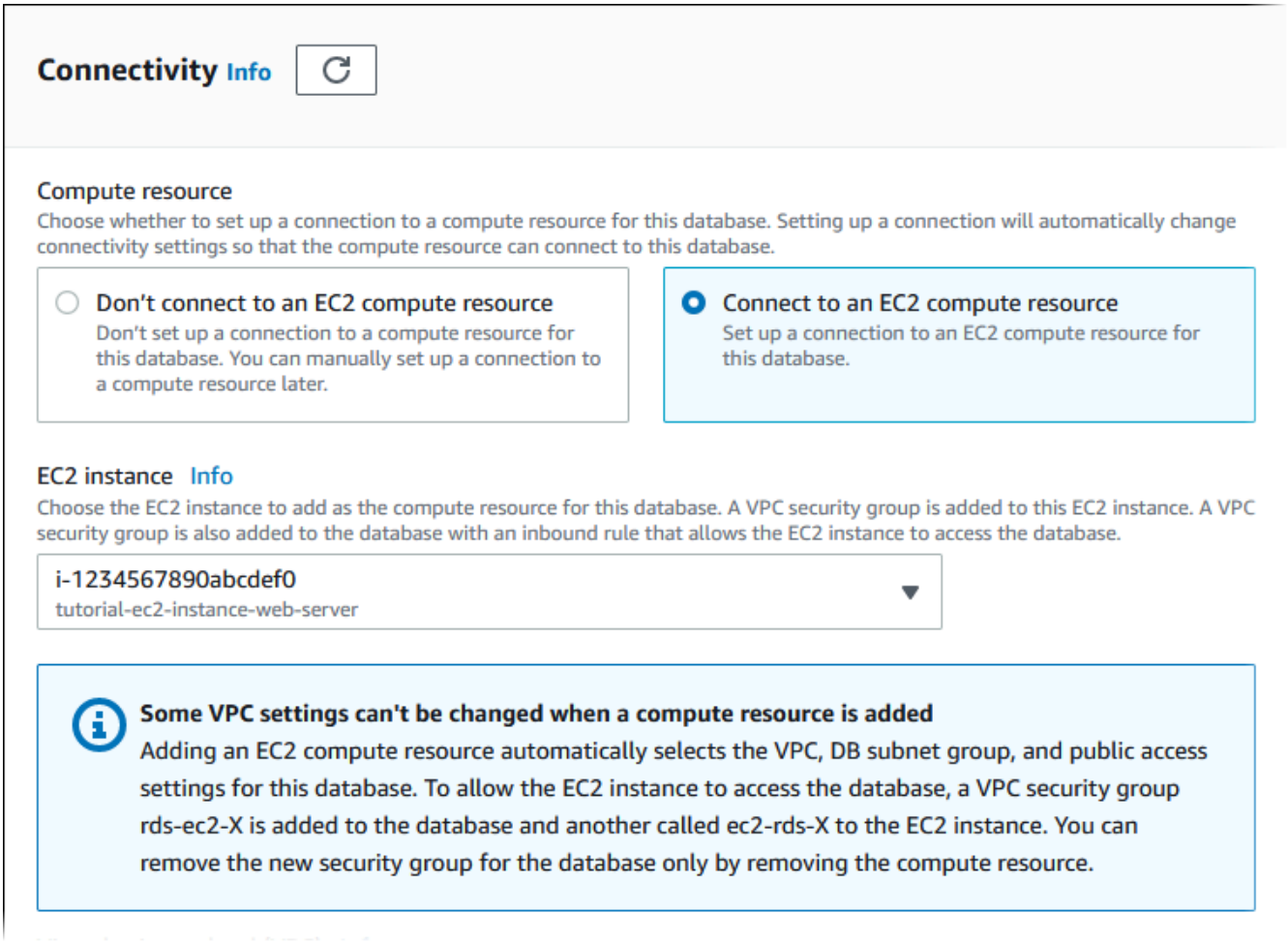

9. Di bagian Konfigurasi instans, atur nilai-nilai ini:

   - Kelas runtutan (termasuk kelas t)
   - db.t3.small atau db.t3.medium

> **Note**
>
> Sebaiknya hanya gunakan kelas instans DB T untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.

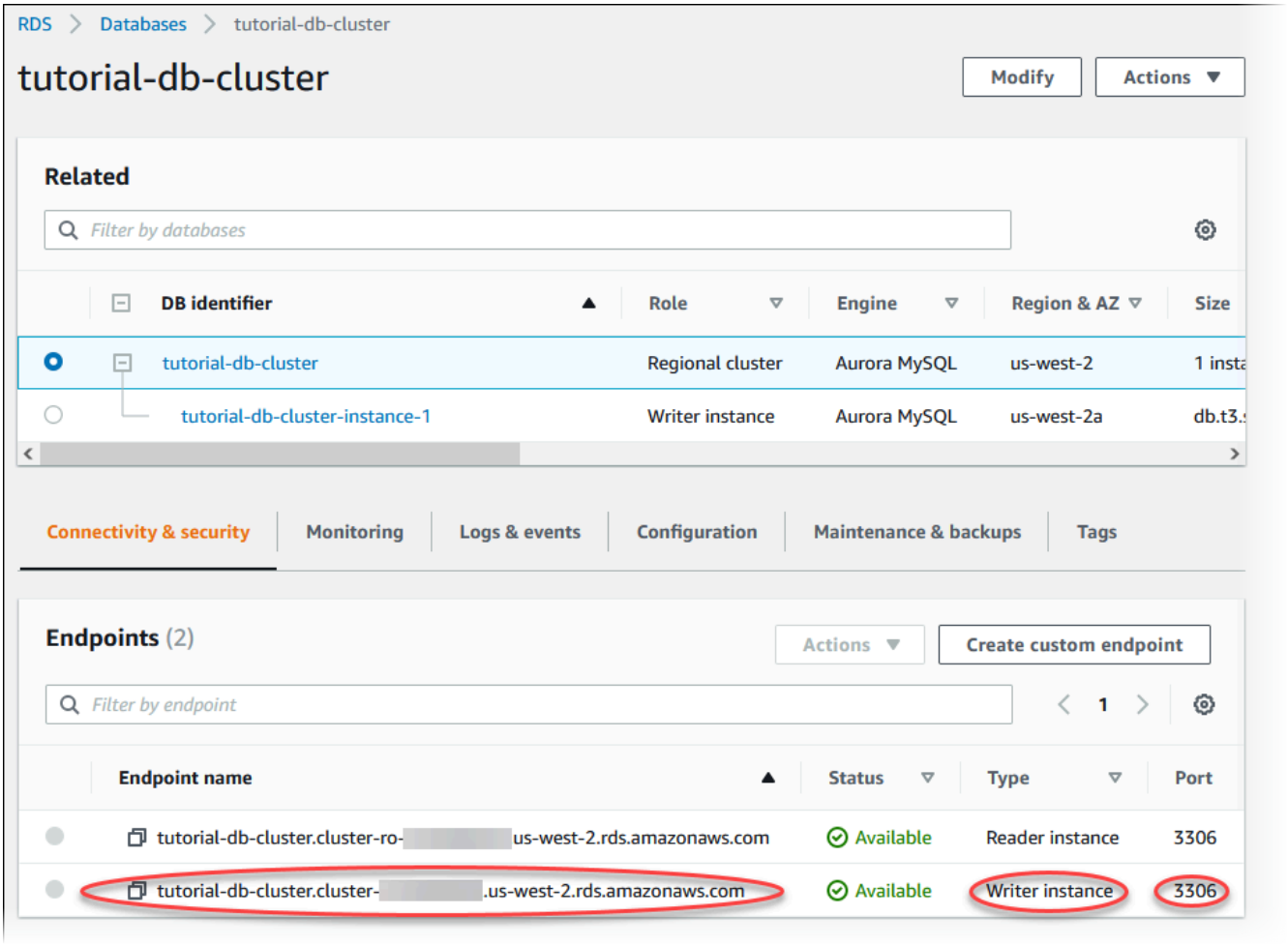



10. Di bagian Ketersediaan dan daya tahan, gunakan nilai default.
11. Di bagian Konektivitas, atur nilai-nilai ini dan biarkan nilai lainnya sebagai default:
    - Untuk sumber daya Komputasi, pilih Hubungkan ke sumber daya komputasi EC2.
    - Untuk contoh EC2, pilih instans EC2 yang Anda buat sebelumnya, seperti tutorial-ec2 -. instance-web-server




12. Buka bagian Konfigurasi tambahan, dan masukkan **`sample`** untuk Nama database awal. Biarkan opsi lainnya menggunakan pengaturan default.
13. Untuk membuat klaster DB Aurora MySQL, pilih Buat basis data.

    Klaster DB baru Anda muncul di daftar Basis Data dengan status Membuat.
14. Tunggu sampai Status klaster DB baru Anda menampilkan status Tersedia. Lalu pilih nama klaster DB untuk menampilkan detailnya.
15. Di bagian Konektivitas & keamanan, lihat Titik Akhir dan Port instans DB tulis.




Catat titik akhir dan port untuk instans DB tulis Anda. Gunakan informasi ini untuk menghubungkan server web Anda ke klaster DB Anda.

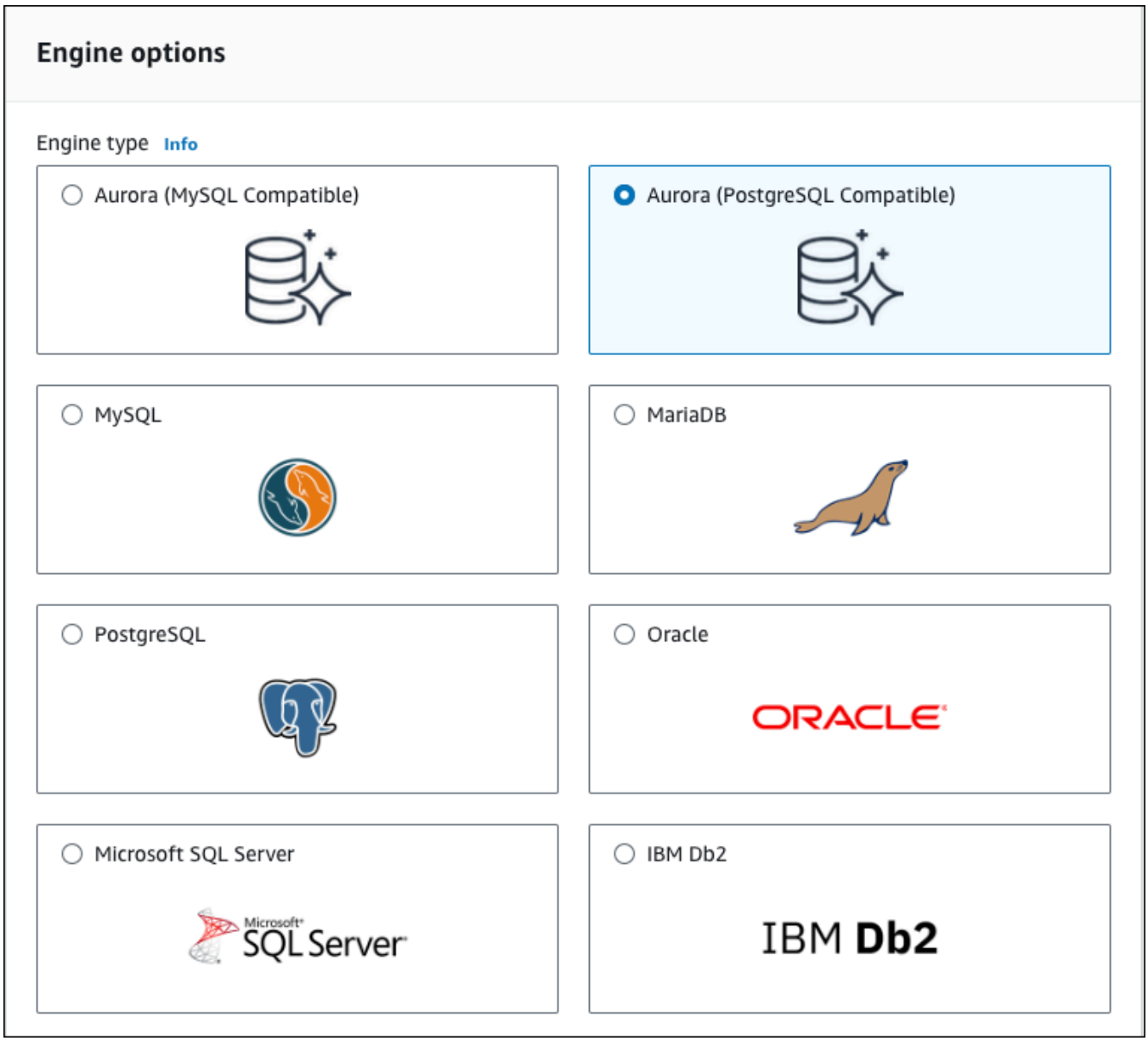
16. Selesaikan Menginstal server web di instans EC2 Anda.

## Aurora PostgreSQL

Untuk membuat klaster DB Aurora PostgreSQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pastikan Wilayah AWS sama dengan tempat Anda membuat instans EC2.
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.

5. Di halaman Buat basis data, pilih Pembuatan standar.
6. Untuk Opsi mesin, pilih Aurora (Kompatibel dengan PostgreSQL).

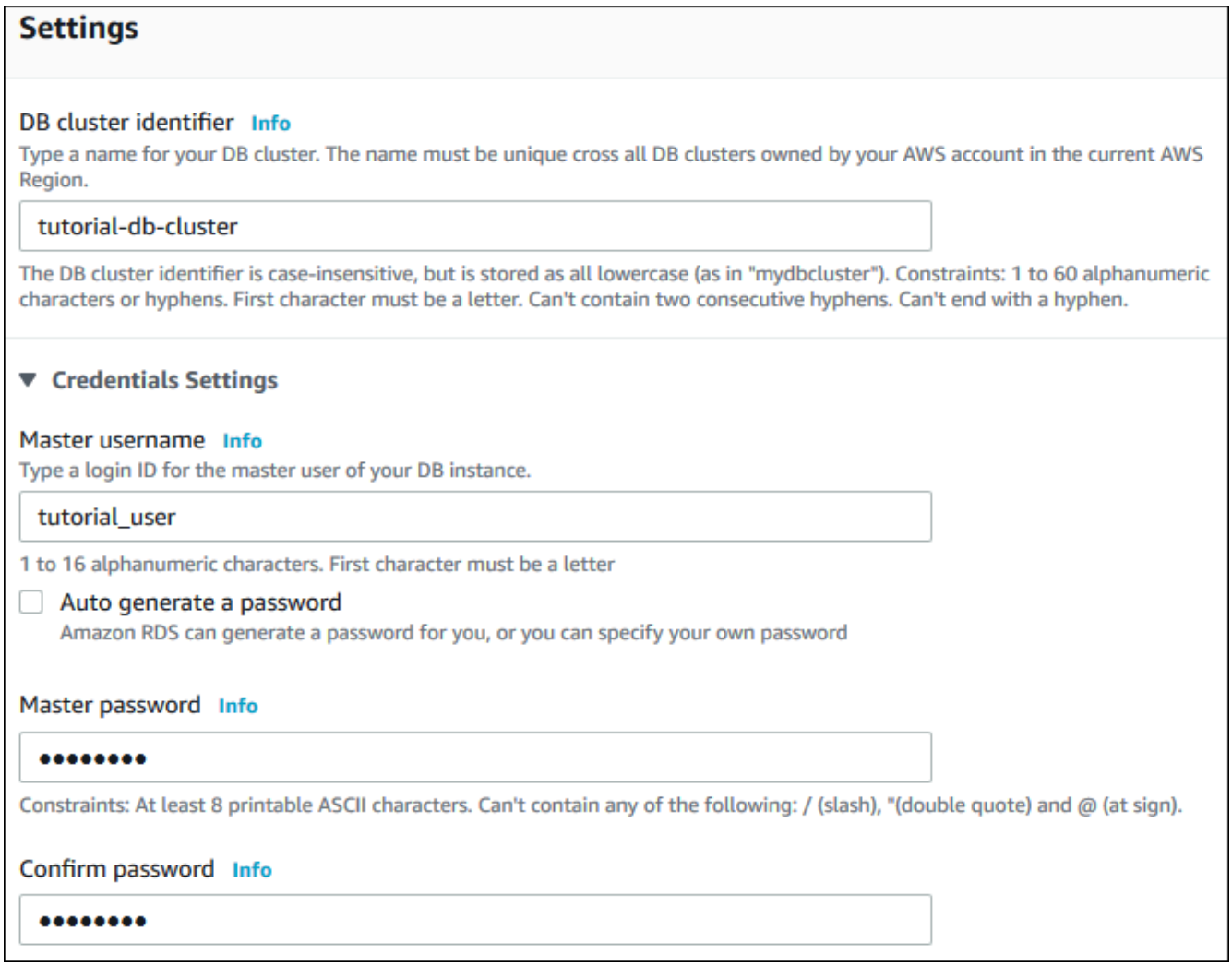



Pertahankan nilai default untuk Versi dan opsi mesin lain.

7. Di bagian Templat, pilih Dev/Test.

8. Di bagian Pengaturan, atur nilai-nilai ini:

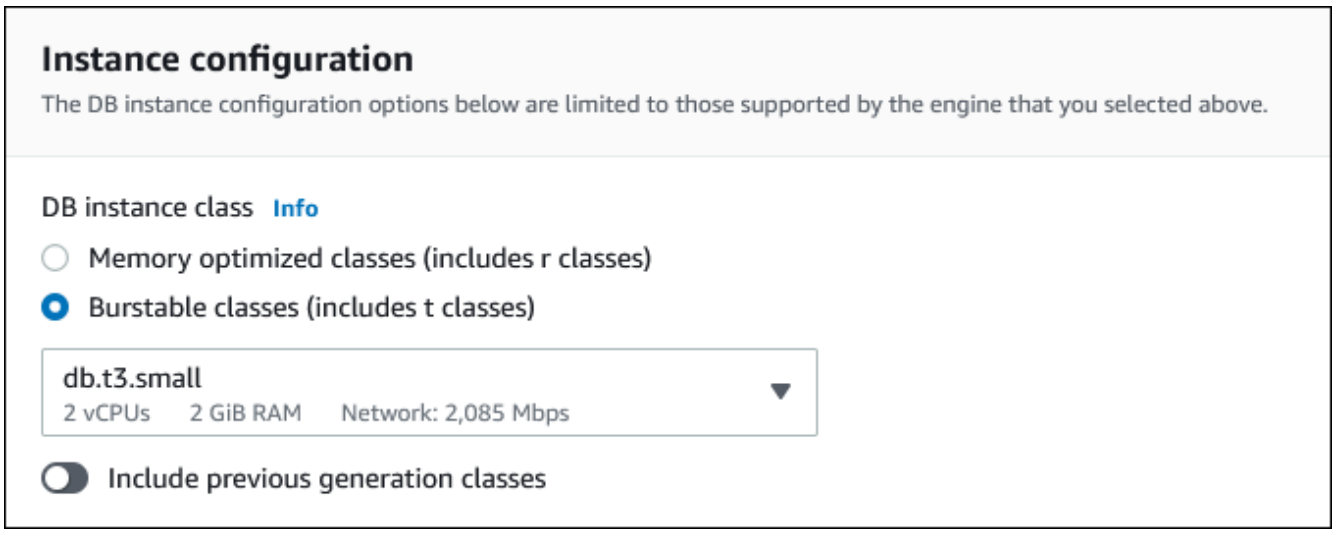
   - Pengidentifikasi klaster DB – Ketikkan **`tutorial-db-cluster`**.
   - Nama pengguna utama – Ketikkan **`tutorial_user`**.
   - Buat kata sandi secara otomatis – Biarkan opsi nonaktif.
   - Kata sandi utama – Ketikkan kata sandi.
   - Konfirmasi kata sandi – Ketik ulang kata sandi.




9. Di bagian Konfigurasi instans, atur nilai-nilai ini:

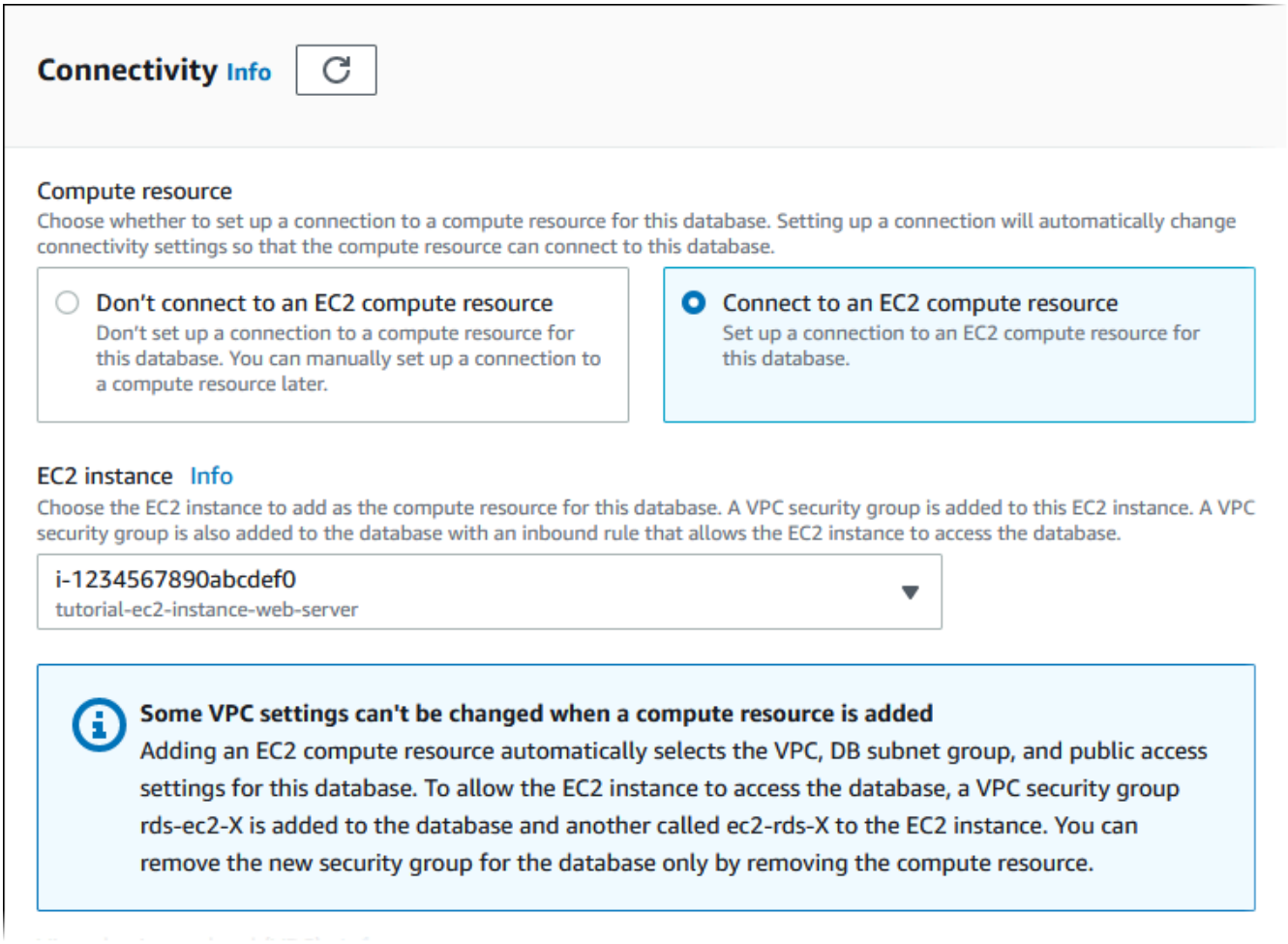
   - Kelas runtutan (termasuk kelas t)
   - db.t3.small atau db.t3.medium

> **Note**
>
> Sebaiknya hanya gunakan kelas instans DB T untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.



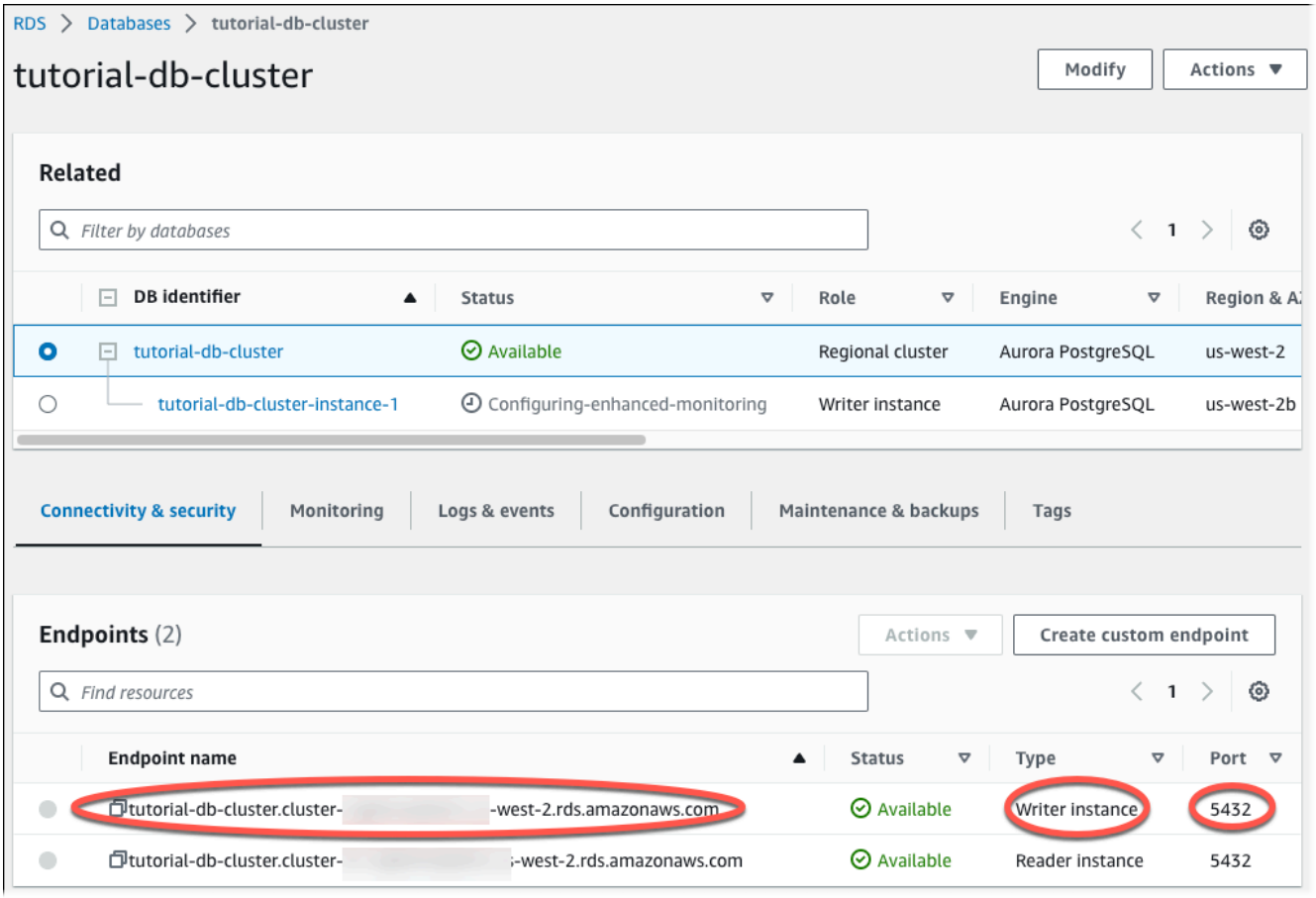

10. Di bagian Ketersediaan dan daya tahan, gunakan nilai default.
11. Di bagian Konektivitas, atur nilai-nilai ini dan biarkan nilai lainnya sebagai default:

    - Untuk sumber daya Komputasi, pilih Hubungkan ke sumber daya komputasi EC2.
    - Untuk contoh EC2, pilih instans EC2 yang Anda buat sebelumnya, seperti tutorial-ec2 -. instance-web-server




12. Buka bagian Konfigurasi tambahan, dan masukkan **`sample`** untuk Nama database awal. Biarkan opsi lainnya menggunakan pengaturan default.
13. Untuk membuat klaster DB Aurora PostgreSQL, pilih Buat basis data.

    Klaster DB baru Anda muncul di daftar Basis Data dengan status Membuat.
14. Tunggu sampai Status klaster DB baru Anda menampilkan status Tersedia. Lalu pilih nama klaster DB untuk menampilkan detailnya.
15. Di bagian Konektivitas & keamanan, lihat Titik Akhir dan Port instans DB tulis.




Catat titik akhir dan port untuk instans DB tulis Anda. Gunakan informasi ini untuk menghubungkan server web Anda ke klaster DB Anda.

16. Selesaikan Menginstal server web di instans EC2 Anda.

## Menginstal server web di instans EC2 Anda

Instal server web pada instans EC2 yang Anda buat di Luncurkan EC2 instance untuk terhubung dengan Anda. Server web terhubung ke klaster DB Amazon Aurora yang Anda buat di Membuat klaster DB Amazon Aurora.

### Menginstal server web Apache dengan PHP dan MariaDB

Hubungkan ke instans EC2 Anda dan instal server web.

Menghubungkan ke instans EC2 dan menginstal server web Apache dengan PHP

1. Connect ke instans EC2 yang Anda buat sebelumnya dengan mengikuti langkah-langkah di Connect to Linux Anda di Panduan Pengguna Amazon EC2.

Sebaiknya Anda menghubungkan ke instans EC2 menggunakan SSH. Jika utilitas klien SSH diinstal di Windows, Linux, atau Mac, Anda dapat menghubungkan ke instans menggunakan format perintah berikut:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

Misalnya, asumsikan bahwa `ec2-database-connect-key-pair.pem` disimpan di `/dir1` di Linux, dan DNS IPv4 publik untuk instans EC2 Anda adalah `ec2-12-345-678-90.compute-1.amazonaws.com`. Perintah SSH Anda akan tampak seperti berikut:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

2. Dapatkan pembaruan keamanan dan perbaikan bug terbaru dengan memperbarui perangkat lunak di instans EC2 Anda. Untuk melakukannya, gunakan perintah berikut.

> **Note**
>
> Opsi `-y` menginstal pembaruan tanpa meminta konfirmasi. Hilangkan opsi ini untuk memeriksa pembaruan sebelum menginstal.

```
sudo dnf update -y
```

3. Setelah pembaruan selesai, instal server web Apache, PHP, dan MariaDB atau perangkat lunak PostgreSQL menggunakan perintah berikut. Perintah ini menginstal beberapa paket perangkat lunak dan dependensi terkait bersamaan.

   MariaDB & MySQL

   ```
   sudo dnf install -y httpd php php-mysqli mariadb105
   ```

   PostgreSQL

   ```
   sudo dnf install -y httpd php php-pgsql postgresql15
   ```

Jika Anda mengalami kesalahan, instans Anda mungkin tidak diluncurkan dengan AMI Amazon Linux 2023. Sebaiknya gunakan AMI Amazon Linux 2 AMI. Anda dapat melihat versi Amazon Linux Anda menggunakan perintah berikut.

```
cat /etc/system-release
```

Untuk informasi selengkapnya, lihat Memperbarui perangkat lunak instans.

4. Mulai server web dengan perintah yang ditampilkan berikut ini.

```
sudo systemctl start httpd
```

Anda dapat menguji apakah server web Anda terinstal dan berjalan dengan benar. Untuk melakukannya, masukkan nama Sistem Nama Domain (DNS) publik dari instans EC2 Anda di bilah alamat browser web, misalnya: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`. Jika server web Anda berjalan, maka Anda akan melihat halaman uji Apache.

Jika Anda tidak melihat halaman uji Apache, periksa aturan masuk Anda untuk grup keamanan VPC yang Anda buat di Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 . Pastikan aturan masuk Anda menyertakan aturan yang mengizinkan akses HTTP (port 80) untuk alamat IP agar terhubung ke server web.

> **Note**
>
> Halaman uji Apache hanya muncul jika tidak ada konten di direktori root dokumen, `/var/www/html`. Setelah konten ditambahkan ke direktori root dokumen, konten tersebut akan muncul di alamat DNS publik dari instans EC2 Anda. Sebelumnya, konten tersebut muncul di halaman uji Apache.

5. Konfigurasikan server web untuk memulai setiap boot sistem menggunakan perintah `systemctl`.

```
sudo systemctl enable httpd
```

Untuk mengizinkan `ec2-user` mengelola file di direktori root default untuk server web Apache Anda, ubah kepemilikan dan izin direktori `/var/www`. Ada banyak cara untuk menyelesaikan tugas ini. Dalam tutorial ini, Anda menambahkan `ec2-user` ke grup `apache`, untuk memberikan kepemilikan grup `apache` atas direktori `/var/www` dan menetapkan izin tulis ke grup.

Mengatur izin file untuk server web Apache

1. Tambahkan pengguna `ec2-user` ke grup `apache`.

   ```
   sudo usermod -a -G apache ec2-user
   ```

2. Keluar untuk menyegarkan izin Anda dan masukkan grup `apache` baru.

   ```
   exit
   ```

3. Masuk kembali dan verifikasi apakah grup `apache` ada dengan perintah `groups`.

   ```
   groups
   ```

   Output Anda akan terlihat seperti berikut ini:

   ```
   ec2-user adm wheel apache systemd-journal
   ```

4. Ubah kepemilikan grup atas direktori `/var/www` dan kontennya ke grup `apache`.

   ```
   sudo chown -R ec2-user:apache /var/www
   ```

5. Ubah izin direktori atas `/var/www` dan subdirektorinya untuk menambahkan izin tulis grup dan atur ID grup pada subdirektori yang dibuat di masa mendatang.

   ```
   sudo chmod 2775 /var/www
   find /var/www -type d -exec sudo chmod 2775 {} \;
   ```

6. Ubah izin file secara berulang di direktori `/var/www` dan subdirektorinya untuk menambahkan izin tulis grup.

   ```
   find /var/www -type f -exec sudo chmod 0664 {} \;
   ```

Sekarang, `ec2-user` (dan setiap anggota grup `apache` mendatang) dapat menambahkan, menghapus, dan mengedit file pada root dokumen Apache. Ini memungkinkan Anda untuk menambahkan konten, seperti situs web statis atau aplikasi PHP.

> **Note**
>
> Server web yang menjalankan protokol HTTP tidak memberikan keamanan transportasi untuk data yang dikirim atau diterimanya. Saat Anda menghubungkan ke server HTTP menggunakan browser web, banyak informasi yang terlihat oleh penyadap di mana saja di sepanjang jalur jaringan. Informasi ini mencakup URL yang Anda kunjungi, konten halaman web yang Anda terima, dan konten (termasuk kata sandi) dari setiap formulir HTML.
> Praktik terbaik untuk mengamankan server web Anda adalah dengan menginstal dukungan untuk HTTPS (HTTP Secure). Protokol ini melindungi data Anda dengan enkripsi SSL/TLS. Untuk informasi selengkapnya, lihat Tutorial: Mengonfigurasi SSL/TLS dengan Amazon Linux AMI di Panduan Pengguna Amazon EC2.

## Menghubungkan server web Apache ke klaster DB

Selanjutnya, Anda menambahkan konten ke server web Apache yang terhubung ke klaster DB Amazon Aurora.

Menambahkan konten ke server web Apache yang terhubung ke klaster DB Anda

1. Saat masih terhubung ke instans EC2, ubah direktori ke `/var/www` dan buat subdirektori baru yang diberi nama `inc`.

   ```
   cd /var/www
   mkdir inc
   cd inc
   ```

2. Buat file baru dalam direktori `inc` yang diberi nama `dbinfo.inc`, lalu edit file tersebut dengan menggunakan nano (atau editor pilihan Anda).

   ```
   >dbinfo.inc
   nano dbinfo.inc
   ```

3. Tambahkan konten berikut ini ke file `dbinfo.inc`. Di sini, *`db_instance_endpoint`* adalah titik akhir tulis klaster DB, tanpa port, untuk klaster DB Anda.

> **Note**
>
> Sebaiknya tempatkan nama pengguna dan informasi kata sandi dalam folder yang bukan bagian dari root dokumen untuk server web Anda. Hal ini mengurangi kemungkinan informasi keamanan Anda terungkap.
> Pastikan untuk mengubah `master password` ke kata sandi yang sesuai di aplikasi Anda.

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. Simpan dan tutup file `dbinfo.inc`. Jika Anda menggunakan nano, simpan dan tutup file dengan menggunakan Ctrl+S dan Ctrl+X.
5. Ubah direktori ke `/var/www/html`.

```
cd /var/www/html
```

6. Buat file baru dalam direktori `html` yang diberi nama `SamplePage.php`, lalu edit file tersebut dengan menggunakan nano (atau editor pilihan Anda).

```
>SamplePage.php
nano SamplePage.php
```

7. Tambahkan konten berikut ini ke file `SamplePage.php`:

   MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php
```

```
  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
```

```
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}
```

```
/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

### PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
```

```
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
```

```
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
```

```
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;

}
?>
```

8. Simpan dan tutup file `SamplePage.php`.
9. Verifikasi bahwa server web Anda berhasil terhubung ke klaster DB Anda dengan membuka browser web dan menelusuri ke `http://`*`EC2 instance endpoint`*`/SamplePage.php`, misalnya: `http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php`.

Anda dapat menggunakan `SamplePage.php` untuk menambahkan data ke klaster DB Anda. Data yang Anda tambahkan kemudian ditampilkan di halaman. Untuk memverifikasi apakah data dimasukkan ke dalam tabel, instal klien MySQL pada instans Amazon EC2. Kemudian, hubungkan ke klaster DB dan kueri tabelnya.

Untuk informasi tentang menghubungkan ke klaster DB, lihat Menghubungkan ke klaster DB Amazon Aurora.

Untuk memastikan klaster DB Anda seaman mungkin, verifikasi bahwa sumber di luar VPC tidak dapat menghubungkan ke klaster DB Anda.

Setelah selesai menguji server web dan database, Anda harus menghapus klaster DB dan instans Amazon EC2 Anda.

- Untuk membuat klaster DB, ikuti petunjuk di Menghapus instans DB atau klaster DB Aurora. Anda tidak perlu membuat snapshot terakhir.
- Untuk mengakhiri instans Amazon EC2, ikuti petunjuk di Mengakhiri instans Anda dalam Panduan Pengguna Amazon EC2.

# tutorial Amazon Aurora dan kode sampel

AWS Dokumentasi mencakup beberapa tutorial yang memandu Anda melalui kasus penggunaan Amazon Aurora yang umum. Banyak dari tutorial ini menunjukkan cara menggunakan Aurora dengan AWS layanan lain. Selain itu, Anda dapat mengakses kode sampel di GitHub.

> **Note**
>
> Tutorial lainnya dapat dilihat di Blog Basis Data AWS. Untuk informasi tentang pelatihan, lihat Pelatihan dan Sertifikasi AWS.

Topik


- Tutorial dalam panduan ini
- Tutorial dalam AWS panduan lain
- AWS lokakarya dan portal konten lab untuk Aurora PostgreSQL
- AWS lokakarya dan portal konten lab untuk
- Tutorial dan kode sampel di GitHub
- Menggunakan layanan ini dengan AWS SDK


## Tutorial dalam panduan ini

Tutorial berikut dalam panduan ini menunjukkan cara melakukan tugas umum dengan Amazon Aurora:

- Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4

  Pelajari cara menyertakan klaster DB dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Dalam hal ini, VPC berbagi data dengan server web yang berjalan pada EC2 instance Amazon di VPC yang sama.
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

  Pelajari cara menyertakan klaster DB dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Dalam hal ini, VPC berbagi data dengan EC2 instance Amazon di VPC yang sama.

Dalam tutorial ini, Anda akan membuat VPC untuk skenario ini yang berfungsi dengan basis data yang berjalan dalam mode tumpukan ganda.

- Tutorial: Membuat server web dan klaster DB Amazon Aurora

  Pelajari cara menginstal server web Apache dengan PHP dan membuat basis data MySQL. Server web berjalan pada instans Amazon menggunakan Amazon Linux, dan database MySQL adalah EC2 instance MySQL DB cluster . Baik EC2 instans Amazon dan cluster DB berjalan di VPC Amazon.

- Tutorial: Memulihkan klaster DB Amazon Aurora dari snapshot klaster DB

  Pelajari cara memulihkan klaster DB dari snapshot klaster DB.

- Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

  Pelajari cara menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

- Tutorial: Log perubahan status instans DB menggunakan Amazon EventBridge

  Pelajari cara mencatat perubahan status instans DB menggunakan Amazon EventBridge dan AWS Lambda.

## Tutorial dalam AWS panduan lain

Tutorial berikut di AWS panduan lain menunjukkan kepada Anda cara melakukan tugas-tugas umum dengan Aurora:

> **Note**
>
> Beberapa tutorial menggunakan instans DB Amazon RDS, tetapi dapat disesuaikan untuk menggunakan klaster DB Aurora.

- Tutorial: Aurora Nirserver di Panduan Developer AWS AppSync

  Pelajari cara menggunakan sumber data AWS AppSync untuk menjalankan perintah SQL terhadap cluster Aurora Serverless DB dengan API Data diaktifkan. Anda dapat menggunakan resolver AWS AppSync untuk menjalankan pernyataan SQL terhadap Data API dengan kueri, mutasi, dan langganan GraphQL.

- Tutorial: Memutar Rahasia untuk AWS Database di AWS Secrets Manager Panduan Pengguna

Pelajari cara membuat rahasia untuk AWS database dan mengkonfigurasi rahasia untuk memutar pada jadwal. Anda memicu satu rotasi secara manual, kemudian mengonfirmasi bahwa versi baru rahasia terus memberikan akses.

- Tutorial dan sampel di Panduan Developer AWS Elastic Beanstalk

  Pelajari cara menerapkan aplikasi yang menggunakan database Amazon RDS. AWS Elastic Beanstalk

- Menggunakan Data dari Basis Data Amazon RDS untuk Membuat Sumber Data Amazon ML di Panduan Developer Amazon Machine Learning

  Pelajari cara membuat objek sumber data Amazon Machine Learning (Amazon ML) dari data yang disimpan dalam instans DB MySQL.

- Mengaktifkan Akses ke Instans Amazon RDS secara Manual di VPC di Panduan Pengguna Amazon QuickSight

  Pelajari cara mengaktifkan QuickSight akses ke instans Amazon RDS DB di VPC.

## AWS lokakarya dan portal konten lab untuk Aurora PostgreSQL

Kumpulan lokakarya dan konten praktis berikut ini membantu Anda memperoleh pemahaman tentang fitur dan kapabilitas Amazon Aurora PostgreSQL:

- Membuat Cluster Aurora Baru Secara Manual

  Pelajari cara membuat klaster Amazon Aurora PostgreSQL secara manual.

- Konfigurasikan Cloud9 dan Inisialisasi Database

  Pelajari cara mengonfigurasi Cloud9 dan menginisialisasi basis data PostgreSQL.

- Kloning Cepat

  Pelajari cara membuat kloning cepat Aurora.

- Manajemen Rencana Kueri

  Pelajari cara mengontrol rencana eksekusi untuk serangkaian pernyataan menggunakan manajemen rencana kueri.

- Manajemen Cache Klaster

Pelajari fitur Manajemen Cache Klaster di Aurora PostgreSQL.

- Streaming Aktivitas Basis Data

  Pelajari cara memantau dan mengaudit aktivitas basis data Anda dengan fitur ini.

- Menggunakan Wawasan Performa

  Pelajari cara memantau dan menyetel instans DB Anda menggunakan Wawasan performa.

- Pemantauan Performa dengan Alat RDS

  Pelajari cara menggunakan AWS dan alat Postgres (Cloudwatch, Enhanced Monitoring, Slow Query Logs, Performance Insights, PostgreSQL Catalog Views) untuk memahami masalah kinerja dan mengidentifikasi cara untuk meningkatkan kinerja database Anda.

- Penskalaan Otomatis Replika Baca

  Pelajari cara kerja penskalaan otomatis replika baca Aurora dalam praktiknya menggunakan skrip pembuat beban.

- Menguji Toleransi Kesalahan

  Pelajari bagaimana klaster DB dapat menoleransi kegagalan.

- Basis data global Aurora

  Pelajari tentang Basis Data Global Aurora.

- Menggunakan Machine Learning

  Pelajari Aurora Machine Learning.

- Aurora Nirserver v2

  Pelajari tentang Aurora Nirserver v2.

- Ekstensi Bahasa Tepercaya untuk Aurora PostgreSQL

  Pelajari cara membuat ekstensi berperforma tinggi yang berjalan dengan aman di Aurora PostgreSQL.

## AWS lokakarya dan portal konten lab untuk

Kumpulan lokakarya dan konten praktis berikut ini membantu Anda memperoleh pemahaman tentang fitur dan kapabilitas Amazon Aurora MySQL:

- Membuat Klaster Aurora

  Pelajari cara membuat klaster Amazon Aurora MySQL secara manual.

- Membuat lingkungan IDE berbasis Cloud Cloud9 untuk menghubungkan ke basis data

  Pelajari cara mengonfigurasi Cloud9 dan menginisialisasi basis data MySQL.

- Kloning Cepat

  Pelajari cara membuat kloning cepat Aurora.

- Melacak Mundur sebuah Klaster

  Pelajari cara melacak mundur klaster DB.

- Menggunakan Wawasan Performa

  Pelajari cara memantau dan menyetel instans DB Anda menggunakan Wawasan performa.

- Pemantauan Performa dengan Alat RDS

  Pelajari cara menggunakan AWS dan alat SQL untuk memahami masalah kinerja dan mengidentifikasi cara untuk meningkatkan kinerja database Anda.

- Menganalisis Performa Kueri

  Pelajari cara memecahkan masalah terkait performa SQL menggunakan alat lain.

- Penskalaan Otomatis Replika Baca

  Pelajari cara kerja penskalaan otomatis replika baca.

- Menguji Toleransi Kesalahan

  Pelajari tentang fitur toleransi kesalahan dan ketersediaan tinggi di Aurora MySQL.

- Basis data global Aurora

  Pelajari tentang Basis Data Global Aurora.

- Aurora Nirserver v2

  Pelajari tentang Aurora Nirserver v2.

- Menggunakan Machine Learning

  Pelajari Aurora Machine Learning.

# Tutorial dan kode sampel di GitHub

Tutorial dan kode contoh berikut GitHub menunjukkan kepada Anda cara melakukan tugas-tugas umum dengan Aurora:

- Membuat perpustakaan peminjaman Aurora Serverless v2

  Pelajari cara membuat aplikasi perpustakaan peminjaman tempat pengunjung dapat meminjam dan mengembalikan buku. Contohnya menggunakan Aurora Serverless v2 dan AWS SDK untuk Python (Boto3).
- Membuat aplikasi pelacak item Amazon Aurora dengan API REST Spring yang mengueri data Aurora Serverless v2 menggunakan SDK for Java 2.x

  Pelajari cara membuat API REST Spring yang mengueri data Aurora Serverless v2. Ini untuk penggunaan oleh aplikasi React menggunakan SDK for Java 2.x.
- Membuat aplikasi pelacak item Amazon Aurora yang menanyakan data menggunakan Aurora Serverless v2AWS SDK untuk PHP

  Pelajari cara membuat aplikasi yang menggunakan `RdsDataClient` API Data dan Aurora Serverless v2 untuk melacak serta melaporkan item pekerjaan. Contoh menggunakan AWS SDK untuk PHP.
- Membuat aplikasi pelacak item Amazon Aurora yang mengueri data Aurora Serverless v2 menggunakan AWS SDK untuk Python (Boto3)

  Pelajari cara membuat aplikasi yang menggunakan `RdsDataClient` API Data dan Aurora Serverless v2 untuk melacak serta melaporkan item pekerjaan. Contoh menggunakan AWS SDK untuk Python (Boto3).

# Menggunakan layanan ini dengan AWS SDK

AWS kit pengembangan perangkat lunak (SDKs) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

| Dokumentasi SDK | Contoh kode |
| --- | --- |
| AWS SDK untuk C++ | AWS SDK untuk C++ contoh kode |

| Dokumentasi SDK | Contoh kode |
| --- | --- |
| AWS CLI | AWS CLI contoh kode |
| AWS SDK untuk Go | AWS SDK untuk Go contoh kode |
| AWS SDK untuk Java | AWS SDK untuk Java contoh kode |
| AWS SDK untuk JavaScript | AWS SDK untuk JavaScript contoh kode |
| AWS SDK untuk Kotlin | AWS SDK untuk Kotlin contoh kode |
| AWS SDK untuk .NET | AWS SDK untuk .NET contoh kode |
| AWS SDK untuk PHP | AWS SDK untuk PHP contoh kode |
| Alat AWS untuk PowerShell | Alat AWS untuk PowerShell contoh kode |
| AWS SDK untuk Python (Boto3) | AWS SDK untuk Python (Boto3) contoh kode |
| AWS SDK untuk Ruby | AWS SDK untuk Ruby contoh kode |
| AWS SDK for Rust | AWS SDK for Rust contoh kode |
| AWS SDK untuk SAP ABAP | AWS SDK untuk SAP ABAP contoh kode |
| AWS SDK for Swift | AWS SDK for Swift contoh kode |

Untuk contoh khusus untuk layanan ini, lihat Contoh kode untuk Aurora menggunakan AWS SDKs.

Ketersediaan contoh

Tidak dapat menemukan apa yang Anda butuhkan? Minta contoh kode menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

# Akses terprogram ke Aurora

Amazon RDS memberi Anda alat berikut untuk mengelola sumber daya Aurora Anda secara terprogram.

AWS Command Line Interface (AWS CLI)

Anda dapat membuat dan mengelola sumber daya Amazon Aurora Anda dengan menggunakan shell AWS CLI baris perintah. AWS CLI Ini menyediakan akses langsung ke AWS layanan APIs for, seperti AmazonRDS. Untuk sintaks dan contoh perintah untuk AmazonRDS, lihat rdsdalam AWS CLI Command Reference.

AWS CloudFormation

Dengan alat AWS Infrastructure as Code (IAc) ini, Anda dapat membuat templat yang menjelaskan semua sumber daya Aurora Amazon yang Anda inginkan, AWS CloudFormation serta menyediakan serta mengonfigurasi sumber daya tersebut untuk Anda. Untuk informasi selengkapnya, lihat the section called “Menciptakan sumber daya dengan AWS CloudFormation”.

AWS kit pengembangan perangkat lunak () SDKs

AWS SDKsmenyediakan banyak teknologi populer dan bahasa pemrograman. Mereka memudahkan Anda untuk memanggil AWS layanan dari dalam aplikasi Anda dalam bahasa atau teknologi itu. Untuk informasi selengkapnya tentang iniSDKs, lihat Alat untuk mengembangkan dan mengelola aplikasi di AWS.

Amazon RDS API

Ini API adalah antarmuka tingkat protokol untuk Amazon. RDS Saat menggunakan iniAPI, Anda harus memformat setiap HTTPS permintaan dengan benar dan menambahkan tanda tangan digital yang valid ke setiap permintaan. Untuk informasi selengkapnya, lihat Referensi Amazon RDS API.

Konsol-ke-Kode

Dengan alat ini, Anda dapat menghasilkan kode untuk tindakan yang Anda lakukan di RDS konsol Amazon, dan menggunakan kode itu di alat lain seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat the section called “Konsol-ke-Kode”.

RDSData API

RDSData API (DataAPI) menyediakan akses terprogram ke data di database Amazon Aurora Tanpa Server Anda. Dengan DataAPI, Anda dapat menggunakan HTTP titik akhir yang

aman untuk menjalankan SQL pernyataan Anda tanpa mengelola koneksi. Untuk informasi selengkapnya, lihat Menggunakan Amazon RDS Data API.

# Gunakan Console-to-Code untuk menghasilkan kode untuk tindakan konsol Amazon Aurora Anda

Konsol tersebut menyediakan jalur dengan panduan untuk membuat sumber daya dan menguji prototipe. Jika Anda ingin membuat sumber daya yang sama dalam skala besar, Anda memerlukan kode otomatisasi. Console-to-Codeadalah fitur Amazon Q Developer yang dapat membantu Anda memulai dengan kode otomatisasi Anda. Console-to-Codemerekam tindakan konsol Anda, termasuk nilai default dan nilai parameter yang Anda berikan. Kemudian menggunakan AI generatif untuk menyarankan kode dalam bahasa dan format pilihan Anda untuk tindakan yang Anda pilih. Karena alur kerja konsol memastikan nilai parameter yang Anda tentukan valid bersama-sama, kode yang Anda hasilkan dengan menggunakan Console-to-Code memiliki nilai parameter yang kompatibel. Anda dapat menggunakan kode sebagai titik awal, dan kemudian menyesuaikannya untuk membuatnya siap produksi untuk kasus penggunaan spesifik Anda.

Misalnya, dengan Console-to-Code, Anda dapat merekam pembuatan cluster Aurora DB dan memilih untuk menghasilkan kode dalam AWS CloudFormation JSON format. Kemudian, Anda dapat menyalin kode itu dan menyesuaikannya untuk digunakan dalam AWS CloudFormation template Anda.

Console-to-Code saat ini dapat menghasilkan infrastructure-as-code (IAc) dalam bahasa dan format berikut:

- CDKJawa
- CDKPython
- CDK TypeScript
- CloudFormation JSON
- CloudFormation YAML

Untuk informasi selengkapnya dan petunjuk tentang cara menggunakan Console-to-Code, lihat Mengotomatisasi AWS layanan dengan Pengembang Amazon Q Console-to-Code di Panduan Pengguna Pengembang Amazon Q.

# Menyiapkan cluster DB Amazon Aurora

Bagian ini menunjukkan cara mengatur klaster DB Aurora Anda. Sebelum membuat klaster DB Aurora, tentukan kelas instans DB yang akan menjalankan klaster DB. Juga, tentukan di mana cluster DB akan berjalan dengan memilih AWS Region. Selanjutnya, buat klaster DB. Jika Anda memiliki data di luar Aurora, Anda dapat memigrasikan data ke dalam klaster DB Aurora.

Topik


- Membuat klaster DB Amazon Aurora
- Membuat sumber Amazon Amazon Aurora dengan AWS CloudFormation
- Menghubungkan ke klaster DB Amazon Aurora
-
- Memigrasikan data ke klaster DB Amazon Aurora
- Membuat ElastiCache cache Amazon menggunakan pengaturan instans Amazon DB cluster Aurora RDS
- EC2AWS Database Migration Service
- Tutorial: Membuat cluster dengan kustom dan grup opsi baru

# Membuat klaster DB Amazon Aurora

Klaster DB Amazon Aurora terdiri dari instans DB, yang kompatibel dengan MySQL atau PostgreSQL, dan volume klaster yang menyimpan data untuk klaster DB, yang disalin ke tiga Zona Ketersediaan sebagai satu volume virtual. Secara default, klaster Aurora DB berisi instans DB primer yang melakukan pembacaan dan penulisan, dan, secara opsional, hingga 15 Replika Aurora (instans DB pembaca). Untuk informasi selengkapnya tentang klaster Aurora DB, lihat Klaster DB Amazon Aurora.

Aurora memiliki dua jenis utama klaster DB:

- Terprovisi Aurora – Anda memilih kelas instans DB untuk instans penulis dan pembaca berdasarkan beban kerja yang Anda harapkan. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB. Terprovisi Aurora memiliki beberapa opsi, termasuk basis data global Aurora. Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.
- Aurora Serverless— Aurora Serverless v2 adalah konfigurasi penskalaan otomatis sesuai permintaan untuk Aurora. Kapasitas disesuaikan secara otomatis berdasarkan permintaan aplikasi. Anda hanya dikenai biaya untuk sumber daya yang menggunakan klaster DB Anda. Otomatisasi ini sangat berguna untuk lingkungan dengan beban kerja yang sangat bervariasi dan tidak dapat diprediksi. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2.

Di bagian berikut ini, Anda dapat mengetahui cara membuat klaster Aurora DB. Untuk memulai, pertama-tama lihat Prasyarat klaster DB.

Untuk petunjuk tentang menghubungkan ke klaster Aurora DB Anda, lihat Menghubungkan ke klaster DB Amazon Aurora.

Daftar Isi


- Prasyarat klaster DB
  - Konfigurasikan jaringan untuk klaster DB
    - Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance
    - Konfigurasikan jaringan secara manual
  - Prasyarat tambahan
- Membuat klaster DB
  - Membuat instance DB primer (penulis)
- Pengaturan untuk klaster Aurora DB

- Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora
- Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

## Prasyarat klaster DB

> **Important**
>
> Sebelum dapat membuat klaster Aurora DB, Anda harus menyelesaikan tugas-tugas dalam Menyiapkan lingkungan Anda untuk Amazon Aurora.

Berikut ini adalah prasyarat yang harus diselesaikan sebelum membuat klaster DB.

Topik

- Konfigurasikan jaringan untuk klaster DB
- Prasyarat tambahan

### Konfigurasikan jaringan untuk klaster DB

Anda dapat membuat cluster Amazon Aurora DB hanya di cloud pribadi virtual (VPC) berdasarkan layanan VPC Amazon, di Wilayah AWS yang memiliki setidaknya dua Availability Zone. Grup subnet DB yang Anda pilih untuk klaster DB harus mencakup setidaknya dua Zona Ketersediaan. Konfigurasi ini memastikan bahwa klaster DB Anda selalu memiliki setidaknya satu instans DB tersedia untuk failover jika terjadi kegagalan Zona Ketersediaan.

Jika Anda berencana untuk mengatur konektivitas antara cluster DB baru Anda dan EC2 instance di VPC yang sama, Anda dapat melakukannya selama pembuatan cluster DB. Jika Anda berencana untuk terhubung ke cluster DB Anda dari sumber daya selain EC2 instance di VPC yang sama, Anda dapat mengonfigurasi koneksi jaringan secara manual.

Topik

- Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance
- Konfigurasikan jaringan secara manual

Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance

Saat membuat klaster Aurora DB, Anda dapat menggunakannya AWS Management Console untuk mengatur konektivitas antara EC2 instans Amazon dan cluster DB baru. Ketika Anda melakukannya, RDS mengonfigurasi VPC dan pengaturan jaringan Anda secara otomatis. Cluster DB dibuat dalam VPC yang sama dengan EC2 instance sehingga EC2 instance dapat mengakses cluster DB.

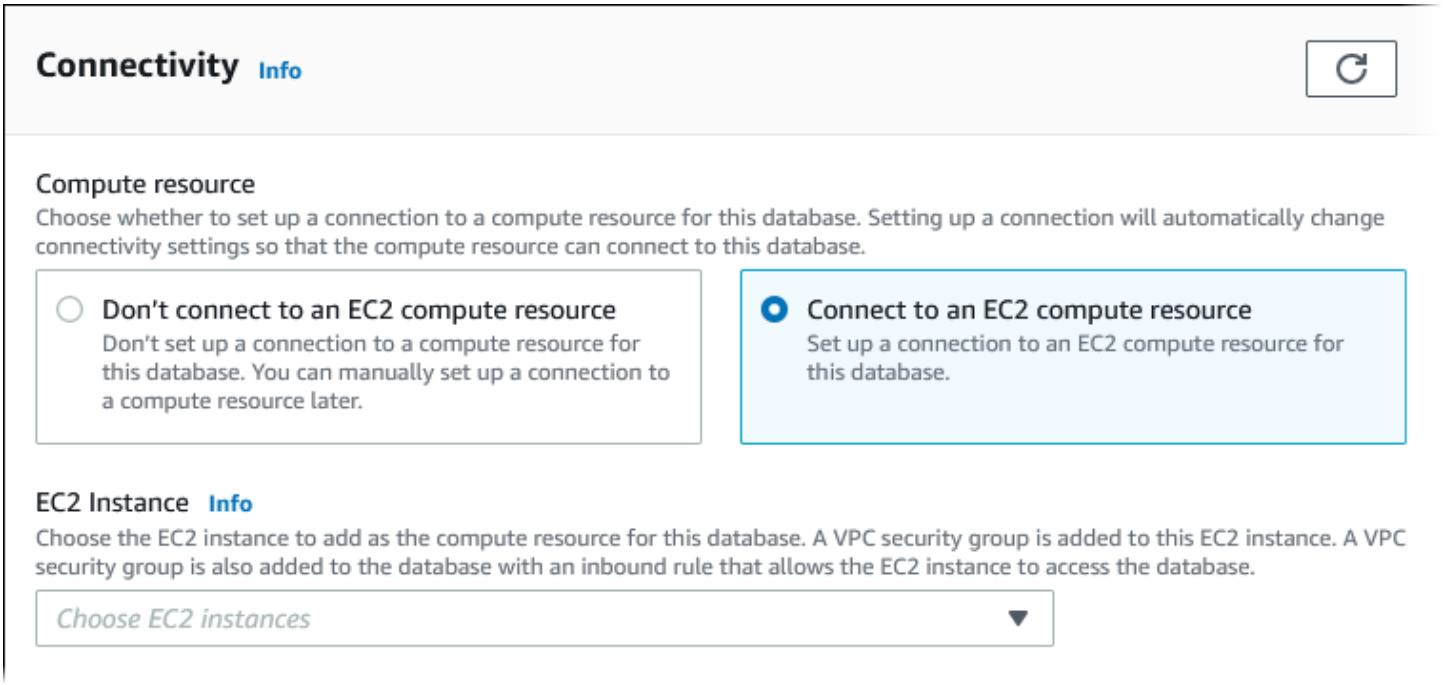
Berikut ini adalah persyaratan untuk menghubungkan EC2 instance dengan cluster DB:

- EC2 Instance harus ada di Wilayah AWS sebelum Anda membuat cluster DB.

  Jika tidak ada EC2 instance di Wilayah AWS, konsol menyediakan tautan untuk membuatnya.
- Saat ini, klaster DB tidak dapat berupa klaster DB Aurora Serverless atau bagian dari basis data global Aurora.
- Pengguna yang membuat instans DB harus memiliki izin untuk melakukan operasi berikut:
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Menggunakan opsi ini membuat klaster DB privat. Klaster DB menggunakan grup subnet DB hanya dengan subnet privat untuk membatasi akses ke sumber daya dalam VPC.

Untuk menghubungkan EC2 instance ke cluster DB, pilih Connect to an EC2 compute resource di bagian Connectivity pada halaman Create database.




Bila Anda memilih Connect to an EC2 compute resource, RDS akan menyetel opsi berikut secara otomatis. Anda tidak dapat mengubah pengaturan ini kecuali Anda memilih untuk tidak mengatur konektivitas dengan EC2 instans dengan memilih Jangan sambungkan ke sumber daya EC2 komputasi.

| Opsi konsol | Pengaturan otomatis |
| --- | --- |
| Jenis jaringan | RDS menetapkan jenis jaringan ke IPv4. Saat ini, mode dual-stack tidak didukung saat Anda mengatur koneksi antara EC2 instance dan cluster DB. |
| Cloud Privat Virtual (VPC) | RDS menetapkan VPC ke yang terkait dengan EC2 instance. |
| Grup subnet DB | RDS memerlukan grup subnet DB dengan subnet pribadi di Availability Zone yang sama dengan instance. EC2 Jika grup subnet DB yang memenuhi persyaratan ini ada, RDS menggunakan grup subnet DB yang ada. Secara default, opsi ini diatur ke Pengaturan otomatis.<br><br>Ketika Anda memilih Pengaturan otomatis dan tidak ada grup subnet DB yang memenuhi persyaratan ini, tindakan berikut terjadi. RDS menggunakan tiga subnet pribadi yang tersedia di |

| Opsi konsol | Pengaturan otomatis |
| --- | --- |
| | tiga Availability Zone di mana salah satu Availability Zones sama dengan instance. EC2 Jika subnet privat tidak tersedia di Zona Ketersediaan, RDS membuat subnet privat di Zona Ketersedi aan. Kemudian RDS membuat grup subnet DB.<br><br>Ketika subnet privat tersedia, RDS menggunakan tabel rute yang terkait dengan subnet dan menambahkan subnet apa pun yang dibuatnya ke tabel rute ini. Ketika tidak ada subnet privat yang tersedia, RDS membuat tabel rute tanpa akses gateway internet dan menambahkan subnet yang dibuatnya ke tabel rute.<br><br>RDS juga memungkinkan Anda untuk menggunakan grup subnet DB yang ada. Pilih Pilih yang ada jika Anda ingin menggunakan grup subnet DB pilihan Anda yang sudah ada. |
| Akses publik | RDS memilih Tidak sehingga klaster DB tidak dapat diakses publik.<br><br>Untuk keamanan, ini adalah praktik terbaik untuk menjaga basis data tetap privat dan memastikannya tidak dapat diakses dari internet. |

| Opsi konsol | Pengaturan otomatis |
| --- | --- |
| Grup keamanan VPC (firewall) | RDS membuat grup keamanan baru yang terkait dengan klaster DB. Grup keamanan diberi nama `rds-ec2-`*n*, dengan *n* berupa angka. Grup keamanan ini menyertakan aturan masuk dengan grup keamanan EC2 VPC (firewall) sebagai sumbernya. Grup keamanan ini yang terkait dengan cluster DB memungkinkan EC2 instance untuk mengakses cluster DB.<br><br>RDS juga membuat grup keamanan baru yang terkait dengan EC2 instance. Grup keamanan diberi nama `ec2-rds-`*n*, dengan *n* berupa angka. Grup keamanan ini mencakup aturan keluar dengan grup keamanan VPC dari klaster DB sebagai sumbernya. Grup keamanan ini memungkinkan EC2 instance untuk mengirim lalu lintas ke cluster DB.<br><br>Anda dapat menambahkan grup keamanan baru lainnya dengan memilih Buat baru dan mengetik nama grup keamanan baru.<br><br>Anda dapat menambahkan grup keamanan yang ada dengan memilih Pilih yang ada dan memilih grup keamanan untuk ditambahkan. |
| Zona Ketersediaan | Jika Anda tidak membuat Replika Aurora dalam Ketersediaan & daya tahan selama pembuatan cluster DB (penerapan Single-AZ ), RDS memilih Availability Zone instance. EC2<br><br>Saat Anda membuat Replika Aurora selama pembuatan cluster DB (penerapan Multi-AZ), RDS memilih Availability Zone EC2 instance untuk satu instans DB di cluster DB. RDS secara acak memilih Zona Ketersediaan yang berbeda untuk instans DB lainnya di klaster DB. Entah instans DB primer atau Aurora Replica dibuat di Availability Zone yang sama dengan instance. EC2 Ada kemungkinan biaya lintas Availability Zone jika instans dan EC2 instance DB utama berada di Availability Zone yang berbeda. |

Untuk informasi selengkapnya tentang pengaturan ini, lihat Pengaturan untuk klaster Aurora DB.

Jika Anda membuat perubahan apa pun pada pengaturan ini setelah cluster DB dibuat, perubahan tersebut dapat memengaruhi koneksi antara EC2 instance dan cluster DB.

Konfigurasikan jaringan secara manual

Jika Anda berencana untuk terhubung ke cluster DB Anda dari sumber daya selain EC2 instance di VPC yang sama, Anda dapat mengonfigurasi koneksi jaringan secara manual. Jika Anda menggunakan AWS Management Console untuk membuat klaster DB, Anda dapat meminta Amazon RDS untuk membuat VPC secara otomatis. Atau Anda dapat menggunakan VPC yang ada atau membuat VPC baru untuk klaster Aurora DB Anda. Pendekatan apa pun yang Anda ambil, VPC Anda harus memiliki setidaknya satu subnet di masing-masing dari setidaknya dua Zona Ketersediaan untuk digunakan dengan klaster DB Amazon Aurora.

Secara default, Amazon RDS membuat instans DB primer dan Replika Aurora di Zona Ketersediaan secara otomatis untuk Anda. Untuk memilih Zona Ketersediaan tertentu, Anda perlu mengubah pengaturan deployment Multi-AZ Ketersediaan & durabilitas menjadi Jangan buat Replika Aurora. Tindakan ini akan membuka pengaturan Zona Ketersediaan yang memungkinkan Anda memilih dari beberapa Zona Ketersediaan di VPC Anda. Namun, kami sangat menyarankan agar Anda mempertahankan pengaturan default serta membiarkan Amazon RDS membuat deployment Multi-AZ dan memilih Zona Ketersediaan untuk Anda. Dengan demikian, klaster Aurora DB Anda akan dibuat dengan failover cepat dan fitur ketersediaan tinggi yang merupakan dua manfaat utama Aurora.

Jika Anda tidak memiliki VPC default atau Anda belum membuat VPC, Anda dapat meminta Amazon RDS untuk membuat VPC secara otomatis ketika Anda membuat klaster DB menggunakan konsol. Jika tidak, Anda harus melakukan hal berikut:

- Buat VPC dengan setidaknya satu subnet di masing-masing setidaknya dua Availability Zones di Wilayah AWS mana Anda ingin menyebarkan cluster DB Anda. Untuk informasi selengkapnya, lihat Bekerja dengan klaster DB dalam VPC dan Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .
- Tentukan grup keamanan VPC yang mengizinkan koneksi ke klaster DB Anda. Untuk informasi selengkapnya, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan dan Mengontrol akses dengan grup keamanan.
- Tentukan grup subnet RDS yang mendefinisikan setidaknya dua subnet di VPC yang dapat digunakan oleh klaster DB. Untuk informasi selengkapnya, lihat Menggunakan grup subnet DB.

Untuk informasi tentang VPCs, lihatAmazon VPC dan Aurora. Untuk tutorial yang mengonfigurasi jaringan untuk klaster DB privat, lihat Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .

Jika Anda ingin terhubung ke sumber daya yang tidak berada di VPC yang sama dengan klaster Aurora DB, lihat skenario yang sesuai dalam Skenario untuk mengakses cluster DB di VPC.

### Prasyarat tambahan

Sebelum Anda membuat klaster DB Anda, pertimbangkan prasyarat tambahan berikut:

- Jika Anda terhubung ke kredensyal AWS menggunakan AWS Identity and Access Management (IAM), AWS akun Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk melakukan operasi Amazon RDS. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.

  Jika Anda menggunakan IAM untuk mengakses konsol Amazon RDS, Anda harus terlebih dahulu masuk ke AWS Management Console dengan kredensyal pengguna Anda. Lalu pergi ke konsol Amazon RDS di https://console.aws.amazon.com/rds/.
- Jika Anda ingin menyesuaikan parameter konfigurasi untuk klaster DB, Anda harus menentukan grup parameter klaster DB dan grup parameter DB dengan pengaturan parameter yang diperlukan. Untuk informasi tentang membuat atau memodifikasi grup parameter klaster DB atau grup parameter DB, lihat .
- Tentukan nomor port TCP/IP yang akan ditentukan untuk klaster DB Anda. Firewall di beberapa perusahaan memblokir koneksi ke port default (3306 untuk MySQL, 5432 untuk PostgreSQL) untuk Aurora. Jika firewall perusahaan Anda memblokir port default, pilih port lain untuk klaster DB Anda. Semua instans dalam klaster DB menggunakan port yang sama.
- Jika versi mesin utama untuk database Anda telah mencapai akhir RDS dari tanggal dukungan standar, Anda harus menggunakan opsi Extended Support CLI atau parameter RDS API. Untuk informasi selengkapnya, lihat RDS Extended Support diPengaturan untuk klaster Aurora DB.

## Membuat klaster DB

Anda dapat membuat cluster Aurora DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Anda dapat membuat cluster DB menggunakan AWS Management Console dengan Easy create diaktifkan atau tidak diaktifkan. Dengan Pembuatan mudah aktif, Anda hanya menentukan jenis mesin DB, ukuran instans DB, dan pengidentifikasi instans DB. Pembuatan mudah menggunakan pengaturan default untuk opsi konfigurasi lainnya. Dengan Pembuatan mudah tidak aktif, Anda menentukan lebih banyak opsi konfigurasi saat Anda membuat basis data, termasuk opsi untuk ketersediaan, keamanan, pencadangan, dan pemeliharaan.

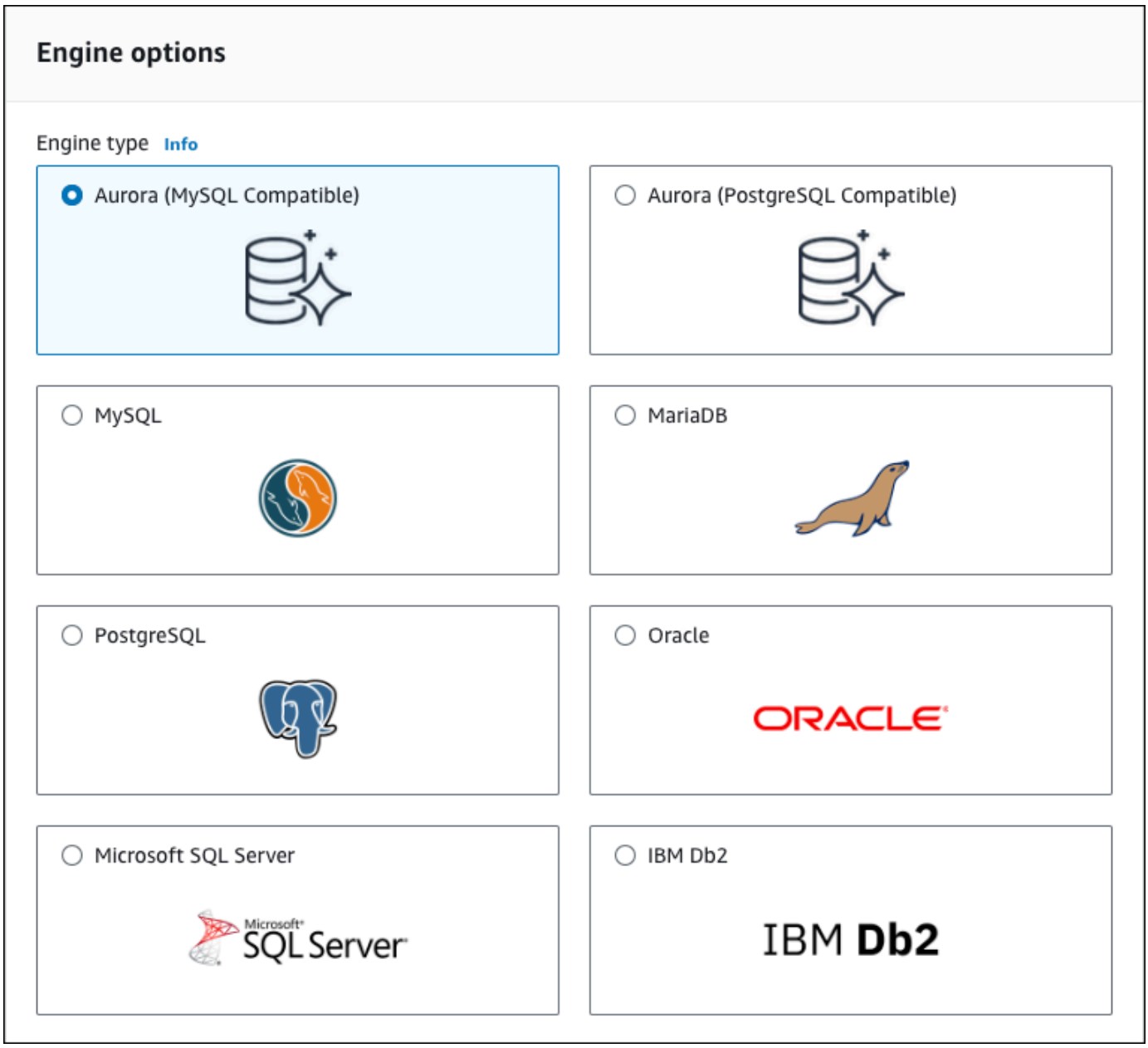
> **Note**
>
> Untuk contoh ini, Pembuatan standar aktif, dan Pembuatan mudah tidak diaktifkan. Untuk informasi tentang membuat cluster DB dengan Easy create diaktifkan, lihatMemulai dengan Amazon Aurora.

Untuk membuat klaster Aurora DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih AWS Wilayah di mana Anda ingin membuat cluster DB.

   Aurora tidak tersedia di semua AWS Wilayah. Untuk daftar AWS Wilayah tempat Aurora tersedia, lihat. Ketersediaan wilayah
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.
5. Untuk Pilih metode pembuatan basis data, pilih Pembuatan standar.
6. Untuk Jenis engine, pilih salah satu dari berikut ini:

   - Aurora (Kompatibel dengan MySQL)
   - Aurora (Kompatibel dengan PostgreSQL)




7. Pilih Versi mesin.

   Untuk informasi selengkapnya, lihat Versi Amazon Aurora. Anda dapat menggunakan filter untuk memilih versi yang kompatibel dengan fitur yang Anda inginkan, seperti Aurora Serverless v2. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2.

8. Di Templat, pilih templat yang sesuai dengan kasus penggunaan Anda.

9. Untuk memasukkan kata sandi master, lakukan hal berikut:

   a. Di bagian Pengaturan, perluas Pengaturan Kredensial.

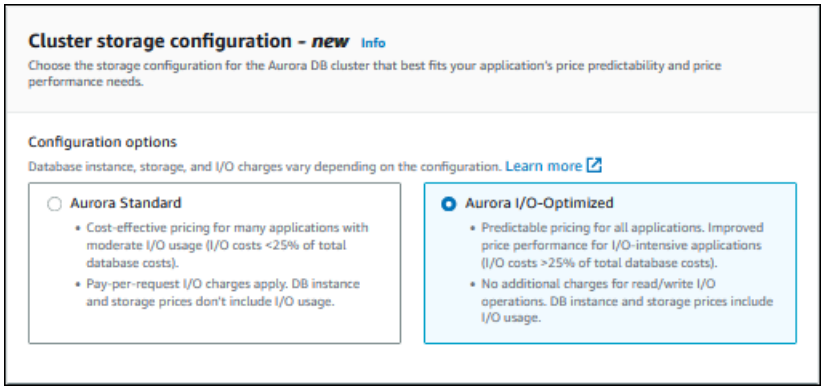
b. Hapus kotak centang Buat otomatis kata sandi.

c. (Opsional) Ubah nilai Nama pengguna master dan masukkan kata sandi yang sama di Kata sandi master dan Konfirmasikan kata sandi.

Secara default, instans DB baru menggunakan kata sandi yang dibuat secara otomatis untuk pengguna utama.

10. Di bagian Konektivitas di bagian Grup keamanan VPC (firewall), jika Anda memilih Buat baru, grup keamanan VPC dibuat dengan aturan masuk yang memungkinkan alamat IP komputer lokal Anda mengakses basis data.

11. Untuk Konfigurasi penyimpanan klaster, pilih Aurora I/O-Optimized atau Aurora Standard. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.




12. (Opsional) Atur koneksi ke sumber daya komputasi untuk klaster DB ini.

Anda dapat mengonfigurasi konektivitas antara EC2 instans Amazon dan cluster DB baru selama pembuatan cluster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance.

13. Untuk bagian yang lainnya, tentukan pengaturan klaster DB Anda. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

14. Pilih Buat basis data.

Jika Anda memilih untuk menggunakan kata sandi yang dibuat otomatis, tombol Lihat detail kredensial muncul pada halaman Basis data.

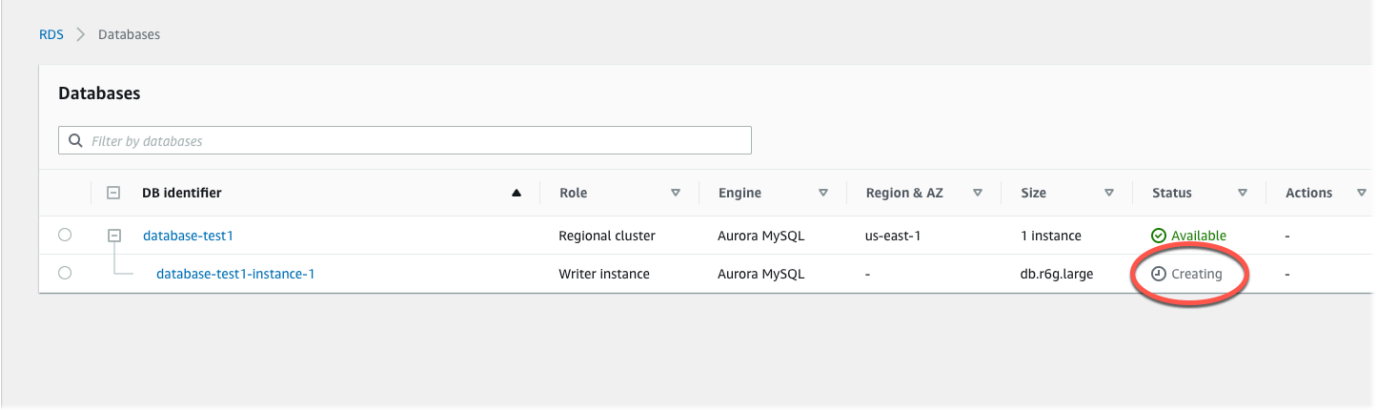
Untuk melihat nama pengguna dan kata sandi master klaster DB, pilih Lihat detail kredensial.

Untuk terhubung ke instans DB sebagai pengguna master, gunakan nama pengguna dan kata sandi yang muncul.

> **Important**
>
> Anda tidak dapat melihat kata sandi pengguna master lagi. Jika Anda tidak mencatatnya, Anda mungkin harus mengubahnya. Jika perlu mengubah kata sandi pengguna utama setelah instans DB tersedia, Anda dapat mengubah instans DB untuk melakukannya. Untuk informasi selengkapnya tentang cara memodifikasi instans DB, lihat Memodifikasi klaster DB Amazon Aurora.

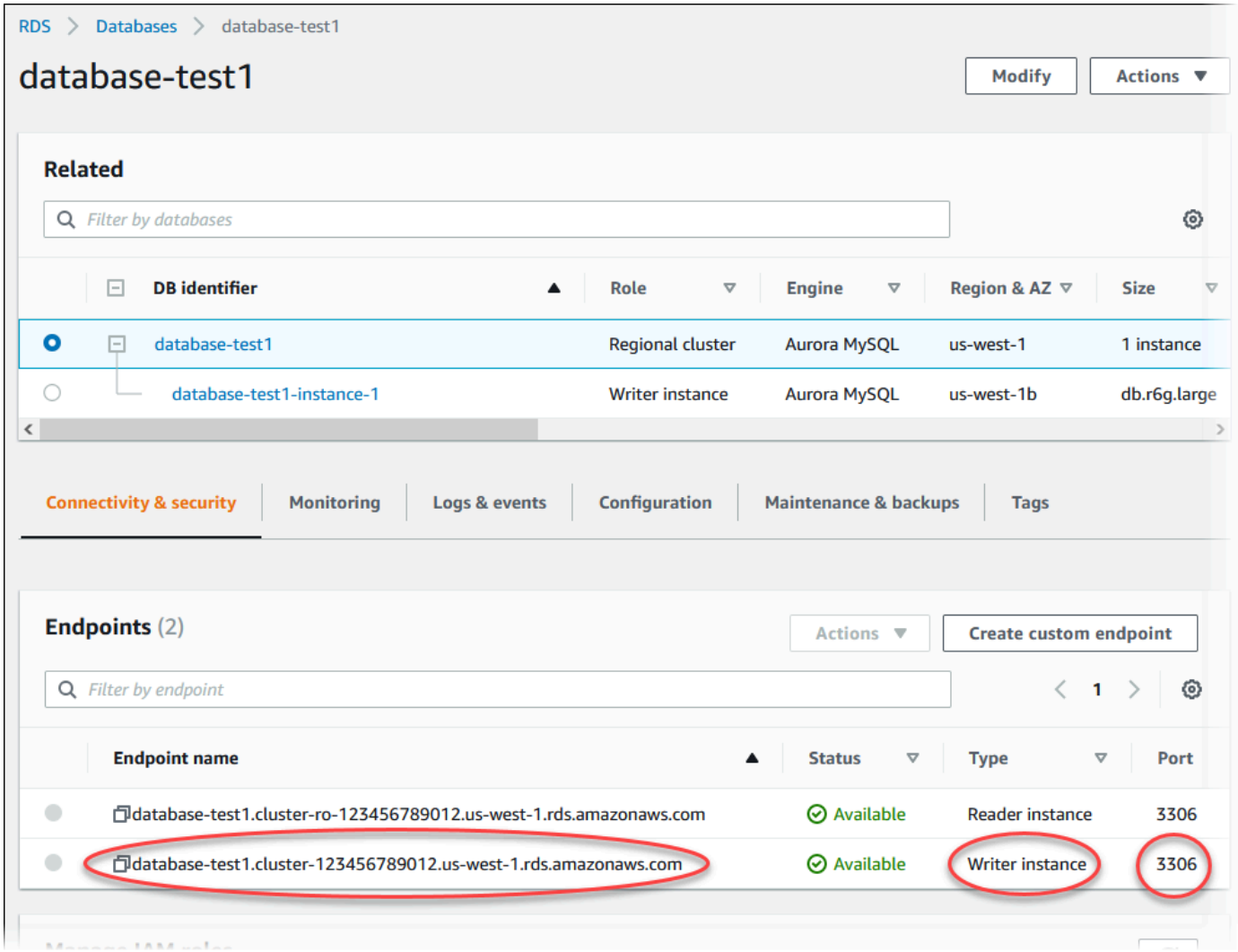
15. Untuk Basis data, pilih nama klaster Aurora DB baru.

    Pada konsol RDS, detail untuk klaster DB baru akan muncul. Klaster DB dan instans DB-nya memiliki status membuat hingga klaster DB siap digunakan.




    Saat statusnya berubah menjadi tersedia untuk keduanya, Anda dapat terhubung ke klaster DB. Tergantung pada kelas instans DB dan jumlah penyimpanan, diperlukan waktu hingga 20 menit sebelum klaster DB baru tersedia.

    Untuk melihat klaster yang baru dibuat, pilih Basis data dari panel navigasi di konsol Amazon RDS. Lalu pilih klaster DB untuk menampilkan detail klaster DB. Untuk informasi selengkapnya, lihat Melihat klaster DB Amazon Aurora.




Pada tab Koneksi & keamanan, catat port dan titik akhir instans DB penulis. Gunakan titik akhir dan port klaster dalam string koneksi JDBC dan ODBC Anda untuk aplikasi apa pun yang melakukan operasi tulis atau baca.

AWS CLI

> **Note**
>
> Sebelum Anda dapat membuat cluster Aurora DB menggunakan AWS CLI, Anda harus memenuhi prasyarat yang diperlukan, seperti membuat VPC dan grup subnet RDS DB. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Anda dapat menggunakan AWS CLI untuk membuat cluster Aurora MySQL DB atau cluster Aurora PostgreSQL DB.

Untuk membuat cluster DB MySQL Aurora menggunakan AWS CLI

Saat Anda membuat klaster DB atau instans DB yang kompatibel dengan Aurora MySQL 8.0 atau 5.7, Anda menentukan `aurora-mysql` untuk opsi `--engine`.

Selesaikan langkah-langkah berikut:

1. Identifikasi grup subnet DB dan ID grup keamanan VPC untuk cluster DB baru Anda, lalu panggil create-db-cluster AWS CLI perintah untuk membuat cluster DB MySQL Aurora.

   Misalnya, perintah berikut membuat klaster DB yang kompatibel dengan MySQL 8.0 bernama `sample-cluster`. Klaster ini menggunakan versi mesin default dan jenis penyimpanan Aurora I/O-Optimized.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-mysql --engine-version 8.0 \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-mysql --engine-version 8.0 ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Perintah berikut membuat klaster DB yang kompatibel dengan MySQL 5.7 bernama `sample-cluster`. Klaster ini menggunakan versi mesin default dan jenis penyimpanan Aurora Standard.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
       --engine aurora-mysql --engine-version 5.7 \
       --storage-type aurora \
   ```

```
    --master-username user-name --manage-master-user-password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. Jika Anda menggunakan konsol untuk membuat klaster DB, Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

   Panggil create-db-instance AWS CLI perintah untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai nilai opsi `--db-cluster-identifier`.

   > **Note**
   >
   > Anda tidak dapat mengatur opsi `--storage-type` untuk instans DB. Anda dapat mengaturnya hanya untuk klaster DB.

   Misalnya, perintah berikut membuat instans DB yang kompatibel dengan MySQL 5.7 atau MySQL 8.0 bernama `sample-instance`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
   class db.r5.large
   ```

   Untuk Windows:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance ^
   ```

```
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

Untuk membuat cluster DB PostgreSQL Aurora menggunakan AWS CLI

1. Identifikasi grup subnet DB dan ID grup keamanan VPC untuk cluster DB baru Anda, lalu panggil create-db-cluster AWS CLI perintah untuk membuat cluster Aurora PostgreSQL DB.

   Misalnya, perintah berikut membuat klaster DB baru bernama `sample-cluster`. Klaster ini menggunakan versi mesin default dan jenis penyimpanan Aurora I/O-Optimized.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-postgresql \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-postgresql ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

2. Jika Anda menggunakan konsol untuk membuat klaster DB, Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

   Panggil create-db-instance AWS CLI perintah untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai nilai opsi `--db-cluster-identifier`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
   ```

```
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

API RDS

> **Note**
>
> Sebelum Anda dapat membuat cluster Aurora DB menggunakan AWS CLI, Anda harus memenuhi prasyarat yang diperlukan, seperti membuat VPC dan grup subnet RDS DB. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Identifikasi grup subnet DB dan ID grup keamanan VPC untuk cluster DB baru Anda, lalu panggil operasi DBCluster Create untuk membuat cluster DB.

Saat Anda membuat klaster DB atau instans DB Aurora MySQL versi 2 atau 3, tentukan `aurora-mysql` untuk parameter `Engine`.

Saat Anda membuat klaster DB atau instans DB Aurora PostgreSQL, tentukan `aurora-postgresql` untuk parameter `Engine`.

Jika Anda menggunakan konsol untuk membuat klaster DB, Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan RDS API untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda menggunakan Create. DBInstance Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

### Membuat instance DB primer (penulis)

Jika Anda menggunakan AWS Management Console untuk membuat cluster DB, Amazon RDS secara otomatis membuat instance utama (penulis) untuk cluster DB Anda. Jika Anda menggunakan AWS CLI atau RDS API untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

Untuk informasi selengkapnya, lihat Membuat klaster DB.

> **Note**
>
> Jika Anda memiliki cluster DB tanpa instance DB penulis, juga disebut cluster tanpa kepala, Anda tidak dapat menggunakan konsol untuk membuat instance penulis. Anda harus menggunakan AWS CLI atau RDS API.

Contoh berikut menggunakan create-db-instance AWS CLI perintah untuk membuat instance penulis untuk klaster Aurora PostgreSQL DB bernama. `headless-test`

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

## Pengaturan untuk klaster Aurora DB

Tabel berikut berisi detail pengaturan yang Anda pilih saat membuat klaster Aurora DB.

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
|---|---|---|
| Peningkatan versi minor otomatis | Pilih Aktifkan peningkatan versi minor otomatis jika Anda ingin klaster Aurora DB Anda menerima peningkatan versi minor pilihan di mesin DB secara otomatis saat tersedia. | Atur nilai ini untuk setiap instans DB di klaster Aurora Anda. Jika ada instans DB di klaster Anda yang menonaktifkan pengaturan ini, klaster tersebut tidak ditingkat kan secara otomatis. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| | Pengaturan Peningkatan versi minor otomatis berlaku untuk klaster DB Aurora PostgreSQL dan Aurora MySQL.<br><br>Untuk informasi selengkap nya tentang pembaruan mesin untuk Aurora PostgreSQL, lihat Pembaruan mesin database untuk Amazon Aurora Postgre SQL.<br><br>Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin database untuk Amazon Aurora My SQL. | Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--auto-minor-version-upgrade\|--no-auto-minor-version-upgrade` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `AutoMinorVersionUpgrade` . |
| AWS KMS key | Hanya tersedia jika Enkripsi diatur ke Aktifkan enkripsi. Pilih AWS KMS key yang akan digunakan untuk mengenkripsi klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--kms-key-id` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `KmsKeyId`. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Backtrack | Hanya berlaku untuk Aurora MySQL. Pilih Aktifkan Backtrack untuk mengaktifkan backtrack atau Nonaktifkan Backtrack untuk menonaktifkan backtrack. Dengan backtrack, Anda dapat memundurkan klaster DB ke waktu tertentu, tanpa membuat klaster DB baru. Ini dinonaktifkan secara default. Jika Anda mengaktifkan backtracking, tentukan juga jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking klaster DB Anda (jendela backtrack target). Untuk informasi selengkapnya, lihat Melakukan backtracking klaster DB Aurora. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--backtrack-window` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `BacktrackWindow` . |
| Otoritas sertifikat | Otoritas sertifikat (CA) untuk sertifikat server yang digunakan oleh instans DB di klaster DB.<br><br>Untuk informasi selengkapnya, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke . | Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--ca-certificate-identifier` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `CACertificateIdentifier` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Konfigurasi penyimpanan klaster | Jenis penyimpanan untuk klaster DB: Aurora I/O-Optimized atau Aurora Standard.<br><br>Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--storage-type` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `StorageType` . |
| Salin tanda ke snapshot | Pilih opsi ini untuk menyalin tag instans DB apa pun ke snapshot DB saat Anda membuat snapshot.<br><br>Untuk informasi selengkapnya, lihat Menandai sumber daya Amazon Aurora dan Amazon RDS. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--copy-tags-to-snapshot \| --no-copy-tags-to-snapshot` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `CopyTagsToSnapshot` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Autentikasi basis data | Autentikasi basis data yang ingin Anda gunakan.<br><br>Untuk MySQL:<br><br>• Pilih Autentikasi kata sandi untuk mengautentikasi pengguna basis data dengan kata sandi basis data saja.<br>• Pilih Kata sandi dan autentikasi basis data IAM untuk mengautentikasi pengguna basis data dengan kata sandi basis data dan kredensial pengguna melalui pengguna dan peran IAM. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.<br><br>Untuk PostgreSQL:<br><br>• Pilih Autentikasi basis data IAM untuk mengautentikasi pengguna basis data dengan kata sandi basis data dan kredensial pengguna melalui pengguna dan peran. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.<br>• Pilih Autentikasi Kerberos untuk mengautentikasi kata sandi basis data dan kredensial pengguna menggunakan autentikasi Kerberos. Untuk informasi selengkapnya, lihat | Untuk menggunakan otentikasi database IAM dengan AWS CLI, jalankan `create-db-cluster` dan atur opsi. `--enable-iam-database-authentication \| --no-enable-iam-database-authentication`<br><br>Untuk menggunakan autentikasi basis data IAM dengan API RDS, panggil `CreateDBCluster` dan atur parameter `EnableIAMDatabaseAuthentication` .<br><br>Untuk menggunakan otentikasi Kerberos dengan AWS CLI, jalankan `create-db-cluster` dan atur opsi dan. `--domain --domain-iam-role-name`<br><br>Untuk menggunakan autentikasi Kerberos dengan API RDS, panggil `CreateDBCluster` dan atur parameter `Domain` dan `DomainIAMRoleName` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| | Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL. | |
| Port basis data | Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Klaster DB Aurora MySQL diatur secara default ke port MySQL default, 3306, dan klaster DB Aurora PostgreSQL diatur secara default ke port PostgreSQL default, 5432. Firewall di beberapa perusahaan memblokir koneksi ke port default tersebut. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--port` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `Port`. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Pengidentifikasi klaster DB | Masukkan nama untuk cluster DB Anda yang unik untuk akun Anda di AWS Wilayah yang Anda pilih. Pengidentifikasi ini digunakan dalam alamat titik akhir klaster untuk klaster DB Anda. Untuk informasi tentang titik akhir klaster, lihat Koneksi titik akhir Amazon Aurora.<br><br>Pengidentifikasi klaster DB memiliki batasan berikut:<br><br>• Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung.<br>• Karakter pertamanya harus berupa huruf.<br>• Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.<br>• Ini harus unik untuk semua cluster DB per AWS akun, per AWS Wilayah. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--db-cluster-identifier` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DBCluster Identifier` . |
| Grup parameter klaster DB | Pilih grup parameter klaster DB. Aurora memiliki grup parameter klaster DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter klaster DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat . | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--db-cluster-parameter-group-name` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DBCluster ParameterGroupName` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Kelas instans DB | Hanya berlaku untuk jenis kapasitas terprovisi. Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk setiap instans dalam klaster DB. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB. | Atur nilai ini untuk setiap instans DB di klaster Aurora Anda.<br><br>Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--db-instance-class` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `DBInstanceClass` . |
| Grup parameter DB | Pilih grup parameter. Aurora memiliki grup parameter default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter Anda sendiri. Untuk informasi selengkapnya tentang grup parameter, lihat . | Atur nilai ini untuk setiap instans DB di klaster Aurora Anda.<br><br>Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--db-parameter-group-name` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `DBParameterGroupName` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Grup subnet DB | Grup subnet DB yang Anda ingin gunakan untuk klaster DB.<br>Pilih Pilih yang ada untuk menggunakan grup subnet DB yang ada. Kemudian pilih grup subnet yang diperlukan dari daftar dropdown Grup subnet DB yang ada.<br><br>Pilih Pengaturan otomatis untuk membiarkan RDS memilih grup subnet DB yang kompatibel. Jika tidak ada, RDS membuat grup subnet baru untuk klaster Anda.<br><br>Untuk informasi selengkapnya, lihat Prasyarat klaster DB. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--db-subnet-group-name` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DBSubnetGroupName` . |
| Aktifkan perlindungan penghapusan | Pilih Aktifkan perlindungan penghapusan agar klaster DB Anda tidak terhapus. Jika Anda membuat klaster DB produksi dengan konsol, perlindungan penghapusan diaktifkan secara default. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--deletion-protection \| --no-deletion-protection` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DeletionProtection` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Aktifkan enkripsi | Pilih `Enable encryption` untuk mengaktifkan enkripsi saat diam untuk klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--storage-encrypted \| --no-storage-encrypted` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `StorageEncrypted`. |
| Aktifkan Pemantauan yang Ditingkatkan | Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. | Atur nilai-nilai ini untuk setiap instans DB di klaster Aurora Anda.<br><br>Menggunakan AWS CLI, menjalankan `create-db-instance` dan mengatur `--monitoring-interval` dan `--monitoring-role-arn` pilihan.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `MonitoringInterval` dan `MonitoringRoleArn`. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Aktifkan API Data RDS | Pilih Aktifkan API Data RDS untuk mengaktifkan RDS Data API (Data API). Data API menyediakan endpoint HTTP yang aman untuk menjalankan pernyataan SQL tanpa mengelola koneksi. Untuk informasi selengkapnya, lihat Menggunakan Amazon RDS Data API. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--enable-http-endpoint \| --no-enable-http-endpoint` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EnableHttpEndpoint` . |
| Jenis mesin | Pilih mesin basis data yang akan digunakan untuk klaster DB ini. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--engine` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `Engine`. |
| Versi mesin | Hanya berlaku untuk jenis kapasitas terprovisi. Pilih nomor versi mesin DB Anda. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--engine-version` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EngineVersion` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Prioritas failover | Pilih prioritas failover untuk instans. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora. | Atur nilai ini untuk setiap instans DB di klaster Aurora Anda.<br><br>Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--promotion-tier` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `PromotionTier` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Nama basis data awal | Masukkan nama untuk basis data default Anda. Jika Anda tidak memberikan nama untuk klaster DB Aurora MySQL, Amazon RDS tidak membuat basis data pada klaster DB yang sedang Anda buat. Jika Anda tidak memberikan nama untuk klaster DB Aurora PostgreSQL, Amazon RDS membuat basis data bernama `postgres`.<br><br>Untuk Aurora MySQL, nama basis data default memiliki batasan ini:<br>• Harus berisi antara 1–64 karakter alfanumerik.<br>• Tidak boleh menggunakan kata yang dicadangkan oleh mesin basis data.<br><br>Untuk Aurora PostgreSQL, nama basis data default memiliki batasan ini:<br>• Harus berisi antara 1–63 karakter alfanumerik.<br>• Harus dimulai dengan huruf. Karakter selanjutnya dapat berupa huruf, garis bawah, atau digit (0–9).<br>• Tidak boleh menggunakan kata yang dicadangkan oleh mesin basis data. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--database-name` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DatabaseName`. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| | Untuk membuat basis data tambahan, hubungkan ke klaster DB dan gunakan perintah SQL CREATE DATABASE. Untuk informasi selengkapnya tentang menghubungkan ke klaster DB, lihat Menghubungkan ke klaster DB Amazon Aurora. | |
| Ekspor log | Di bagian Log ekspor, pilih log yang ingin Anda mulai terbitkan ke Amazon CloudWatch Logs. Untuk informasi selengkapnya tentang menerbitkan log MySQL Aurora ke Log, lihat. CloudWatch Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch Untuk informasi selengkapnya tentang memublikasikan log PostgreSQL Aurora ke Log, lihat. CloudWatch Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--enable-cloudwatch-logs-exports` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EnableCloudwatchLogsExports` . |
| Jendela pemeliharaan | Klik Pilih jendela dan tentukan rentang waktu mingguan saat pemeliharaan sistem dapat dilakukan. Atau pilih Tidak ada preferensi agar Amazon RDS menetapkan periode secara acak. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--preferred-maintenance-window` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `PreferredMaintenanceWindow` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Kelola kredensi master di AWS Secrets Manager | Pilih Kelola kredensial master di AWS Secrets Manager untuk mengelola kata sandi pengguna master dalam rahasia di Secrets Manager.<br><br>Anda juga dapat memilih kunci KMS yang akan digunakan untuk melindungi rahasia. Pilih dari kunci KMS di akun Anda, atau masukkan kunci dari akun yang berbeda.<br><br>Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. | Menggunakan AWS CLI, menjalankan `create-db-cluster` dan mengatur `--manage-master-user-password \| --no-manage-master-user-password` dan `--master-user-secret-kms-key-id` pilihan.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `ManageMasterUserPassword` dan `MasterUserSecretKmsKeyId` . |
| Kata sandi master | Masukkan kata sandi untuk masuk ke klaster DB Anda:<br><br>• Untuk Aurora MySQL, kata sandi harus berisi 8–41 karakter ASCII yang dapat dicetak.<br>• Untuk Aurora PostgreSQL, kata sandi harus berisi 8–99 karakter ASCII yang dapat dicetak.<br>• Kata sandi tidak dapat berisi `/`, `"`, `@`, atau spasi. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--master-user-password` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `MasterUserPassword` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Nama pengguna master | Masukkan nama yang akan digunakan sebagai nama pengguna master untuk masuk ke klaster DB Anda:<br>• Untuk Aurora MySQL, nama harus berisi 1–16 karakter alfanumerik.<br>• Untuk Aurora PostgreSQL, nama harus berisi 1–63 karakter alfanumerik.<br>• Karakter pertama harus berupa huruf.<br>• Nama tidak boleh menggunakan kata yang dicadangkan oleh mesin basis data.<br><br>Anda tidak dapat mengubah nama pengguna master setelah klaster DB dibuat. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--master-username` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `MasterUsername` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Deployment Multi-AZ | Hanya berlaku untuk jenis kapasitas terprovisi. Tentukan apakah Anda ingin membuat Replika Aurora di Zona Ketersediaan lainnya untuk dukungan failover. Jika Anda memilih Buat Replika di Zona yang Berbeda, Amazon RDS akan membuat Replika Aurora untuk Anda dalam klaster DB Anda di Zona Ketersediaan yang berbeda dari instans primer untuk klaster DB Anda. Untuk informasi selengkapnya tentang beberapa Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--availability-zones` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `AvailabilityZones` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Jenis jaringan | Protokol alamat IP yang didukung oleh klaster DB.<br><br>IPv4untuk menentukan bahwa sumber daya dapat berkomunikasi dengan cluster DB hanya melalui protokol IPv4 pengalamatan.<br><br>Mode dual-stack untuk menentukan bahwa sumber daya dapat berkomunikasi dengan cluster DB over IPv4, IPv6, atau keduanya. Gunakan mode dual-stack jika Anda memiliki sumber daya yang harus berkomunikasi dengan cluster DB Anda melalui protokol IPv6 pengalamatan. Untuk menggunakan mode dual-stack, pastikan setidaknya dua subnet yang mencakup dua Availability Zone yang mendukung protokol dan jaringan. IPv4 IPv6 Juga, pastikan Anda mengaitkan blok IPv6 CIDR dengan subnet di grup subnet DB yang Anda tentukan.<br><br>Untuk informasi selengkapnya, lihat Penentuan alamat IP Amazon Aurora. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--network-type` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `NetworkType` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Akses publik | Pilih Dapat diakses publik untuk memberikan alamat IP publik pada klaster DB, atau pilih Tidak dapat diakses publik. Instans dalam klaster DB Anda dapat berupa campuran antara instans DB publik dan privat. Untuk informasi selengkapnya tentang menyembunyikan instans dari akses publik, lihat Menyembunyikan klaster DB dalam VPC dari internet.<br><br>Untuk terhubung ke instans DB dari luar Amazon VPC-nya, instans DB harus dapat diakses secara publik, akses harus diberikan menggunakan aturan masuk grup keamanan instans DB, dan persyaratan lain harus dipenuhi. Untuk mengetahui informasinya, lihat Tidak dapat terhubung ke instans DB Amazon RDS.<br><br>Jika instans DB Anda tidak dapat diakses publik, Anda juga dapat menggunakan koneksi AWS Site-to-Site VPN atau AWS Direct Connect koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan. | Atur nilai ini untuk setiap instans DB di klaster Aurora Anda.<br><br>Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--publicly-accessible \| --no-publicly-accessible` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `PubliclyAccessible` . |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| RDS Extended Support | Pilih Aktifkan RDS Extended Support untuk memungkinkan versi mesin utama yang didukung untuk terus berjalan melewati akhir Aurora dari tanggal dukungan standar.<br><br>Saat Anda membuat cluster DB, Amazon Aurora default ke RDS Extended Support. Untuk mencegah pembuatan cluster DB baru setelah Aurora berakhir pada tanggal dukungan standar dan untuk menghindari biaya untuk RDS Extended Support, nonaktifkan pengaturan ini. Cluster DB Anda yang ada tidak akan dikenakan biaya hingga tanggal mulai penetapan harga RDS Extended Support.<br><br>Untuk informasi selengkapnya, lihat Amazon RDS Extended Support dengan . | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--engine-lifecycle-support` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EngineLifecycleSupport` . |
| Proksi RDS | Pilih Buat Proksi RDS guna membuat proksi untuk klaster DB Anda. Amazon RDS secara otomatis membuat peran IAM dan rahasia Secrets Manager untuk proksi.<br><br>Untuk informasi selengkapnya, lihat Proksi Amazon RDS untuk Aurora. | Tidak tersedia saat membuat klaster DB. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Periode retensi | Pilih durasi waktu, dari 1 hingga 35 hari, saat Aurora harus mempertahankan salinan cadangan basis data. Salinan cadangan dapat digunakan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--backup-retention-period` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `BackupRetentionPeriod` . |
| Nyalakan DevOps Guru | Pilih Aktifkan DevOps Guru untuk mengaktifkan Amazon DevOps Guru untuk database Aurora Anda. Agar DevOps Guru for RDS dapat memberikan analisis terperinci tentang anomali kinerja, Performance Insights harus diaktifkan. Untuk informasi selengkapnya, lihat Menyiapkan DevOps Guru untuk RDS. | Anda dapat mengaktifkan DevOps Guru untuk RDS dari dalam konsol RDS, tetapi tidak dengan menggunakan RDS API atau CLI. Untuk informasi selengkapnya tentang mengaktifkan DevOps Guru, lihat Panduan Pengguna Amazon DevOps Guru. |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Aktifkan Wawasan Performa | Pilih Aktifkan Wawasan Performa untuk mengaktifkan Wawasan Performa Amazon RDS. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. | Atur nilai-nilai ini untuk setiap instans DB di klaster Aurora Anda.<br><br>Menggunakan AWS CLI, jalankan `create-db-instance` dengan `--enable-performance-insights \| --no-enable-performance-insights` dan tetapkan opsi `--performance-insights-kms-key-id` dan `--performance-insights-retention-period`.<br><br>Menggunakan API RDS, panggil `CreateDBInstance` `EnablePerformanceInsights` dan tetapkan parameter `PerformanceInsightsKMSKeyId` dan `PerformanceInsightsRetentionPeriod`. |
| Cloud Privat Virtual (VPC) | Pilih VPC untuk meng-host klaster DB. Pilih Buat VPC Baru agar Amazon RDS membuat VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. | Untuk API AWS CLI dan, Anda menentukan grup keamanan VPC. IDs |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Grup keamanan VPC (firewall) | Pilih Buat baru agar Amazon RDS membuat grup keamanan VPC untuk Anda. Atau klik Pilih yang ada dan tentukan satu atau beberapa grup keamanan VPC untuk mengamankan akses jaringan ke klaster DB.<br><br>Saat Anda memilih Buat baru pada konsol RDS, grup keamanan baru dibuat dengan peran masuk yang memungkinkan akses ke instans DB dari alamat IP yang terdeteksi di browser Anda.<br><br>Untuk informasi selengkapnya, lihat Prasyarat klaster DB. | Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--vpc-security-group-ids` opsi.<br><br>Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `VpcSecurityGroupIds` . |

## Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `create-db-cluster`dan operasi RDS API `CreateDBCluster`tidak berlaku untuk cluster Amazon Aurora DB.

> **Note**
>
> AWS Management Console Tidak menampilkan pengaturan ini untuk cluster Aurora DB.

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| `--allocated-storage` | `AllocatedStorage` |
| `--auto-minor-version-upgrade \| --no-auto-minor-version-upgrade` | `AutoMinorVersionUpgrade` |

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| `--db-cluster-instance-class` | `DBClusterInstanceClass` |
| `--enable-performance-insights \| --no-enable-performance-insights` | `EnablePerformanceInsights` |
| `--iops` | `Iops` |
| `--monitoring-interval` | `MonitoringInterval` |
| `--monitoring-role-arn` | `MonitoringRoleArn` |
| `--option-group-name` | `OptionGroupName` |
| `--performance-insights-kms-key-id` | `PerformanceInsightsKMSKeyId` |
| `--performance-insights-retention-period` | `PerformanceInsightsRetentionPeriod` |
| `--publicly-accessible \| --no-publicly-accessible` | `PubliclyAccessible` |

## Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `create-db-instance`dan operasi RDS API `CreateDBInstance`tidak berlaku untuk instans DB Amazon Aurora DB cluster.

> **Note**
>
> AWS Management Console Tidak menampilkan pengaturan ini untuk instans Aurora DB.

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| `--allocated-storage` | `AllocatedStorage` |

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| --availability-zone | AvailabilityZone |
| --backup-retention-period | BackupRetentionPeriod |
| --backup-target | BackupTarget |
| --character-set-name | CharacterSetName |
| --character-set-name | CharacterSetName |
| --custom-iam-instance-profile | CustomIamInstanceProfile |
| --db-security-groups | DBSecurityGroups |
| --deletion-protection \| --no-deletion-protection | DeletionProtection |
| --domain | Domain |
| --domain-iam-role-name | DomainIAMRoleName |
| --enable-cloudwatch-logs-exports | EnableCloudwatchLogsExports |
| --enable-customer-owned-ip \| --no-enable-customer-owned-ip | EnableCustomerOwnedIp |
| --enable-iam-database-authentication \| --no-enable-iam-database-authentication | EnableIAMDatabaseAuthentication |
| --engine-version | EngineVersion |
| --iops | Iops |
| --kms-key-id | KmsKeyId |
| --master-username | MasterUsername |
| --master-user-password | MasterUserPassword |

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| `--max-allocated-storage` | `MaxAllocatedStorage` |
| `--multi-az \| --no-multi-az` | `MultiAZ` |
| `--nchar-character-set-name` | `NcharCharacterSetName` |
| `--network-type` | `NetworkType` |
| `--option-group-name` | `OptionGroupName` |
| `--preferred-backup-window` | `PreferredBackupWindow` |
| `--processor-features` | `ProcessorFeatures` |
| `--storage-encrypted \| --no-storage-encrypted` | `StorageEncrypted` |
| `--storage-type` | `StorageType` |
| `--tde-credential-arn` | `TdeCredentialArn` |
| `--tde-credential-password` | `TdeCredentialPassword` |
| `--timezone` | `Timezone` |
| `--vpc-security-group-ids` | `VpcSecurityGroupIds` |

# Membuat sumber Amazon Amazon Aurora dengan AWS CloudFormation

Amazon Aurora terintegrasi dengan AWS CloudFormation, layanan yang membantu Anda memodelkan dan mengatur AWS sumber daya Anda sehingga Anda dapat menghabiskan lebih sedikit waktu untuk membuat dan mengelola sumber daya dan infrastruktur Anda. Anda membuat templat yang menjelaskan semua AWS sumber daya yang Anda inginkan (seperti cluster DB dan grup parameter cluster DB), serta menyediakan serta AWS CloudFormation mengonfigurasi sumber daya tersebut untuk Anda.

Ketika Anda menggunakan AWS CloudFormation, Anda dapat menggunakan kembali template Anda untuk mengatur sumber daya Aurora Anda secara konsisten dan berulang kali. Jelaskan sumber daya Anda sekali, lalu sediakan sumber daya yang sama berulang-ulang di beberapa AWS akun dan Wilayah.

## Aurora dan template AWS CloudFormation

AWS CloudFormation template adalah file teks yang diformat dalam JSON atauYAML. Template ini menjelaskan sumber daya yang ingin Anda sediakan di AWS CloudFormation tumpukan Anda. Jika Anda tidak terbiasa dengan JSON atauYAML, Anda dapat menggunakan AWS CloudFormation Designer untuk membantu Anda memulai dengan AWS CloudFormation template. Untuk informasi selengkapnya, lihat Apa itu AWS CloudFormation Designer? di Panduan Pengguna AWS CloudFormation .

Aurora mendukung pembuatan sumber daya di. AWS CloudFormation Untuk informasi selengkapnya, termasuk contoh JSON dan YAML templat untuk sumber daya ini, lihat referensi jenis RDS sumber daya di Panduan AWS CloudFormation Pengguna.

## Pelajari lebih lanjut tentang AWS CloudFormation

Untuk mempelajari selengkapnya AWS CloudFormation, lihat sumber daya berikut:

- AWS CloudFormation
- AWS CloudFormation Panduan Pengguna
- AWS CloudFormation APIReferensi
- AWS CloudFormation Panduan Pengguna Antarmuka Baris Perintah

# Menghubungkan ke klaster DB Amazon Aurora

Anda dapat menghubungkan ke klaster DB Aurora menggunakan alat yang sama yang Anda gunakan untuk menghubungkan ke basis data MySQL atau PostgreSQL. Anda menentukan string koneksi dengan skrip, utilitas, atau aplikasi apa pun yang terhubung ke instans DB MySQL atau PostgreSQL. Anda menggunakan kunci publik yang sama untuk koneksi Lapisan Soket Aman (SSL).

Dalam string koneksi, Anda biasanya menggunakan informasi host dan port dari titik akhir khusus yang terkait dengan klaster DB. Dengan titik akhir ini, Anda dapat menggunakan parameter koneksi yang sama terlepas dari berapa banyak instans DB yang ada di klaster. Anda juga menggunakan informasi host dan port dari instans DB tertentu di klaster DB Aurora Anda untuk tugas-tugas khusus, seperti pemecahan masalah.

> **Note**
>
> Untuk klaster DB Aurora Serverless, Anda terhubung ke titik akhir basis data bukan instans DB. Anda dapat menemukan titik akhir basis data untuk klaster DB Aurora Serverless pada tab Konektivitas & keamanan di AWS Management Console. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.

Terlepas dari mesin DB Aurora dan alat-alat khusus yang Anda gunakan untuk menangani klaster atau instans DB, titik akhir harus dapat diakses. Cluster Aurora DB hanya dapat dibuat di cloud pribadi virtual (VPC) berdasarkan layanan VPC Amazon. Artinya, Anda mengakses titik akhir dari dalam VPC atau luar VPC menggunakan salah satu pendekatan berikut.

- Akses cluster Aurora DB di dalam VPC — Aktifkan akses ke cluster Aurora DB melalui VPC. Anda melakukannya dengan mengedit Aturan masuk pada Grup keamanan untuk VPC untuk mengizinkan akses ke klaster DB Aurora spesifik. Untuk mempelajari lebih lanjut, termasuk cara mengonfigurasi VPC Anda untuk berbagai skenario klaster Aurora DB, lihat Amazon Virtual Private Cloud dan Amazon Aurora. VPCs
- Akses cluster Aurora DB di luar VPC — Untuk mengakses cluster Aurora DB dari luar VPC, gunakan alamat titik akhir publik cluster DB.

Untuk informasi selengkapnya, lihat Pemecahan masalah kegagalan koneksi Aurora.

Daftar Isi

- Menghubungkan ke cluster Aurora DB dengan driver AWS
- Menghubungkan ke klaster DB Amazon Aurora MySQL
  - Utilitas koneksi untuk Aurora MySQL
  - Menghubungkan ke Aurora MySQL dengan utilitas MySQL
  - Menghubungkan ke Aurora MySQL dengan Driver Amazon Web Services () JDBC AWS
  - Menghubungkan ke Aurora MySQL dengan Driver Python Amazon Web Services ()AWS
  - Menghubungkan ke Aurora MySQL dengan Amazon Web Services () ODBC Driver untuk MySQL AWS
  - Menghubungkan ke Aurora MySQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS
  - Menghubungkan ke Aurora MySQL menggunakan SSL
- Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.
  - Utilitas koneksi untuk Aurora PostgreSQL
  - Menghubungkan ke Aurora PostgreSQL dengan Driver Amazon Web Services () JDBC AWS
  - Menghubungkan ke Aurora PostgreSQL dengan Driver Python Amazon Web Services ()AWS
  - Menghubungkan ke Aurora PostgreSQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS
- Pemecahan masalah kegagalan koneksi Aurora


## Menghubungkan ke cluster Aurora DB dengan driver AWS

AWS Rangkaian driver telah dirancang untuk memberikan dukungan untuk waktu peralihan dan failover yang lebih cepat, dan otentikasi dengan, AWS Identity and Access Management (IAM) AWS Secrets Manager, dan Federated Identity. AWS Driver mengandalkan pemantauan status cluster DB dan menyadari topologi cluster untuk menentukan penulis baru. Pendekatan ini mengurangi waktu peralihan dan failover menjadi satu digit detik, dibandingkan dengan puluhan detik untuk driver open-source.

Tabel berikut mencantumkan fitur yang didukung untuk masing-masing driver. Ketika fitur layanan baru diperkenalkan, tujuan dari AWS rangkaian driver adalah untuk memiliki dukungan bawaan untuk fitur layanan ini.

| Fitur | AWS Pengemudi JDBC | AWS Pengemudi Python | AWS Driver ODBC untuk MySQL | AWS Pembungkus NodeJS Tingkat Lanjut |
|---|---|---|---|---|
| Dukungan Failover | Ya | Ya | Ya | Ya |
| Pemantauan failover yang ditingkatkan | Ya | Ya | Ya | Ya |
| Membaca/ menulis pemisahan | Ya | Ya | Tidak | Ya |
| Pelacak koneksi Aurora | Ya | Ya | Tidak | Ya |
| Koneksi metadata driver | Ya | N/A | N/A | N/A |
| Telemetri | Ya | Ya | Tidak | Ya |
| Secrets Manager | Ya | Ya | Ya | Ya |
| Autentikasi IAM | Ya | Ya | Ya | Ya |
| Identitas Federasi (AD FS) | Ya | Ya | Tidak | Ya |
| Identitas Federasi (Okta) | Ya | Ya | Ya | Ya |
| Database Tanpa Batas Aurora PostgreSQL | Ya (Aurora PostgreSQL saja) | Tidak | Tidak | Ya (Aurora PostgreSQL saja) |

Untuk informasi lebih lanjut tentang AWS driver, lihat driver bahasa yang sesuai untuk klaster Aurora MySQL atau Aurora PostgreSQL DB Anda.

## Menghubungkan ke klaster DB Amazon Aurora MySQL

Untuk mengautentikasi ke cluster DB MySQL Aurora Anda, Anda dapat menggunakan nama pengguna MySQL dan otentikasi kata sandi atau otentikasi basis data (IAM). AWS Identity and Access Management Untuk informasi selengkapnya tentang menggunakan autentikasi nama pengguna dan kata sandi MySQL, lihat Access control and account management dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang menggunakan autentikasi basis data IAM, lihat Autentikasi basis data IAM.

Saat Anda memiliki koneksi ke klaster DB Amazon Aurora dengan kompatibilitas MySQL 8.0, Anda dapat menjalankan perintah SQL yang kompatibel dengan MySQL versi 8.0. Versi minimum yang kompatibel adalah MySQL 8.0.23. Untuk informasi selengkapnya tentang sintaksis SQL MySQL 8.0, lihat Panduan referensi MySQL 8.0. Untuk informasi tentang batasan yang berlaku untuk Aurora MySQL versi 3, lihat Membandingkan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition.

Saat Anda memiliki koneksi ke klaster DB Amazon Aurora dengan kompatibilitas MySQL 5.7, Anda dapat menjalankan perintah SQL yang kompatibel dengan MySQL versi 5.7. Untuk informasi selengkapnya tentang sintaksis SQL MySQL 5.7, lihat Panduan referensi MySQL 5.7. Untuk informasi tentang batasan yang berlaku untuk Aurora MySQL 5.7, lihat Aurora SQL Versi saya 2 kompatibel dengan My 5.7 SQL.

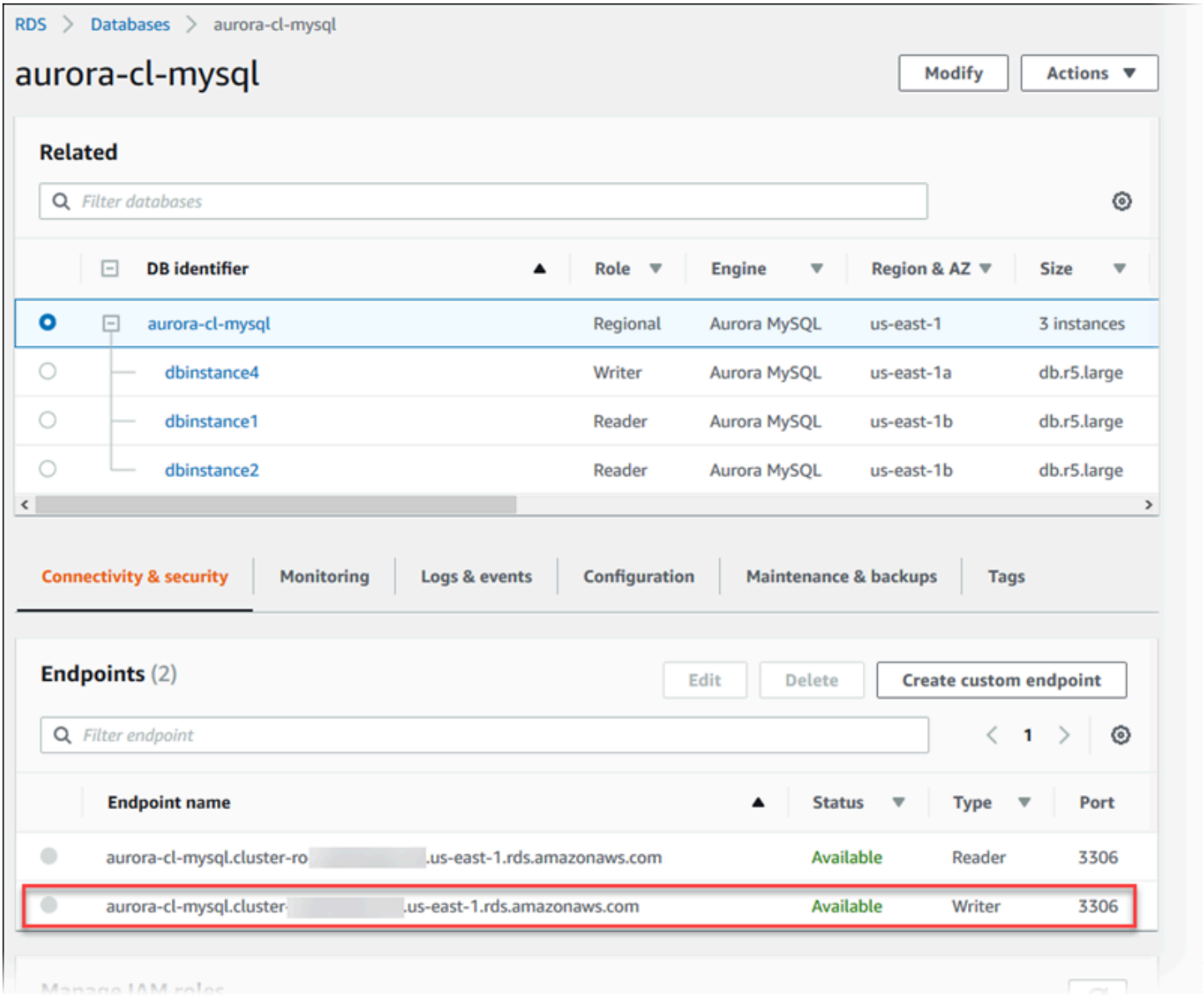
> **Note**
>
> Untuk panduan berguna dan mendetail tentang menghubungkan ke klaster DB Amazon Aurora MySQL, Anda dapat melihat buku panduan Manajemen koneksi Aurora.

Dalam tampilan detail untuk klaster DB Anda, Anda dapat menemukan titik akhir klaster, yang dapat Anda gunakan dalam string koneksi MySQL Anda. Titik akhir terdiri dari nama domain dan port untuk klaster DB Anda. Misalnya, jika nilai titik akhir adalah `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306`, maka Anda menentukan nilai berikut dalam string koneksi MySQL:

- Untuk host atau nama host, tentukan `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- Untuk port, tentukan `3306` atau nilai port yang Anda gunakan saat Anda membuat klaster DB

Titik akhir klaster menghubungkan Anda ke instans primer untuk klaster DB. Anda dapat melakukan operasi baca dan tulis menggunakan titik akhir klaster. Klaster DB Anda juga dapat memiliki hingga 15 Replika Aurora yang mendukung akses hanya baca ke data di klaster DB Anda. Instans primer dan setiap Replika Aurora memiliki titik akhir unik yang tidak bergantung pada titik akhir klaster dan memungkinkan Anda terhubung ke instans DB tertentu dalam klaster secara langsung. Titik akhir klaster selalu mengarah ke instans primer. Jika instans primer gagal dan diganti, maka titik akhir klaster mengarah ke instans primer baru.

Untuk melihat titik akhir klaster (titik akhir penulis), pilih Basis data di konsol Amazon RDS dan pilih nama klaster DB untuk menampilkan detail klaster DB.




Topik

- Utilitas koneksi untuk Aurora MySQL
- Menghubungkan ke Aurora MySQL dengan utilitas MySQL
- Menghubungkan ke Aurora MySQL dengan Driver Amazon Web Services () JDBC AWS
- Menghubungkan ke Aurora MySQL dengan Driver Python Amazon Web Services ()AWS
- Menghubungkan ke Aurora MySQL dengan Amazon Web Services () ODBC Driver untuk MySQL AWS
- Menghubungkan ke Aurora MySQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS
- Menghubungkan ke Aurora MySQL menggunakan SSL

## Utilitas koneksi untuk Aurora MySQL

Beberapa utilitas koneksi yang dapat Anda gunakan adalah sebagai berikut:

- Baris perintah – Anda dapat terhubung ke klaster DB Amazon Aurora dengan menggunakan alat seperti utilitas baris perintah MySQL. Untuk informasi selengkapnya tentang menggunakan utilitas MySQL, lihat mysql — the MySQL command-line client dalam dokumentasi MySQL.
- GUI – Anda dapat menggunakan utilitas MySQL Workbench untuk terhubung dengan menggunakan antarmuka UI. Untuk informasi selengkapnya, lihat halaman Unduh MySQL Workbench.
- AWS driver:
  - Menghubungkan ke Aurora MySQL dengan Driver Amazon Web Services () JDBC AWS
  - Menghubungkan ke Aurora MySQL dengan Driver Python Amazon Web Services ()AWS
  - Menghubungkan ke Aurora MySQL dengan Amazon Web Services () ODBC Driver untuk MySQL AWS
  - Menghubungkan ke Aurora MySQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS

## Menghubungkan ke Aurora MySQL dengan utilitas MySQL

Gunakan prosedur berikut: Prosedur ini mengasumsikan bahwa Anda mengonfigurasi klaster DB Anda di subnet privat di VPC Anda. Anda terhubung menggunakan EC2 instans Amazon yang Anda konfigurasikan sesuai dengan tutorial diTutorial: Membuat server web dan klaster DB Amazon Aurora.

> **Note**
>
> Prosedur ini tidak memerlukan penginstalan server web dalam tutorial tersebut, tetapi memerlukan penginstalan MariaDB 10.5.

Untuk terhubung ke klaster DB menggunakan utilitas MySQL

1. Masuk ke EC2 instance yang Anda gunakan untuk terhubung ke cluster DB Anda.

   Anda akan melihat output seperti yang berikut ini.

   ```
   Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

          __|  __|_  )
          _|  (     /   Amazon Linux 2 AMI
         ___|\___|___|

   https://aws.amazon.com/amazon-linux-2/
   [ec2-user@ip-10-0-xxx.xxx ~]$
   ```

2. Ketikkan perintah berikut pada baris perintah untuk terhubung ke instans DB primer dari klaster DB Anda.

   Untuk parameter `-h`, ganti nama DNS titik akhir untuk instans primer Anda. Untuk parameter `-u`, ganti ID pengguna untuk akun pengguna basis data.

   ```
   mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
    -u database_user -p
   ```

   Contoh:

   ```
   mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
   central-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. Masukkan kata sandi untuk pengguna basis data.

   Anda akan melihat output seperti yang berikut ini.

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 1770
   ```

```
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

4. Masukkan perintah SQL Anda.

## Menghubungkan ke Aurora MySQL dengan Driver Amazon Web Services () JDBC AWS

Amazon Web Services (AWS) JDBC Driver dirancang sebagai pembungkus JDBC tingkat lanjut. Pembungkus ini melengkapi dan memperluas fungsionalitas driver JDBC yang ada untuk membantu aplikasi memanfaatkan fitur database berkerumun seperti Aurora MySQL. Driver ini kompatibel dengan driver MySQL Connector/J driver and the community MariaDB Connector/J komunitas.

Untuk menginstal Driver AWS JDBC, tambahkan file AWS JDBC Driver .jar (terletak di aplikasi`CLASSPATH`), dan simpan referensi ke driver komunitas masing-masing. Perbarui awalan URL koneksi masing-masing sebagai berikut:

- `jdbc:mysql://` untuk `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` untuk `jdbc:aws-wrapper:mariadb://`

Untuk informasi selengkapnya tentang Driver AWS JDBC dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) JDBC Driver. GitHub

> **Note**
>
> Versi 3.0.3 dari utilitas MariaDB Connector/J menjatuhkan dukungan untuk cluster Aurora DB, jadi kami sangat menyarankan pindah ke Driver JDBC. AWS

## Menghubungkan ke Aurora MySQL dengan Driver Python Amazon Web Services ()AWS

Driver Python Amazon Web Services (AWS) dirancang sebagai pembungkus Python tingkat lanjut. Pembungkus ini melengkapi dan memperluas fungsionalitas driver Psycopg open-source. Driver

AWS Python mendukung Python versi 3.8 dan lebih tinggi. Anda dapat menginstal `aws-advanced-python-wrapper` paket menggunakan `pip` perintah, bersama dengan paket `psycopg` open-source.

Untuk informasi selengkapnya tentang Driver AWS Python dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services ()AWS Python Driver. GitHub

## Menghubungkan ke Aurora MySQL dengan Amazon Web Services () ODBC Driver untuk MySQL AWS

Driver AWS ODBC untuk MySQL adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora MySQL. Driver dapat ada di samping driver MySQL/ODBC Connector/ODBC dan kompatibel dengan alur kerja yang sama.

Untuk informasi selengkapnya tentang Driver AWS ODBC untuk MySQL dan petunjuk lengkap untuk menginstal dan menggunakannya, lihat Driver Amazon Web Services AWS() ODBC untuk repositori MySQL. GitHub

## Menghubungkan ke Aurora MySQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS

AWS Advanced NodeJS Wrapper melengkapi dan memperluas fungsionalitas driver NodeJS yang ada. Ini membantu aplikasi memanfaatkan fitur database berkerumun seperti Aurora MySQL.

Untuk informasi selengkapnya tentang AWS Advanced NodeJS Wrapper dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) Advanced NodeJS Wrapper. GitHub

## Menghubungkan ke Aurora MySQL menggunakan SSL

Anda dapat menggunakan enkripsi SSL pada koneksi ke instans DB Amazon Aurora MySQL. Untuk informasi, lihat Koneksi TLS ke cluster DB MySQL Aurora.

Untuk terhubung menggunakan SSL, gunakan utilitas MySQL yang dijelaskan dalam prosedur berikut. Jika Anda menggunakan autentikasi basis data IAM, Anda harus menggunakan koneksi SSL. Untuk informasi, lihat Autentikasi basis data IAM.

> **Note**
>
> Untuk terhubung ke titik akhir klaster menggunakan SSL, utilitas koneksi klien Anda harus mendukung Nama Alternatif Subjek (SAN). Jika koneksi klien Anda tidak mendukung SAN,

Anda dapat terhubung langsung ke instans di klaster DB Aurora Anda. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Koneksi titik akhir Amazon Aurora.

Untuk terhubung ke klaster DB dengan SSL menggunakan utilitas MySQL

1. Unduh kunci publik untuk sertifikat penandatanganan Amazon RDS.

   Untuk informasi tentang mengunduh sertifikat, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .
2. Ketikkan perintah berikut pada baris perintah untuk terhubung ke instans primer dari klaster DB dengan SSL menggunakan utilitas MySQL. Untuk parameter `-h`, ganti nama DNS titik akhir untuk instans primer Anda. Untuk parameter `-u`, ganti ID pengguna untuk akun pengguna basis data. Untuk parameter `--ssl-ca`, ganti nama file sertifikat SSL yang sesuai. Ketikkan kata sandi pengguna master saat diminta.

   ```
   mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-cert
   ```

Anda akan melihat output seperti yang berikut ini.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Untuk petunjuk umum tentang membuat string koneksi RDS for MySQL dan menemukan kunci publik untuk koneksi SSL, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

## Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.

Anda dapat terhubung ke instans DB di klaster DB Amazon Aurora PostgreSQL menggunakan alat yang sama yang Anda gunakan untuk menghubungkan ke basis data PostgreSQL. Sebagai bagian dari hal ini, Anda menggunakan kunci publik yang sama untuk koneksi Lapisan Soket Aman (SSL). Anda dapat menggunakan informasi titik akhir dan port dari instans primer atau Replika Aurora di

klaster DB Aurora PostgreSQL Anda dalam string koneksi skrip, utilitas, atau aplikasi apa pun yang terhubung ke instans DB PostgreSQL. Dalam string koneksi, tentukan alamat DNS dari instans primer atau titik akhir Replika Aurora sebagai parameter host. Tentukan nomor port dari titik akhir sebagai parameter port.

Saat Anda memiliki koneksi ke instans DB di klaster DB Amazon Aurora PostgreSQL, Anda dapat menjalankan perintah SQL apa pun yang kompatibel dengan PostgreSQL.

Dalam tampilan detail untuk klaster DB Aurora PostgreSQL, Anda dapat menemukan nama, status, jenis, dan nomor port titik akhir klaster. Anda menggunakan titik akhir dan nomor port di string koneksi PostgreSQL Anda. Misalnya, jika nilai titik akhir adalah `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`, maka Anda menentukan nilai berikut dalam string koneksi PostgreSQL:

- Untuk host atau nama host, tentukan `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- Untuk port, tentukan `5432` atau nilai port yang Anda gunakan saat Anda membuat klaster DB

Titik akhir klaster menghubungkan Anda ke instans primer untuk klaster DB. Anda dapat melakukan operasi baca dan tulis menggunakan titik akhir klaster. Klaster DB Anda juga dapat memiliki hingga 15 Replika Aurora yang mendukung akses hanya baca ke data di klaster DB Anda. Setiap instans DB dalam klaster Aurora (yaitu, instans primer dan setiap Replika Aurora) memiliki titik akhir unik yang independen dari titik akhir klaster. Titik akhir unik ini memungkinkan Anda terhubung ke instans DB tertentu dalam klaster secara langsung. Titik akhir klaster selalu mengarah ke instans primer. Jika instans primer gagal dan diganti, titik akhir klaster mengarah ke instans primer baru.

Untuk melihat titik akhir klaster (titik akhir penulis), pilih Basis data di konsol Amazon RDS dan pilih nama klaster DB untuk menampilkan detail klaster DB.

## Utilitas koneksi untuk Aurora PostgreSQL

Beberapa utilitas koneksi yang dapat Anda gunakan adalah sebagai berikut:

- Baris perintah – Anda dapat terhubung ke klaster DB Aurora PostgreSQL dengan menggunakan alat seperti psql, terminal interaktif PostgreSQL. Untuk informasi selengkapnya tentang menggunakan terminal interaktif PostgreSQL, lihat psql dalam dokumentasi PostgreSQL.
- GUI – Anda dapat menggunakan utilitas pgAdmin untuk terhubung ke klaster DB Aurora PostgreSQL dengan menggunakan antarmuka UI. Untuk informasi selengkapnya, lihat halaman Unduhan dari situs web pgAdmin.

- AWS driver:
  - Menghubungkan ke Aurora PostgreSQL dengan Driver Amazon Web Services () JDBC AWS
  - Menghubungkan ke Aurora PostgreSQL dengan Driver Python Amazon Web Services ()AWS
  - Menghubungkan ke Aurora PostgreSQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS

## Menghubungkan ke Aurora PostgreSQL dengan Driver Amazon Web Services () JDBC AWS

Amazon Web Services (AWS) JDBC Driver dirancang sebagai pembungkus JDBC tingkat lanjut. Pembungkus ini melengkapi dan memperluas fungsionalitas driver JDBC yang ada untuk membantu aplikasi memanfaatkan fitur database berkerumun seperti Aurora PostgreSQL. Driver ini kompatibel dengan driver PgJDBC komunitas.

Untuk menginstal Driver AWS JDBC, tambahkan file AWS JDBC Driver .jar (terletak di aplikasi`CLASSPATH`), dan simpan referensi ke driver komunitas PgJDBC. Perbarui awalan URL koneksi dari `jdbc:postgresql://` `kejdbc:aws-wrapper:postgresql://`.

Untuk informasi selengkapnya tentang Driver AWS JDBC dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) JDBC Driver. GitHub

## Menghubungkan ke Aurora PostgreSQL dengan Driver Python Amazon Web Services ()AWS

Driver Python Amazon Web Services (AWS) dirancang sebagai pembungkus Python tingkat lanjut. Pembungkus ini melengkapi dan memperluas fungsionalitas driver Psycopg open-source. Driver AWS Python mendukung Python versi 3.8 dan lebih tinggi. Anda dapat menginstal `aws-advanced-python-wrapper` paket menggunakan `pip` perintah, bersama dengan paket `psycopg` open-source.

Untuk informasi selengkapnya tentang Driver AWS Python dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services ()AWS Python Driver. GitHub

## Menghubungkan ke Aurora PostgreSQL dengan Amazon Web Services () Advanced NodeJS Wrapper AWS

AWS Advanced NodeJS Wrapper melengkapi dan memperluas fungsionalitas driver NodeJS yang ada. Ini membantu aplikasi memanfaatkan fitur database berkerumun seperti Aurora PostgreSQL.

Untuk informasi selengkapnya tentang AWS Advanced NodeJS Wrapper dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) Advanced NodeJS Wrapper. GitHub

## Pemecahan masalah kegagalan koneksi Aurora

Penyebab umum kegagalan koneksi ke klaster DB Aurora baru adalah sebagai berikut:

- Grup keamanan di VPC tidak mengizinkan akses — VPC Anda harus mengizinkan koneksi dari perangkat Anda atau dari EC2 instans Amazon dengan konfigurasi grup keamanan yang tepat di VPC. Untuk mengatasinya, modifikasi aturan masuk grup keamanan VPC Anda untuk mengizinkan koneksi. Sebagai contoh, lihat Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .
- Port diblokir oleh aturan firewall – Periksa nilai port yang dikonfigurasi untuk klaster DB Aurora Anda. Jika aturan firewall memblokir port tersebut, Anda dapat membuat ulang instans menggunakan port yang berbeda.
- Konfigurasi IAM tidak lengkap atau salah – Jika Anda membuat instans DB Aurora Anda untuk menggunakan autentikasi berbasis IAM, pastikan bahwa autentikasi tersebut dikonfigurasi dengan benar. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

Untuk informasi selengkapnya tentang pemecahan masalah koneksi Aurora DB, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Parameter basis data menentukan konfigurasi basis data. Misalnya, parameter basis data dapat menentukan jumlah sumber daya, seperti memori, yang akan dialokasikan ke basis data.

Anda mengelola konfigurasi basis data dengan mengaitkan instans DB dan klaster DB Aurora dengan grup parameter. Aurora menentukan grup parameter dengan pengaturan default. Anda dapat juga menentukan grup parameter Anda sendiri dengan pengaturan yang disesuaikan.

Topik


- Ikhtisar grup parameter
- Grup parameter cluster DB untuk cluster Amazon Aurora DB
- Grup parameter DB untuk instans Aurora DB Amazon
- Membandingkan grup parameter DB
- Menentukan parameter DB


## Ikhtisar grup parameter

Grup parameter klaster DB bertindak sebagai kontainer untuk nilai konfigurasi mesin yang diterapkan pada setiap instans DB di klaster DB Aurora. Misalnya, model penyimpanan bersama Aurora mengharuskan setiap instans DB dalam klaster Aurora menggunakan pengaturan yang sama untuk parameter seperti `innodb_file_per_table`. Dengan demikian, parameter yang memengaruhi tata letak penyimpanan fisik adalah bagian dari grup parameter klaster. Grup parameter klaster DB juga mencakup nilai default untuk semua parameter tingkat instans.

Grup parameter DB bertindak sebagai kontainer untuk nilai konfigurasi mesin yang diterapkan pada satu atau beberapa instans DB. Grup parameter DB berlaku untuk instans DB di Amazon RDS maupun Aurora. Pengaturan konfigurasi ini berlaku untuk properti yang dapat beragam di antara instans DB dalam klaster Aurora, misalnya ukuran untuk buffer memori.

Topik


- Grup parameter kustom dan default
- Parameter klaster DB statis dan dinamis
- Parameter instans DB statis dan dinamis
- Parameter set karakter
- Parameter dan nilai parameter yang didukung

## Grup parameter kustom dan default

Jika Anda membuat instans DB tanpa menentukan grup parameter DB, instans DB akan menggunakan grup parameter DB default. Demikian pula, jika Anda membuat klaster DB Aurora tanpa menentukan grup parameter klaster DB, klaster DB tersebut akan menggunakan grup parameter klaster DB default. Setiap grup parameter default berisi default mesin basis data dan default sistem Amazon RDS berdasarkan mesin, kelas komputasi, dan penyimpanan yang dialokasikan dari instans.

Anda tidak dapat memodifikasi pengaturan parameter dari grup parameter default. Anda dapat melakukan hal berikut sebagai gantinya:

1. Membuat grup parameter baru.
2. Mengubah pengaturan parameter sesuai keinginan Anda. Tidak semua parameter mesin DB dalam grup parameter memenuhi syarat untuk dimodifikasi.
3. Ubah instans DB atau cluster DB Anda untuk mengaitkan grup parameter baru.

   Untuk mengetahui informasi tentang cara memodifikasi klaster atau instans DB, lihat Memodifikasi klaster DB Amazon Aurora.

   > **Note**
   >
   > Jika Anda telah memodifikasi instans DB Anda untuk menggunakan grup parameter kustom, dan Anda memulai instans DB, RDS secara otomatis mem-boot ulang instans DB sebagai bagian dari proses startup. Untuk instans RDS untuk SQL Server Multi-AZ dengan AlwaysOn atau opsi Mirroring diaktifkan, failover diharapkan saat instance di-boot ulang setelah proses startup.

RDS menerapkan parameter statis dan dinamis yang dimodifikasi dalam grup parameter yang baru terkait hanya setelah instance DB di-boot ulang. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini akan diterapkan segera tanpa boot ulang. Untuk informasi selengkapnya tentang cara mengubah grup parameter DB, lihat Memodifikasi klaster DB Amazon Aurora.

Jika Anda memperbarui parameter dalam grup parameter DB, perubahan berlaku untuk semua instans DB yang terkait dengan grup parameter tersebut. Demikian pula, jika Anda memperbarui

parameter dalam grup parameter klaster DB Aurora, perubahan berlaku untuk semua klaster DB Aurora yang terkait dengan grup parameter klaster DB tersebut.

Jika Anda tidak ingin membuat grup parameter dari awal, Anda dapat menyalin grup parameter yang ada dengan AWS CLI copy-db-parameter-groupperintah atau perintah copy-db-cluster-parameter-group. Menyalin grup parameter dapat membantu dalam beberapa kasus. Misalnya, Anda mungkin ingin menyertakan sebagian besar parameter dan nilai kustom grup parameter yang ada dalam grup parameter baru.

### Parameter klaster DB statis dan dinamis

Parameter klaster DB bersifat statis atau dinamis. Perbedaannya terletak pada hal berikut:

- Saat Anda mengubah parameter statis dan menyimpan grup parameter klaster DB, perubahan parameter akan berlaku setelah Anda mem-boot ulang instans DB secara manual di setiap klaster DB terkait. Bila Anda menggunakan AWS Management Console untuk mengubah nilai parameter cluster DB statis, selalu digunakan `pending-reboot` untuk`ApplyMethod`.
- Saat Anda mengubah parameter dinamis, perubahan parameter akan langsung diterapkan secara default, tanpa harus di-boot ulang. Saat Anda menggunakan konsol, konsol tersebut selalu menggunakan `immediate` untuk `ApplyMethod`. Untuk menunda perubahan parameter hingga setelah Anda me-reboot instans DB di cluster DB terkait, gunakan AWS CLI atau RDS API. Atur `ApplyMethod` ke `pending-reboot` untuk perubahan parameter.

Untuk informasi selengkapnya tentang menggunakan AWS CLI untuk mengubah nilai parameter, lihat modify-db-cluster-parameter-group. Untuk informasi selengkapnya tentang menggunakan RDS API untuk mengubah nilai parameter, lihat Memodifikasi DBCluster ParameterGroup.

Jika Anda mengubah grup parameter klaster DB yang terkait dengan klaster DB, boot ulang instans DB di klaster DB tersebut. Dengan boot ulang, perubahan akan diterapkan ke semua instans DB di klaster DB. Untuk menentukan apakah instans DB dari klaster DB harus di-boot ulang untuk menerapkan perubahan, jalankan perintah AWS CLI berikut.

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

Periksa nilai `DBClusterParameterGroupStatus` untuk instans DB primer dalam output. Jika nilainya adalah `pending-reboot`, boot ulang instans DB dari klaster DB.

### Parameter instans DB statis dan dinamis

Parameter instans DB bersifat statis atau dinamis. Perbedaannya terletak dalam hal berikut:

- Saat Anda mengubah parameter statis dan menyimpan grup parameter DB, perubahan parameter akan diterapkan setelah Anda mem-boot ulang instans DB terkait secara manual. Untuk parameter statis, konsol selalu menggunakan `pending-reboot` untuk `ApplyMethod`.
- Saat Anda mengubah parameter dinamis, perubahan parameter akan langsung diterapkan secara default, tanpa harus di-boot ulang. Saat Anda menggunakan AWS Management Console untuk mengubah nilai parameter instans DB, selalu digunakan `immediate` untuk parameter dinamis `ApplyMethod` for. Untuk menunda perubahan parameter hingga setelah Anda me-reboot instans DB terkait, gunakan AWS CLI atau RDS API. Atur `ApplyMethod` ke `pending-reboot` untuk perubahan parameter.

Untuk informasi selengkapnya tentang menggunakan AWS CLI untuk mengubah nilai parameter, lihat modify-db-parameter-group. Untuk informasi selengkapnya tentang menggunakan RDS API untuk mengubah nilai parameter, lihat Memodifikasi DBParameter Grup.

Jika instans DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait, konsol menunjukkan status boot ulang tertunda untuk grup parameter DB. Status ini tidak menyebabkan boot ulang otomatis selama periode pemeliharaan berikutnya. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, boot ulang instans DB secara manual.

### Parameter set karakter

Sebelum membuat klaster DB, atur parameter apa pun yang terkait dengan set karakter atau kolasi basis data di grup parameter Anda. Lakukan juga sebelum Anda membuat basis data di dalamnya. Dengan cara ini, Anda memastikan basis data default dan basis data baru menggunakan nilai kolasi dan set karakter yang Anda tentukan. Jika Anda mengubah parameter kolasi atau set karakter, perubahan parameter tidak diterapkan ke basis data yang sudah ada.

Untuk beberapa mesin DB, Anda dapat mengubah nilai kolasi atau set karakter untuk basis data yang sudah ada menggunakan perintah `ALTER DATABASE`, contohnya:

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

Untuk informasi selengkapnya tentang cara mengubah nilai set kolasi atau set karakter untuk basis data, periksa dokumentasi untuk mesin DB Anda.

### Parameter dan nilai parameter yang didukung

Untuk menentukan parameter yang didukung untuk mesin DB, lihat parameter di grup parameter DB dan grup parameter klaster DB yang digunakan oleh instans DB atau klaster DB. Untuk informasi

selengkapnya, lihat Melihat nilai parameter untuk grup parameter DB di Aurora dan Melihat nilai parameter untuk grup parameter cluster DB di Amazon Aurora.

Dalam banyak kasus, Anda juga dapat menentukan parameter bilangan bulat dan Boolean menggunakan ekspresi, rumus, dan fungsi. Fungsi dapat mencakup ekspresi log matematis. Namun, tidak semua parameter mendukung ekspresi, rumus, dan fungsi untuk nilai parameter. Untuk informasi selengkapnya, lihat Menentukan parameter DB.

Untuk basis data global Aurora, Anda dapat menentukan pengaturan konfigurasi yang berbeda untuk setiap klaster Aurora. Pastikan pengaturannya cukup mirip untuk menghasilkan perilaku yang konsisten jika Anda menggunakan klaster sekunder sebagai klaster primer. Misalnya, gunakan pengaturan yang sama untuk zona waktu dan set karakter di semua klaster basis data global Aurora.

Pengaturan parameter yang tidak tepat dalam grup parameter dapat menimbulkan dampak buruk yang tidak diinginkan, termasuk penurunan performa dan ketidakstabilan sistem. Selalu berhati-hati saat memodifikasi parameter basis data, dan cadangkan data sebelum memodifikasi grup parameter. Coba perubahan pengaturan grup parameter pada instans DB atau klaster DB uji coba sebelum menerapkan perubahan grup parameter tersebut ke instans DB atau klaster DB produksi.

## Grup parameter cluster DB untuk cluster Amazon Aurora DB

Klaster DB Amazon Aurora menggunakan grup parameter klaster DB. Bagian berikut menjelaskan cara mengonfigurasi dan mengelola grup parameter klaster DB.

Topik

- Parameter klaster DB dan instans DB Amazon Aurora
- Membuat grup parameter cluster DB di Amazon Aurora
- Mengaitkan grup parameter cluster DB dengan cluster DB di Amazon Aurora
- Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora
- Menyetel ulang parameter dalam grup parameter cluster DB di Amazon Aurora
- Menyalin grup parameter cluster DB di Amazon Aurora
- Daftar grup parameter cluster DB di Amazon Aurora
- Melihat nilai parameter untuk grup parameter cluster DB di Amazon Aurora
- Menghapus grup parameter cluster DB di Amazon Aurora

## Parameter klaster DB dan instans DB Amazon Aurora

Aurora menggunakan sistem pengaturan konfigurasi dua tingkat:

- Parameter dalam Grup parameter klaster DB berlaku untuk setiap instans DB dalam klaster DB. Data Anda disimpan dalam subsistem penyimpanan bersama Aurora. Oleh karena itu, semua parameter yang terkait dengan tata letak fisik data tabel harus sama untuk semua instans DB dalam klaster Aurora. Demikian juga, karena instans DB Aurora dihubungkan dengan replikasi, semua parameter untuk pengaturan replikasi harus identik di seluruh klaster Aurora.
- Parameter dalam Grup parameter DB berlaku untuk satu instans DB dalam klaster DB Aurora. Parameter ini terkait dengan sejumlah aspek, seperti penggunaan memori yang dapat divariasikan di seluruh instans DB dalam klaster Aurora yang sama. Misalnya, klaster sering kali berisi instans DB dengan kelas instans AWS yang berbeda.

Setiap klaster Aurora dikaitkan dengan grup parameter klaster DB. Grup parameter ini menetapkan nilai default untuk setiap nilai konfigurasi untuk mesin DB yang sesuai. Grup parameter klaster mencakup default untuk parameter tingkat klaster dan tingkat instans. Setiap instans DB dalam klaster terprovisi atau klaster Aurora Serverless v2 mewarisi pengaturan dari grup parameter klaster DB tersebut.

Setiap instans DB juga terkait dengan grup parameter DB. Nilai dalam grup parameter DB dapat menggantikan nilai default dari grup parameter klaster. Misalnya, jika satu instans dalam klaster mengalami masalah, Anda dapat menetapkan grup parameter DB kustom ke instans tersebut. Grup parameter kustom mungkin memiliki pengaturan tertentu untuk parameter yang terkait dengan debugging atau penyetelan performa.

Aurora menetapkan grup parameter default saat Anda membuat klaster atau instans DB baru, berdasarkan mesin dan versi basis data yang ditentukan. Sebagai gantinya, Anda dapat menentukan grup parameter kustom. Anda membuat sendiri grup parameter tersebut, dan Anda dapat mengedit nilai parameter. Anda dapat menentukan grup parameter kustom ini saat pembuatan. Anda juga dapat memodifikasi klaster atau instans DB nanti untuk menggunakan grup parameter kustom.

Untuk instans terprovisi dan instans Aurora Serverless v2, nilai konfigurasi apa pun yang dimodifikasi di grup parameter klaster DB akan mengganti nilai default dalam grup parameter DB. Jika Anda mengedit nilai yang sesuai dalam grup parameter DB, nilai tersebut akan mengganti pengaturan dari grup parameter klaster DB.

Pengaturan parameter DB apa pun yang Anda modifikasi akan diprioritaskan daripada nilai grup parameter klaster DB, bahkan jika Anda mengubah parameter konfigurasi kembali ke nilai default-nya. Anda dapat melihat parameter mana yang diganti dengan menggunakan describe-db-parameters AWS CLI perintah atau operasi Deskripsikan DBParameters RDS API. Bidang `Source` berisi nilai `user` jika Anda telah memodifikasi parameter tersebut. Untuk mengatur ulang satu atau beberapa parameter sehingga nilai dari grup parameter cluster DB diutamakan, gunakan reset-db-parameter-group AWS CLI perintah atau operasi Reset DBParameter Group RDS API.

Parameter instans DB dan klaster DB yang tersedia untuk Anda di Aurora bervariasi, bergantung pada kompatibilitas mesin basis data.

| Mesin basis data | Parameter |
| --- | --- |
| Aurora MySQL | Lihat Parameter konfigurasi Aurora MySQL.<br><br>Untuk klaster Aurora Serverless, lihat detail tambahan dalam Menggunakan grup parameter untuk Aurora Serverless v2 dan Kelompok parameter untuk Aurora Serverless v1. |
| Aurora PostgreSQL | Lihat Parameter Amazon Aurora PostgreSQL.<br><br>Untuk klaster Aurora Serverless, lihat detail tambahan dalam Menggunakan grup parameter untuk Aurora Serverless v2 dan Kelompok parameter untuk Aurora Serverless v1. |

> **Note**
>
> Klaster Aurora Serverless v1 hanya memiliki grup parameter klaster DB, bukan grup parameter DB. Untuk klaster Aurora Serverless v2, Anda membuat semua perubahan pada parameter kustom dalam grup parameter klaster DB.
> Aurora Serverless v2 menggunakan grup parameter klaster DB dan grup parameter DB. Dengan Aurora Serverless v2, Anda dapat memodifikasi hampir semua parameter konfigurasi. Aurora Serverless v2 mengganti pengaturan beberapa parameter konfigurasi terkait kapasitas, sehingga beban kerja Anda tidak terganggu saat instans Aurora Serverless v2 diturunkan skalanya.

Untuk mempelajari selengkapnya tentang pengaturan konfigurasi Aurora Serverless dan pengaturan mana yang dapat dimodifikasi, lihat Menggunakan grup parameter untuk Aurora Serverless v2 dan Kelompok parameter untuk Aurora Serverless v1.

## Membuat grup parameter cluster DB di Amazon Aurora

Anda dapat membuat grup parameter cluster DB baru menggunakan AWS Management Console, AWS CLI, atau RDS API.

Setelah Anda membuat grup parameter klaster DB, tunggu minimal 5 menit sebelum membuat klaster DB yang menggunakan grup parameter klaster DB tersebut. Dengan demikian, Amazon RDS dapat sepenuhnya membuat grup parameter sebelum digunakan oleh klaster DB baru. Anda dapat menggunakan halaman grup Parameter di konsol Amazon RDS atau describe-db-cluster-parametersperintah untuk memverifikasi bahwa grup parameter cluster DB Anda dibuat.

Batasan berikut berlaku untuk nama grup parameter klaster DB:

- Nama harus berisi 1 sampai 255 huruf, angka, atau tanda hubung.

  Nama grup parameter default boleh menyertakan titik, seperti `default.aurora-mysql5.7`. Namun, nama grup parameter kustom tidak boleh menyertakan titik.
- Karakter pertama harus berupa huruf.
- Nama tidak boleh diakhiri dengan tanda hubung atau berisi dua tanda hubung berturut-turut.

### Konsol

Untuk membuat grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Pilih Buat grup parameter.
4. Untuk nama grup Parameter, masukkan nama grup parameter cluster DB baru.
5. Untuk Deskripsi, masukkan deskripsi untuk grup parameter cluster DB baru.
6. Untuk jenis Engine, pilih mesin database Anda.
7. Untuk keluarga grup Parameter, pilih keluarga grup parameter DB.

8. Pilih Buat.

AWS CLI

Untuk membuat grup parameter cluster DB, gunakan AWS CLI `create-db-cluster-parameter-group`perintah.

Contoh berikut membuat grup parameter klaster DB yang bernama mydbclusterparametergroup untuk Aurora MySQL versi 5.7 dengan deskripsi "My new cluster parameter group".

Sertakan parameter wajib berikut:

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

Untuk menampilkan daftar semua kumpulan grup parameter yang tersedia, gunakan perintah berikut:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> Output berisi duplikat.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

Untuk Windows:

```
aws rds create-db-cluster-parameter-group ^
```

```
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

Perintah ini menghasilkan output seperti yang berikut ini:

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
}
```

API RDS

Untuk membuat grup parameter klaster DB, gunakan tindakan `CreateDBClusterParameterGroup` API RDS.

Sertakan parameter-parameter yang diperlukan berikut:

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

## Mengaitkan grup parameter cluster DB dengan cluster DB di Amazon Aurora

Anda dapat membuat sendiri grup parameter klaster DB dengan pengaturan yang disesuaikan. Anda dapat mengaitkan grup parameter cluster DB dengan cluster DB menggunakan AWS Management Console, the AWS CLI, atau file RDSAPI. Anda dapat melakukannya saat membuat atau memodifikasi klaster DB.

Untuk informasi tentang cara membuat grup parameter klaster DB, lihat Membuat grup parameter cluster DB di Amazon Aurora. Untuk informasi tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora. Untuk informasi tentang mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

> **Note**
>
> Untuk Aurora Postgre SQL 15.2, 14.7, 13.10, 12.14, dan semua 11 versi, ketika Anda mengubah grup parameter cluster DB yang terkait dengan cluster DB, reboot setiap instance replika untuk menerapkan perubahan.
> Untuk menentukan apakah instance DB utama dari cluster DB harus di-boot ulang untuk menerapkan perubahan, jalankan perintah berikut: AWS CLI
> `aws rds describe-db-clusters --db-cluster-identifier` *`db_cluster_identifier`*
> Periksa nilai `DBClusterParameterGroupStatus` untuk instans DB primer dalam output. Jika nilainya adalah `pending-reboot`, boot ulang instans DB primer dari klaster DB.

Konsol

Untuk mengaitkan grup parameter klaster DB dengan klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin Anda modifikasi.
3. Pilih Ubah. Halaman Modifikasi klaster DB akan muncul.
4. Ubah pengaturan Grup parameter klaster DB.
5. Pilih Lanjutkan dan periksa ringkasan modifikasi.

   Perubahan akan langsung diterapkan, terlepas dari pengaturan Penjadwalan modifikasi.
6. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Ubah klaster untuk menyimpan perubahan Anda.

   Alternatifnya, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan Anda.

AWS CLI

Untuk mengaitkan grup parameter cluster DB dengan cluster DB, gunakan AWS CLI `modify-db-cluster`perintah dengan opsi berikut:

- `--db-cluster-name`

- `--db-cluster-parameter-group-name`

Contoh berikut mengaitkan grup parameter DB `mydbclpg` dengan klaster DB `mydbcluster`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS API

Untuk mengaitkan grup parameter cluster DB dengan cluster DB, gunakan RDS API `ModifyDBCluster`operasi dengan parameter berikut:

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

## Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora

Anda dapat mengubah nilai parameter dalam grup parameter klaster DB buatan pelanggan. Anda tidak dapat mengubah nilai parameter dalam grup parameter klaster DB default. Perubahan pada parameter dalam grup parameter klaster DB buatan pelanggan diterapkan ke semua klaster DB yang dikaitkan dengan grup parameter klaster DB.

Konsol

Untuk mengubah grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

3. Dalam daftar, pilih grup parameter yang ingin Anda modifikasi.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Ubah nilai parameter yang ingin Anda modifikasi. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengubah nilai di grup parameter default.
6. Pilih Simpan perubahan.
7. Reboot instance DB utama (penulis) di cluster untuk menerapkan perubahan padanya.
8. Kemudian reboot instans DB pembaca untuk menerapkan perubahan pada mereka.

   Jika Anda tidak me-reboot instans DB, maka operasi failover bisa memakan waktu lebih lama dari biasanya.

AWS CLI

Untuk memodifikasi grup parameter cluster DB, gunakan AWS CLI `modify-db-cluster-parameter-group`perintah dengan parameter yang diperlukan berikut:

- `--db-cluster-parameter-group-name`
- `--parameters`

Contoh berikut memodifikasi nilai `server_audit_logging` dan `server_audit_logs_upload` dalam grup parameter klaster DB yang bernama mydbclusterparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
```

```
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Perintah tersebut menghasilkan output seperti berikut ini:

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

API RDS

Untuk memodifikasi grup parameter klaster DB, gunakan perintah `ModifyDBClusterParameterGroup` API RDS dengan parameter wajib berikut:

- `DBClusterParameterGroupName`
- `Parameters`

## Menyetel ulang parameter dalam grup parameter cluster DB di Amazon Aurora

Anda dapat mengatur ulang parameter ke nilai default-nya dalam grup parameter klaster DB yang dibuat pelanggan. Perubahan pada parameter dalam grup parameter klaster DB buatan pelanggan diterapkan ke semua klaster DB yang dikaitkan dengan grup parameter klaster DB.

> **Note**
>
> Dalam grup parameter klaster DB default, parameter selalu diatur ke nilai default-nya.

Konsol

Untuk mengatur ulang parameter dalam grup parameter klaster DB ke nilai default-nya

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter.
4. Untuk Tindakan grup parameter, pilih Edit.

5. Pilih parameter yang ingin Anda atur ulang ke nilai default-nya. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengatur ulang nilai dalam grup parameter default.
6. Pilih Atur ulang lalu konfirmasikan dengan memilih Atur ulang parameter.
7. Reboot instans DB utama di cluster DB untuk menerapkan perubahan ke semua instance DB di cluster DB.

AWS CLI

Untuk mengatur ulang parameter dalam grup parameter cluster DB ke nilai defaultnya, gunakan AWS CLI `reset-db-cluster-parameter-group`perintah dengan opsi wajib berikut:`--db-cluster-parameter-group-name`.

Untuk mengatur ulang semua parameter dalam grup parameter klaster DB, tentukan opsi `--reset-all-parameters`. Untuk mengatur ulang parameter tertentu, tentukan opsi `--parameters`.

Contoh berikut mengatur ulang semua parameter dalam grup parameter DB yang bernama mydbparametergroup ke nilai default-nya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Untuk Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

Contoh berikut mengatur ulang `server_audit_logging` dan `server_audit_logs_upload` ke nilai default-nya dalam grup parameter klaster DB yang bernama mydbclusterparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Perintah tersebut menghasilkan output seperti berikut ini:

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

API RDS

Untuk mengatur ulang parameter dalam grup parameter klaster DB ke nilai default-nya, gunakan perintah `ResetDBClusterParameterGroup` API RDS dengan parameter wajib berikut: `DBClusterParameterGroupName`.

Untuk mengatur ulang semua parameter dalam grup parameter klaster DB, atur parameter `ResetAllParameters` ke `true`. Untuk mengatur ulang parameter tertentu, tentukan parameter `Parameters`.

## Menyalin grup parameter cluster DB di Amazon Aurora

Anda dapat menyalin grup parameter klaster DB kustom yang Anda buat. Menyalin grup parameter adalah solusi yang mudah jika Anda sudah membuat grup parameter klaster DB dan Anda ingin menyertakan sebagian besar parameter dan nilai kustom dari grup tersebut dalam grup parameter klaster DB yang baru. Anda dapat menyalin grup parameter cluster DB dengan menggunakan perintah AWS CLI copy-db-cluster-parameter-group atau operasi RDS API Copy DBCluster ParameterGroup.

Setelah Anda membuat grup parameter klaster DB, tunggu minimal 5 menit sebelum membuat klaster DB yang menggunakan grup parameter klaster DB tersebut. Dengan demikian, Amazon

RDS dapat sepenuhnya menyalin grup parameter sebelum digunakan oleh klaster DB baru. Anda dapat menggunakan halaman grup Parameter di konsol Amazon RDS atau describe-db-cluster-parametersperintah untuk memverifikasi bahwa grup parameter cluster DB Anda dibuat.

> **Note**
>
> Anda tidak dapat menyalin grup parameter default. Namun, Anda dapat membuat grup parameter baru yang didasarkan pada grup parameter default.
> Anda tidak dapat menyalin grup parameter cluster DB ke grup yang berbeda Akun AWS atau Wilayah AWS.

Konsol

Untuk menyalin grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter kustom yang ingin Anda salin.
4. Untuk Tindakan grup parameter, pilih Salin.
5. Di Pengidentifikasi grup parameter DB baru, masukkan nama untuk grup parameter baru.
6. Di Deskripsi, masukkan deskripsi untuk grup parameter baru.
7. Pilih Salin.

AWS CLI

Untuk menyalin grup parameter cluster DB, gunakan AWS CLI `copy-db-cluster-parameter-group`perintah dengan parameter yang diperlukan berikut:

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

Contoh berikut membuat grup parameter klaster DB baru yang bernama `mygroup2` yang merupakan salinan dari grup parameter klaster DB `mygroup1`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

Untuk Windows:

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

API RDS

Untuk menyalin grup parameter klaster DB, gunakan operasi `CopyDBClusterParameterGroup` API RDS dengan parameter wajib berikut:

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

## Daftar grup parameter cluster DB di Amazon Aurora

Anda dapat mencantumkan grup parameter cluster DB yang telah Anda buat untuk AWS akun Anda.

> **Note**
>
> Grup parameter default secara otomatis dibuat dari templat parameter default ketika Anda membuat klaster DB untuk mesin dan versi DB tertentu. Grup parameter default ini berisi pengaturan parameter pilihan dan tidak dapat dimodifikasi. Saat membuat grup parameter kustom, Anda dapat memodifikasi pengaturan parameter.

Konsol

Untuk mencantumkan semua grup parameter cluster DB untuk AWS akun

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter klaster DB muncul dalam daftar dengan Grup parameter klaster DB untuk Jenis.

AWS CLI

Untuk mencantumkan semua grup parameter cluster DB untuk AWS akun, gunakan AWS CLI `describe-db-cluster-parameter-groups`perintah.

Example

Contoh berikut menampilkan daftar semua grup parameter klaster DB yang tersedia untuk akun AWS .

```
aws rds describe-db-cluster-parameter-groups
```

Contoh berikut menjelaskan grup parameter mydbclusterparametergroup.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Untuk Windows:

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Perintah ini menampilkan respons seperti berikut:

```
{
    "DBClusterParameterGroups": [
        {
```

```
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

API RDS

Untuk mencantumkan semua grup parameter cluster DB untuk AWS akun, gunakan `DescribeDBClusterParameterGroups`tindakan RDS API.

## Melihat nilai parameter untuk grup parameter cluster DB di Amazon Aurora

Anda dapat memperoleh daftar semua parameter dalam grup parameter klaster DB dan nilainya.

Konsol

Untuk melihat nilai parameter untuk grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter klaster DB muncul dalam daftar dengan Grup parameter klaster DB untuk Jenis.
3. Pilih nama grup parameter klaster DB untuk melihat daftar parameternya.

AWS CLI

Untuk melihat nilai parameter untuk grup parameter cluster DB, gunakan AWS CLI `describe-db-cluster-parameters`perintah dengan parameter yang diperlukan berikut.

- `--db-cluster-parameter-group-name`

Example

Contoh berikut menampilkan daftar parameter dan nilai parameter untuk grup parameter klaster DB yang bernama mydbclusterparametergroup, dalam format JSON.

Perintah ini menampilkan respons seperti berikut:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```


```
{
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```


### API RDS

Untuk melihat nilai parameter untuk grup parameter klaster DB, gunakan perintah API RDS `DescribeDBClusterParameters` dengan parameter wajib berikut.

- `DBClusterParameterGroupName`

Dalam beberapa kasus, nilai yang diizinkan untuk parameter tidak ditampilkan. Parameter ini selalu merupakan parameter yang sumbernya adalah mesin basis data default.

Untuk melihat nilai-nilai parameter ini, Anda dapat menjalankan pernyataan SQL berikut:

- MySQL:

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
mysql$ SHOW VARIABLES;
```

- PostgreSQL:

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

## Menghapus grup parameter cluster DB di Amazon Aurora

Anda dapat menghapus grup parameter cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API. Grup parameter grup parameter cluster DB memenuhi syarat untuk dihapus hanya jika tidak terkait dengan cluster DB.

Konsol

Untuk menghapus grup parameter

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter muncul dalam daftar.
3. Pilih nama grup parameter cluster DB yang akan dihapus.

4. Pilih Tindakan dan kemudian Hapus.
5. Tinjau nama grup parameter dan kemudian pilih Hapus.

AWS CLI

Untuk menghapus grup parameter cluster DB, gunakan AWS CLI `delete-db-cluster-parameter-group`perintah dengan parameter yang diperlukan berikut.

- `--db-parameter-group-name`

Example

Contoh berikut menghapus kelompok parameter cluster DB bernama mydbparametergroup.

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

API RDS

Untuk menghapus grup parameter cluster DB, gunakan `DeleteDBClusterParameterGroup`perintah RDS API dengan parameter wajib berikut.

- `DBParameterGroupName`

## Grup parameter DB untuk instans Aurora DB Amazon

Instans DB menggunakan grup parameter DB. Bagian berikut menjelaskan konfigurasi dan pengelolaan grup parameter instans DB.

Topik

- Membuat grup parameter DB di Aurora
- Mengaitkan grup parameter DB dengan instans DB di Aurora
- Memodifikasi parameter dalam grup parameter DB di
- Menyetel ulang parameter dalam grup parameter DB ke nilai defaultnya di Aurora
- Menyalin grup parameter DB di Aurora
- Daftar grup parameter DB di Aurora
- Melihat nilai parameter untuk grup parameter DB di Aurora

- Menghapus grup parameter DB di Aurora

## Membuat grup parameter DB di Aurora

Anda dapat membuat grup parameter DB baru menggunakan AWS CLI,, atau RDSAPI. AWS Management Console

Batasan berikut berlaku untuk grup parameter DB:

- Nama harus berisi 1 sampai 255 huruf, angka, atau tanda hubung.

  Nama grup parameter default boleh menyertakan titik, seperti `default.mysql8.0`. Namun, nama grup parameter kustom tidak boleh menyertakan titik.
- Karakter pertama harus berupa huruf.
- Nama tidak boleh diakhiri dengan tanda hubung atau berisi dua tanda hubung berturut-turut.

### Konsol

Untuk membuat grup parameter DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Pilih Buat grup parameter.
4. Untuk nama grup Parameter, masukkan nama grup parameter DB baru Anda.
5. Untuk Deskripsi, masukkan deskripsi untuk grup parameter DB baru Anda.
6. Untuk jenis Engine, pilih mesin DB Anda.
7. Untuk keluarga grup Parameter, pilih keluarga grup parameter DB.
8. Untuk Jenis, jika berlaku, pilih Grup Parameter DB.
9. Pilih Buat.

### AWS CLI

Untuk membuat grup parameter DB, gunakan AWS CLI `create-db-parameter-group`perintah. Contoh berikut membuat grup parameter DB bernama mydbparametergroup untuk SQL versi Saya 8.0 dengan deskripsi "Grup parameter baru saya.”

Sertakan parameter-parameter yang diperlukan berikut:

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

Untuk menampilkan daftar semua kumpulan grup parameter yang tersedia, gunakan perintah berikut:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> Output berisi duplikat.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new parameter group"
```

Untuk Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

Perintah ini menghasilkan output seperti yang berikut ini:

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

RDS API

Untuk membuat grup parameter DB, gunakan RDS API `CreateDBParameterGroup`operasi.

Sertakan parameter-parameter yang diperlukan berikut:

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

## Mengaitkan grup parameter DB dengan instans DB di Aurora

Anda dapat membuat grup parameter DB Anda sendiri dengan pengaturan yang disesuaikan. Anda dapat mengaitkan grup parameter DB dengan instans DB menggunakan AWS CLI,, atau RDSAPI. AWS Management Console Anda dapat melakukannya saat membuat atau memodifikasi instans DB.

Untuk informasi tentang cara membuat grup parameter DB, lihat Membuat grup parameter DB di Aurora. Untuk informasi tentang mengubah instans DB, lihat Memodifikasi instans DB dalam klaster DB.

> **Note**
>
> Ketika Anda mengaitkan grup parameter DB baru dengan instans DB, parameter statis dan dinamis yang dimodifikasi akan diterapkan hanya setelah instans DB di-boot ulang. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini diterapkan segera tanpa boot ulang.

Konsol

Untuk mengaitkan grup parameter DB dengan instans DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, kemudian pilih instans DB yang ingin diubah.
3. Pilih Ubah. Halaman Modifikasi instans DB akan muncul.
4. Ubah pengaturan Grup parameter DB.
5. Pilih Lanjutkan dan periksa ringkasan modifikasi.
6. (Opsional) Pilih Terapkan segera untuk segera menerapkan perubahan. Dalam beberapa kasus, pemadaman dapat terjadi jika opsi ini dipilih.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Modifikasi instans DB untuk menyimpan perubahan Anda.

Atau pilih Kembali untuk mengedit perubahan atau Batalkan untuk membatalkan perubahan Anda.

AWS CLI

Untuk mengaitkan grup parameter DB dengan instans DB, gunakan AWS CLI `modify-db-instance`perintah dengan opsi berikut:

- `--db-instance-identifier`
- `--db-parameter-group-name`

Contoh berikut mengaitkan grup parameter DB `mydbpg` dengan Pembaruan DB `database-1`. Perubahan diterapkan segera dengan menggunakan `--apply-immediately`. Gunakan `--no-apply-immediately` untuk menerapkan perubahan pada periode pemeliharaan berikutnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS API

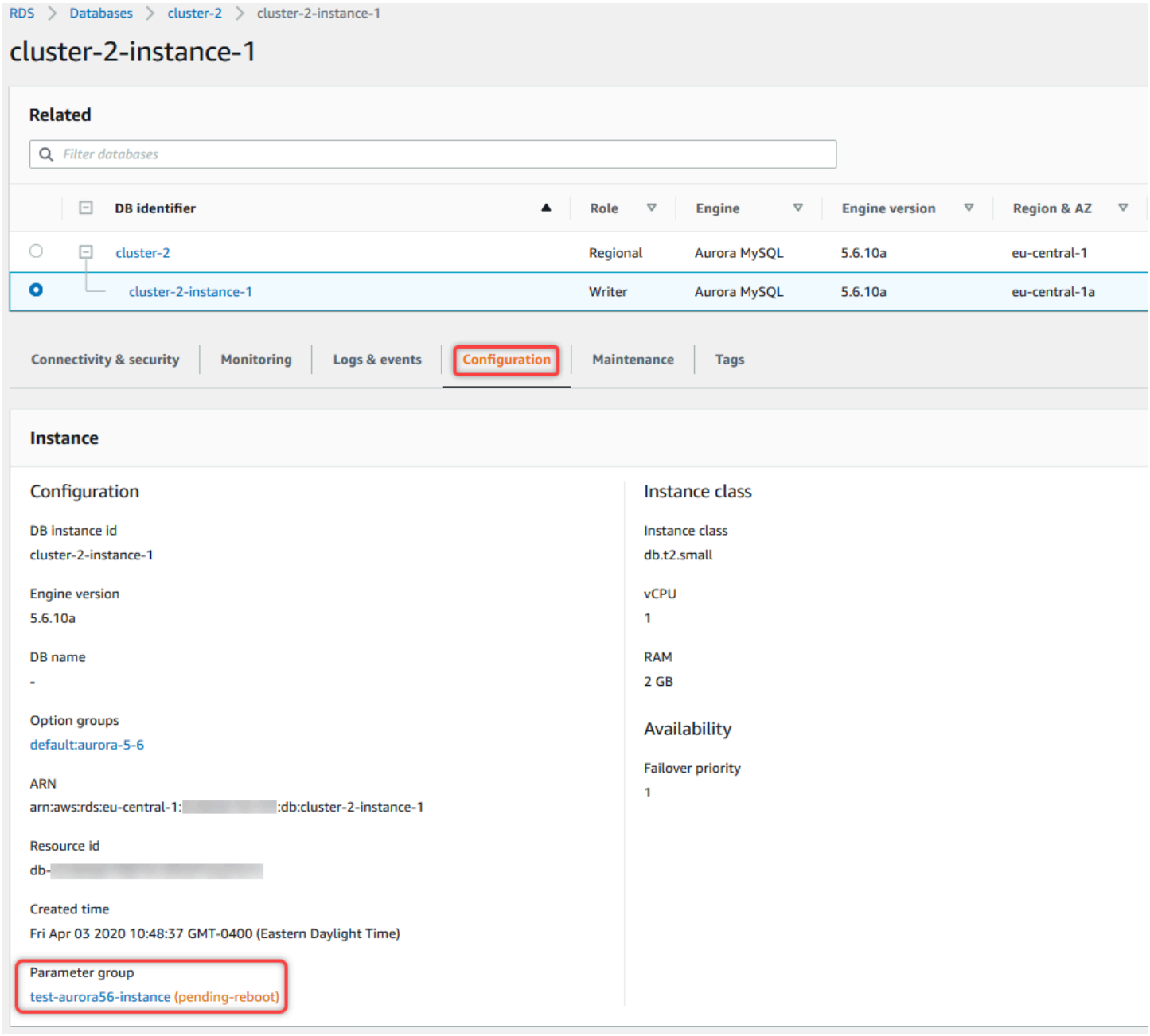
Untuk mengaitkan grup parameter DB dengan instans DB, gunakan RDS API `ModifyDBInstance`operasi dengan parameter berikut:

- `DBInstanceName`
- `DBParameterGroupName`

## Memodifikasi parameter dalam grup parameter DB di

Anda dapat memodifikasi nilai parameter di grup parameter DB yang dibuat pelanggan; Anda tidak dapat mengubah nilai parameter di grup parameter DB default. Perubahan parameter dalam grup parameter DB yang dibuat pelanggan diterapkan ke semua instans DB yang dikaitkan dengan grup parameter DB.

Perubahan pada beberapa parameter diterapkan ke instans DB segera tanpa boot ulang. Perubahan pada parameter lain diterapkan hanya setelah instans DB di-boot ulang. Konsol RDS menunjukkan status grup parameter DB yang terkait dengan instans DB pada tab Konfigurasi. Misalnya, Pembaruan DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait. Jika demikian, konsol RDS menunjukkan grup parameter DB dengan status boot ulang tertunda. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, lakukan boot ulang secara manual terhadap instans DB. Kami menyarankan Anda untuk selalu me-reboot instans DB setelah memodifikasi parameter. Jika Anda tidak me-reboot instans DB, maka operasi failover bisa memakan waktu lebih lama dari biasanya.




Konsol

Untuk memodifikasi parameter dalam kelompok parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih nama grup parameter yang ingin Anda ubah.
4. Untuk Tindakan grup parameter, pilih Edit.

5. Ubah nilai parameter yang ingin Anda modifikasi. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengubah nilai di grup parameter default.

6. Pilih Simpan perubahan.

AWS CLI

Untuk memodifikasi grup parameter DB, gunakan AWS CLI `modify-db-parameter-group`perintah dengan opsi yang diperlukan berikut:

- `--db-parameter-group-name`
- `--parameters`

Contoh berikut memodifikasi nilai `max_connections` dan `max_allowed_packet` dalam grup parameter DB yang bernama mydbparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Perintah tersebut menghasilkan output seperti berikut ini:

```
DBPARAMETERGROUP  mydbparametergroup
```

API RDS

Untuk memodifikasi grup parameter DB, gunakan operasi `ModifyDBParameterGroup` API RDS dengan parameter wajib berikut ini:

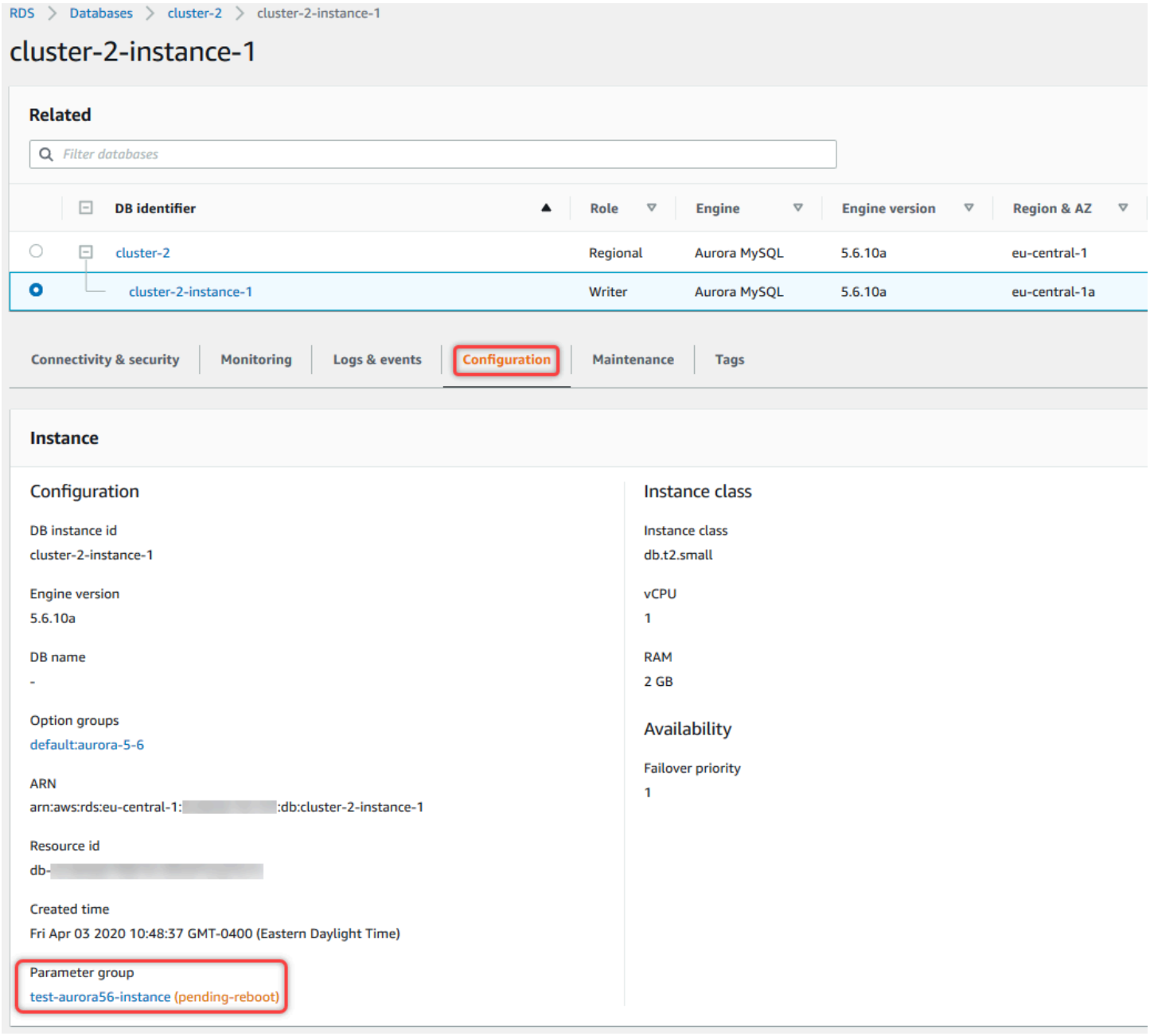
- `DBParameterGroupName`
- `Parameters`

## Menyetel ulang parameter dalam grup parameter DB ke nilai defaultnya di Aurora

Anda dapat mengatur ulang parameter dalam grup parameter DB yang dibuat pelanggan ke nilai default-nya. Perubahan parameter dalam grup parameter DB yang dibuat pelanggan diterapkan ke semua instans DB yang dikaitkan dengan grup parameter DB.

Saat Anda menggunakan konsol, Anda dapat mengatur ulang parameter tertentu ke nilai default-nya. Namun, Anda tidak dapat dengan mudah mengatur ulang semua parameter dalam grup parameter DB secara sekaligus. Saat Anda menggunakan AWS CLI atau RDSAPI, Anda dapat mengatur ulang parameter tertentu ke nilai defaultnya. Anda juga dapat mengatur ulang semua parameter dalam grup parameter DB secara sekaligus.

Perubahan pada beberapa parameter diterapkan langsung ke instans DB tanpa boot ulang. Perubahan pada parameter lain diterapkan hanya setelah instans DB di-boot ulang. RDSKonsol menunjukkan status grup parameter DB yang terkait dengan instans DB pada tab Konfigurasi. Misalnya, Pembaruan DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait. Jika demikian, RDS konsol menampilkan grup parameter DB dengan status pending-reboot. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, lakukan boot ulang secara manual terhadap instans DB.




> **Note**
>
> Dalam grup parameter DB default, parameter selalu diatur ke nilai default-nya.

Konsol

Untuk mengatur ulang parameter dalam grup parameter DB ke nilai default-nya

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Pilih parameter yang ingin Anda atur ulang ke nilai default-nya. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengatur ulang nilai dalam grup parameter default.
6. Pilih Atur ulang lalu konfirmasikan dengan memilih Atur ulang parameter.

AWS CLI

Untuk mengatur ulang beberapa atau semua parameter dalam grup parameter DB, gunakan AWS CLI `reset-db-parameter-group`perintah dengan opsi wajib berikut:`--db-parameter-group-name`.

Untuk mengatur ulang semua parameter dalam grup parameter DB, tentukan opsi `--reset-all-parameters`. Untuk mengatur ulang parameter tertentu, tentukan opsi `--parameters`.

Contoh berikut mengatur ulang semua parameter dalam grup parameter DB yang bernama mydbparametergroup ke nilai default-nya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Untuk Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
```

```
    --reset-all-parameters
```

Contoh berikut mengatur ulang opsi `max_connections` dan `max_allowed_packet` ke nilai default-nya dalam grup parameter DB yang bernama mydbparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Perintah tersebut menghasilkan output seperti berikut ini:

```
DBParameterGroupName  mydbparametergroup
```

RDS API

Untuk mengatur ulang parameter dalam grup parameter DB ke nilai defaultnya, gunakan RDS API `ResetDBParameterGroup`perintah dengan parameter yang diperlukan berikut:`DBParameterGroupName`.

Untuk mengatur ulang semua parameter dalam grup parameter DB, atur parameter `ResetAllParameters` ke `true`. Untuk mengatur ulang parameter tertentu, tentukan parameter `Parameters`.

## Menyalin grup parameter DB di Aurora

Anda dapat menyalin grup parameter DB kustom yang Anda buat. Menyalin grup parameter bisa menjadi solusi yang mudah. Contohnya adalah ketika Anda telah membuat grup parameter DB dan ingin menyertakan sebagian besar parameter dan nilai kustomnya dalam grup parameter DB baru. Anda dapat menyalin grup parameter DB dengan menggunakan file AWS Management

Console. Anda juga dapat menggunakan AWS CLI copy-db-parameter-groupperintah atau operasi opyDBParameterGrup RDS API C.

Setelah Anda menyalin grup parameter DB, tunggu minimal 5 menit sebelum membuat instans DB pertama Anda yang menggunakan grup parameter DB tersebut sebagai grup parameter default. Melakukan hal ini memungkinkan Amazon RDS untuk sepenuhnya menyelesaikan tindakan salin sebelum grup parameter digunakan. Tindakan ini harus dilakukan terutama untuk parameter yang sangat penting saat membuat basis data default untuk instans DB. Contohnya adalah set karakter untuk basis data default yang ditentukan oleh parameter `character_set_database`. Gunakan opsi Parameter Grup RDS konsol Amazon atau describe-db-parametersperintah untuk memverifikasi bahwa grup parameter DB Anda dibuat.

> **Note**
>
> Anda tidak dapat menyalin grup parameter default. Namun, Anda dapat membuat grup parameter baru yang didasarkan pada grup parameter default.
> Anda tidak dapat menyalin grup parameter DB ke grup parameter lain Akun AWS atau Wilayah AWS.

Konsol

Untuk menyalin grup parameter DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter kustom yang ingin Anda salin.
4. Untuk Tindakan grup parameter, pilih Salin.
5. Di Pengidentifikasi grup parameter DB baru, masukkan nama untuk grup parameter baru.
6. Di Deskripsi, masukkan deskripsi untuk grup parameter baru.
7. Pilih Salin.

AWS CLI

Untuk menyalin grup parameter DB, gunakan AWS CLI `copy-db-parameter-group`perintah dengan opsi yang diperlukan berikut:

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

Contoh berikut membuat grup parameter DB baru yang bernama `mygroup2` yang merupakan salinan grup parameter DB `mygroup1`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

Untuk Windows:

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS API

Untuk menyalin grup parameter DB, gunakan RDS API `CopyDBParameterGroup`operasi dengan parameter yang diperlukan berikut:

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

## Daftar grup parameter DB di Aurora

Anda dapat mencantumkan grup parameter DB yang telah Anda buat untuk AWS akun Anda.

> **Note**
>
> Grup parameter default secara otomatis dibuat dari template parameter default ketika Anda membuat instans DB untuk mesin dan versi DB tertentu. Grup parameter default ini berisi pengaturan parameter pilihan dan tidak dapat dimodifikasi. Saat membuat grup parameter kustom, Anda dapat memodifikasi pengaturan parameter.

Konsol

Untuk mencantumkan semua grup parameter DB untuk AWS akun

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter DB muncul dalam daftar.

AWS CLI

Untuk membuat daftar semua grup parameter DB untuk AWS akun, gunakan AWS CLI `describe-db-parameter-groups`perintah.

Example

Contoh berikut menampilkan daftar semua grup parameter DB yang tersedia untuk akun AWS .

```
aws rds describe-db-parameter-groups
```

Perintah ini menampilkan respons seperti berikut:

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

Contoh berikut menjelaskan grup parameter mydbparamgroup1.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

Untuk Windows:

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

Perintah ini menampilkan respons seperti berikut:

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS API

Untuk mencantumkan semua grup parameter DB untuk AWS akun, gunakan RDS API `DescribeDBParameterGroups`operasi.

## Melihat nilai parameter untuk grup parameter DB di Aurora

Anda dapat memperoleh daftar semua parameter dalam grup parameter DB dan nilainya.

Konsol

Untuk melihat nilai parameter untuk grup parameter DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter DB muncul dalam daftar.
3. Pilih nama grup parameter untuk melihat daftar parameternya.

AWS CLI

Untuk melihat nilai parameter untuk grup parameter DB, gunakan AWS CLI `describe-db-parameters`perintah dengan parameter yang diperlukan berikut.

- `--db-parameter-group-name`

Example

Contoh berikut menampilkan daftar parameter dan nilai parameter untuk grup parameter DB yang bernama mydbparametergroup.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

Perintah ini menampilkan respons seperti berikut:

```
DBPARAMETER  Parameter Name            Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
DBPARAMETER  allow-suspicious-udfs                      engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                   engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                      engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size         32768            system           integer
 dynamic     true
DBPARAMETER  socket                    /tmp/mysql.sock  system           string
 static      false
```

RDS API

Untuk melihat nilai parameter untuk grup parameter DB, gunakan RDS API `DescribeDBParameters`perintah dengan parameter yang diperlukan berikut.

- `DBParameterGroupName`

## Menghapus grup parameter DB di Aurora

Anda dapat menghapus grup parameter DB menggunakan AWS Management Console, AWS CLI, atau RDSAPI. Grup parameter memenuhi syarat untuk dihapus hanya jika tidak terkait dengan instans DB.

Konsol

Untuk menghapus grup parameter DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter DB muncul dalam daftar.
3. Pilih nama grup parameter yang akan dihapus.

4. Pilih Tindakan dan kemudian Hapus.
5. Tinjau nama grup parameter dan kemudian pilih Hapus.

AWS CLI

Untuk menghapus grup parameter DB, gunakan AWS CLI `delete-db-parameter-group`perintah dengan parameter yang diperlukan berikut.

- `--db-parameter-group-name`

Example

Contoh berikut menghapus kelompok parameter DB bernama mydbparametergroup.

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

Untuk menghapus grup parameter DB, gunakan RDS API `DeleteDBParameterGroup`perintah dengan parameter yang diperlukan berikut.

- `DBParameterGroupName`

## Membandingkan grup parameter DB

Anda dapat menggunakan AWS Management Console untuk melihat perbedaan antara dua kelompok parameter DB.

Grup parameter yang ditentukan harus berupa grup parameter DB, atau keduanya harus grup parameter klaster DB. Hal ini berlaku meskipun mesin dan versi DB sama. Misalnya, Anda tidak dapat membandingkan grup parameter DB `aurora-mysql8.0` (Aurora My SQL version 3) dan grup parameter cluster `aurora-mysql8.0` DB.

Anda dapat membandingkan Aurora My SQL dan RDS untuk grup parameter My SQL DB, bahkan untuk versi yang berbeda, tetapi Anda tidak dapat membandingkan Aurora Postgre SQL dan RDS untuk grup parameter Postgre DB. SQL

Untuk membandingkan dua kelompok parameter DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter yang ingin Anda bandingkan.

   > **Note**
   >
   > Untuk membandingkan grup parameter default dengan grup parameter kustom, pilih grup parameter default pada tab Default terlebih dahulu, lalu pilih grup parameter kustom pada tab Kustom.

4. Dari Tindakan, pilih Bandingkan.

## Menentukan parameter DB

Jenis parameter DB mencakup yang berikut ini:

- Bilangan Bulat
- Boolean
- String
- Panjang
- Ganda
- Stempel Waktu
- Objek dari tipe data lain yang ditentukan
- Rangkaian nilai tipe bilangan bulat, Boolean, string, panjang, ganda, stempel waktu, atau objek

Anda juga dapat menentukan parameter bilangan bulat dan Boolean menggunakan ekspresi, formula, dan fungsi.

Daftar Isi


- Formula parameter DB
  - Variabel formula parameter DB
  - Operator formula parameter DB

- Fungsi parameter DB
- Ekspresi log parameter DB
- Contoh nilai parameter DB

## Formula parameter DB

Formula parameter DB adalah ekspresi yang menghasilkan nilai bilangan bulat atau nilai Boolean. Anda mengapit ekspresi dalam kurung: {}. Anda dapat menentukan formula untuk nilai parameter DB atau sebagai argumen untuk fungsi parameter DB.

Sintaks

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

Variabel formula parameter DB

Setiap variabel formula menghasilkan nilai bilangan bulat atau Boolean. Nama variabel bersifat peka huruf besar-kecil.

AllocatedStorage

Mengembalikan bilangan bulat yang mewakili ukuran, dalam byte, dari volume data.

DBInstanceClassMemory

Mengembalikan bilangan bulat untuk jumlah byte memori yang tersedia untuk proses basis data. Angka ini dihitung secara internal dengan memulai dengan jumlah total memori untuk kelas instans DB. Dari sini, perhitungan mengurangi memori yang disediakan untuk sistem operasi dan RDS proses yang mengelola instance. Oleh karena itu, angkanya selalu agak lebih rendah dari angka memori yang ditunjukkan pada tabel kelas instans di Kelas instans Amazon Aurora DB. Nilai yang pasti bergantung pada kombinasi faktor. Ini termasuk kelas instance, mesin DB, dan apakah itu berlaku untuk RDS instance atau instance yang merupakan bagian dari cluster Aurora.

DBInstanceVCPU

Mengembalikan bilangan bulat yang mewakili jumlah unit pemrosesan pusat virtual (vCPUs) yang digunakan oleh Amazon RDS untuk mengelola instance.

EndPointPort

Mengembalikan bilangan bulat yang mewakili port yang digunakan saat terhubung ke instans DB.

TrueIfReplica

Mengembalikan 1 jika instans DB adalah replika baca dan 0 jika tidak. Ini adalah nilai default untuk `read_only` parameter di Aurora My. SQL

Operator formula parameter DB

Formula parameter DB mendukung dua operator: pembagian dan perkalian.

Operator pembagian: /

Membagi dividen dengan pembagi, mengembalikan hasil bagi bilangan bulat. Desimal dalam hasil bagi dipotong, tidak dibulatkan.

Sintaks

```
dividend / divisor
```

Argumen terbagi dan pembagi harus merupakan ekspresi bilangan bulat.

Operator perkalian: *

Mengalikan ekspresi, mengembalikan hasil ekspresi. Desimal dalam ekspresi dipotong, tidak dibulatkan.

Sintaks

```
expression * expression
```

Kedua ekspresi harus berupa bilangan bulat.

## Fungsi parameter DB

Anda menentukan argumen fungsi parameter DB sebagai bilangan bulat atau formula.
Setiap fungsi harus memiliki setidaknya satu argumen. Tentukan beberapa argumen sebagai daftar yang dipisahkan koma. Daftar ini tidak dapat memiliki anggota yang kosong, seperti argument1,,argument3. Nama fungsi bersifat peka huruf besar-kecil.

JIKA

Mengembalikan argumen.

Sintaks

```
IF(argument1, argument2, argument3)
```

Mengembalikan argumen kedua jika argumen pertama ternyata benar setelah dievaluasi. Mengembalikan argumen ketiga.

GREATEST

Mengembalikan nilai terbesar dari daftar bilangan bulat atau formula parameter.

Sintaks

```
GREATEST(argument1, argument2,...argumentn)
```

Mengembalikan bilangan bulat.

LEAST

Mengembalikan nilai terkecil dari daftar bilangan bulat atau formula parameter.

Sintaks

```
LEAST(argument1, argument2,...argumentn)
```

Mengembalikan bilangan bulat.

SUM

Menambahkan nilai bilangan bulat atau formula parameter yang ditentukan.

Sintaks

```
SUM(argument1, argument2,...argumentn)
```

Mengembalikan bilangan bulat.

## Ekspresi log parameter DB

Anda dapat mengatur nilai parameter DB bilangan bulat ke ekspresi log. Anda mengapit ekspresi dalam kurung: {}. Sebagai contoh:

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

Fungsi `log` mewakili dasar log 2. Contoh ini juga menggunakan variabel formula `DBInstanceClassMemory`. Lihat Variabel formula parameter DB.

## Contoh nilai parameter DB

Contoh-contoh ini menunjukkan penggunaan formula, fungsi, dan ekspresi untuk nilai parameter DB.

> **Warning**
>
> Pengaturan parameter yang tidak tepat dalam grup parameter DB dapat memiliki efek merugikan yang tidak diinginkan. Efek tersebut termasuk performa terdegradasi dan ketidakstabilan sistem. Selalu berhati-hati saat memodifikasi parameter basis data dan cadangkan data Anda sebelum memodifikasi grup parameter DB Anda. Cobalah perubahan grup parameter pada instance DB pengujian, yang dibuat menggunakan point-in-time-restores, sebelum menerapkan perubahan grup parameter tersebut ke instans DB produksi Anda.

Example menggunakan fungsi parameter DB LEAST

Anda dapat menentukan `LEAST` fungsi dalam nilai SQL `table_definition_cache` parameter Aurora My. Gunakan untuk mengatur jumlah definisi tabel yang dapat disimpan dalam cache definisi menjadi yang lebih kecil antara `DBInstanceClassMemory`/393040 atau 20.000.

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

# Memigrasikan data ke klaster DB Amazon Aurora

Anda memiliki beberapa opsi untuk memigrasikan data dari basis data yang sudah ada ke klaster DB Amazon Aurora, bergantung pada kompatibilitas mesin basis data. Opsi migrasi Anda juga bergantung pada basis data yang Anda migrasikan dan ukuran data yang Anda migrasikan.

## Memigrasi data ke klaster Amazon Aurora My DB SQL

Anda dapat memigrasikan data dari salah satu sumber berikut ke cluster Amazon Aurora SQL My DB.

- Sebuah RDS untuk instans SQL DB Saya
- SQLDatabase saya di luar Amazon RDS
- Database yang tidak kompatibel dengan saya SQL

Untuk informasi selengkapnya, lihat Migrasi data ke cluster Amazon Aurora My DB SQL.

## Migrasi data ke cluster DB Amazon Aurora SQL Postgre

Anda dapat memigrasikan data dari salah satu sumber berikut ke klaster DB Amazon Aurora SQL Postgre.

- Instans Amazon RDS Postgre DB SQL
- Database yang tidak kompatibel dengan Postgre SQL

Untuk informasi selengkapnya, lihat Migrasi data ke Amazon Aurora dengan kompatibilitas Postgre SQL.

# Membuat ElastiCache cache Amazon menggunakan pengaturan instans Amazon DB cluster Aurora RDS

ElastiCache adalah layanan caching dalam memori yang dikelola sepenuhnya yang menyediakan latensi baca dan tulis mikrodetik yang mendukung kasus penggunaan real-time yang fleksibel. ElastiCache dapat membantu Anda mempercepat kinerja aplikasi dan database. Anda dapat menggunakan ElastiCache sebagai penyimpanan data utama untuk kasus penggunaan yang tidak memerlukan daya tahan data, seperti papan peringkat game, streaming, dan analitik data. ElastiCache membantu menghilangkan kompleksitas yang terkait dengan penyebaran dan pengelolaan lingkungan komputasi terdistribusi. Untuk informasi selengkapnya, lihat Kasus ElastiCache Penggunaan Umum dan Bagaimana ElastiCache Dapat Membantu untuk Memcached dan Kasus ElastiCache Penggunaan Umum dan Bagaimana ElastiCache Dapat Membantu Redis. OSS Anda dapat menggunakan RDS konsol Amazon untuk membuat ElastiCache cache.

Anda dapat mengoperasikan Amazon ElastiCache dalam dua format. Anda dapat memulai dengan cache nirserver atau memilih untuk merancang klaster cache Anda sendiri. Jika Anda memilih untuk mendesain cluster cache Anda sendiri, ElastiCache bekerja dengan mesin Redis OSS dan Memcached. Jika Anda tidak yakin mesin mana yang ingin Anda gunakan, lihat Membandingkan Memcached dan RedisOSS. Untuk informasi selengkapnya tentang Amazon ElastiCache, lihat Panduan ElastiCache Pengguna Amazon.

Topik

- Ikhtisar pembuatan ElastiCache cache dengan pengaturan instans DB cluster Aurora RDS
-

## Ikhtisar pembuatan ElastiCache cache dengan pengaturan instans DB cluster Aurora RDS

Beberapa kasus penggunaan untuk mengaitkan ElastiCache cache dengan Anda:

- Anda dapat menghemat biaya dan meningkatkan kinerja Anda dengan menggunakan ElastiCache dengan RDS versus berjalan RDS sendiri.
- Anda dapat menggunakan ElastiCache cache sebagai penyimpanan data utama untuk aplikasi yang tidak memerlukan daya tahan data. Aplikasi Anda yang menggunakan Redis OSS atau Memcached dapat digunakan ElastiCache dengan hampir tidak ada modifikasi.

Saat Anda membuat ElastiCache cache dariRDS, ElastiCache cache mewarisi pengaturan berikut dari instance DB Aurora DB terkait:

- ElastiCache pengaturan konektivitas
- ElastiCache pengaturan keamanan

Anda juga dapat mengatur pengaturan konfigurasi cache sesuai dengan kebutuhan Anda.

## Menyiapkan ElastiCache di aplikasi Anda

Aplikasi Anda harus diatur untuk memanfaatkan ElastiCache cache. Anda juga dapat mengoptimalkan dan meningkatkan kinerja cache dengan menyiapkan aplikasi Anda untuk menggunakan strategi caching tergantung pada kebutuhan Anda.

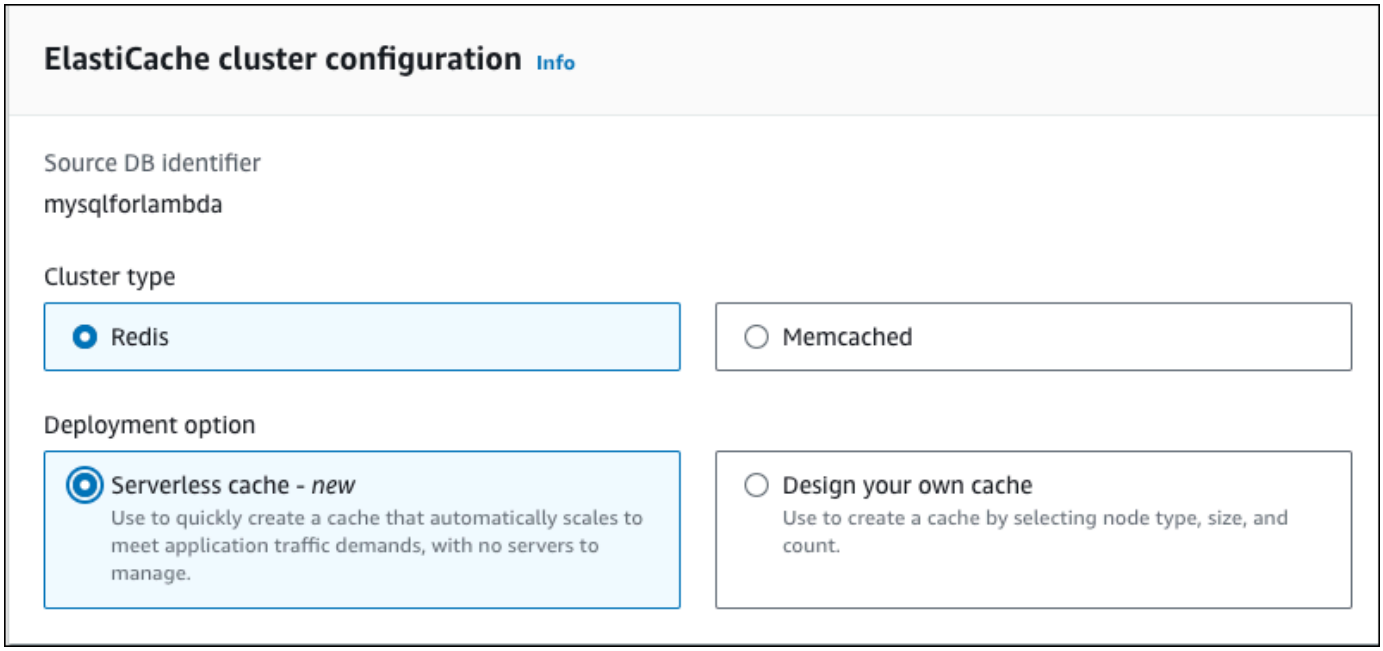
- Untuk mengakses ElastiCache cache Anda dan memulai, lihat Memulai dengan ElastiCache (RedisOSS) dan Memulai dengan ElastiCache (Memcached).
- Untuk informasi lebih lanjut tentang strategi caching, lihat Strategi cache dan praktik terbaik untuk strategi Memcached dan Caching dan praktik terbaik untuk Redis. OSS
- Untuk informasi selengkapnya tentang ketersediaan tinggi di klaster ElastiCache (RedisOSS), lihat Ketersediaan tinggi menggunakan grup replikasi.
- Anda mungkin dikenakan biaya yang terkait dengan penyimpanan cadangan, transfer data di dalam atau di seluruh wilayah, atau penggunaan. AWS Outposts Untuk detail harga, lihat ElastiCache harga Amazon.

Buat ElastiCache cache dengan pengaturan dari cluster DB

1. Untuk membuat klaster DB, ikuti petunjuk di Membuat klaster DB Amazon Aurora.
2. Setelah membuat , konsol menampilkan jendela add-on yang Disarankan. Pilih Buat ElastiCache cluster dari RDS menggunakan pengaturan DB Anda.

   Untuk database yang ada, di halaman Databases, pilih cluster DB yang diperlukan.

   Di bagian ElastiCache konfigurasi, pengidentifikasi Source DB menampilkan cluster DB mana yang mewarisi pengaturan ElastiCache cache.
3. Pilih apakah Anda ingin membuat cluster Redis OSS atau Memcached. Untuk informasi selengkapnya, lihat Membandingkan Memcached dan RedisOSS.




4. Setelah ini, pilih apakah Anda ingin membuat cache Tanpa Server atau Desain cache Anda sendiri. Untuk informasi selengkapnya, lihat Memilih antara opsi penerapan.

   Jika Anda memilih Cache tanpa server:

   a. Dalam pengaturan Cache, masukkan nilai untuk Nama dan Deskripsi.

   b. Di bawah Lihat pengaturan default, biarkan pengaturan default untuk membuat koneksi antara cache Anda dan cluster DB.

   c. Anda juga dapat mengedit pengaturan default dengan memilih Sesuaikan pengaturan default. Pilih pengaturan ElastiCache konektivitas, pengaturan ElastiCache keamanan, dan Batas penggunaan maksimum.

5. Jika Anda memilih Desain cache Anda sendiri:

   a. Jika Anda memilih OSSklaster Redis, pilih apakah Anda ingin mempertahankan mode cluster Diaktifkan atau Dinonaktifkan. Untuk informasi selengkapnya, lihat Replikasi: Redis OSS (Mode Cluster Dinonaktifkan) vs Redis OSS (Mode Cluster Diaktifkan).

   b. Masukkan nilai untuk Nama, Deskripsi, dan Versi mesin.

      Untuk Versi mesin, nilai default yang disarankan adalah versi mesin terbaru. Anda juga dapat memilih versi Engine untuk ElastiCache cache yang paling sesuai dengan kebutuhan Anda.

   c. Pilih jenis simpul dalam opsi Jenis simpul. Untuk informasi selengkapnya, lihat Mengelola simpul.

Jika Anda memilih untuk membuat OSS cluster Redis dengan mode Cluster disetel ke Enabled, maka masukkan jumlah pecahan (partisi/grup node) dalam opsi Jumlah pecahan.

Masukkan jumlah replika setiap serpihan di Jumlah replika.

> **Note**
>
> Jenis node yang dipilih, jumlah pecahan, dan jumlah replika semuanya memengaruhi kinerja cache dan biaya sumber daya Anda. Pastikan pengaturan ini sesuai dengan kebutuhan basis data Anda. Untuk informasi harga, lihat ElastiCache harga Amazon.

d. Pilih pengaturan ElastiCache konektivitas dan pengaturan ElastiCache keamanan. Anda dapat menyimpan pengaturan default atau menyesuaikan pengaturan ini sesuai kebutuhan Anda.

6. Verifikasi pengaturan default dan warisan ElastiCache cache Anda. Beberapa pengaturan tidak dapat diubah setelah pembuatan.

> **Note**
>
> RDSmungkin menyesuaikan jendela cadangan ElastiCache cache Anda untuk memenuhi persyaratan jendela minimum 60 menit. Periode pencadangan basis data sumber Anda tetap sama.

7. Saat Anda siap, pilih Buat ElastiCache cache.

Konsol menampilkan spanduk konfirmasi untuk pembuatan ElastiCache cache. Ikuti tautan di spanduk ke ElastiCache konsol untuk melihat detail cache. ElastiCache Konsol menampilkan ElastiCache cache yang baru dibuat.

# EC2AWS Database Migration Service

Anda dapat menggunakan konsol Aurora untuk memigrasikan database ke EC2 Aurora.  Aurora menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan basis data sumber Anda. EC2 AWS DMS memungkinkan Anda untuk memigrasikan database relasional ke Cloud Anda. AWS Untuk informasi lebih lanjut tentang AWS Database Migration Service, lihat Apa itu AWS Database Migration Service? dalam AWS Database Migration Service User Guide.

Untuk memulai migrasi, Anda harus membuat DB cluster Aurora DB yang setara untuk memigrasikan data ke dalamnya. Setelah Anda membuat database target Anda, Anda dapat mengimpor EC2 database Anda ke dalamnya. Untuk database sumber yang lebih kecil dari 1TiB, tindakan migrasi ini mengurangi waktu dan sumber daya yang diperlukan untuk memigrasikan data Anda ke Aurora.

## Gambaran Umum

Konsol Aurora memungkinkan Anda untuk memigrasikan EC2 database ke database Aurora yang setara.  Anda harus membuat database Aurora untuk mengaktifkan migrasi dari konsol.

Anda dapat memigrasikan EC2 database untuk mesin database berikut:

- Saya SQL
- Postgre SQL

Proses migrasi melibatkan langkah-langkah berikut:

- Buat database yang setara di Aurora . Agar database setara, mereka harus memiliki mesin database yang sama dan versi mesin yang kompatibel. Mereka juga harus samaVPC. Untuk petunjuk tentang membuat database Anda, lihat Membuat klaster DB Amazon Aurora.
- Pilih jenis replikasi untuk database Anda:
  - Migrasi beban penuh — Aurora menyalin database sumber lengkap ke database target, membuat tabel baru di target bila diperlukan.

    > **Note**
    >
    > Opsi ini menyebabkan pemadaman di database Aurora Anda.

  - Beban penuh dan ubah migrasi pengambilan data (CDC) — Mirip dengan migrasi muatan penuh, dengan opsi ini, Aurora menyalin database sumber lengkap ke basis data target. Namun,

setelah migrasi beban penuh, Aurora menerapkan setiap perubahan yang diambil dalam sumber ke database target. Ubah pengambilan data mengumpulkan perubahan pada log database dengan menggunakan asli API mesin database.

> **Note**
>
> Opsi ini menyebabkan pemadaman di database Aurora Anda.

- Ubah pengambilan data (CDC) — Gunakan opsi ini untuk menjaga basis data target Anda tetap tersedia melalui migrasi. Aurora memigrasikan perubahan yang sedang berlangsung di database sumber Anda ke database target.

- Aurora menciptakan sumber daya jaringan yang diperlukan untuk memfasilitasi migrasi. Setelah Aurora membuat sumber daya yang diperlukan, Aurora memberi tahu Anda tentang sumber daya yang dibuat dan memungkinkan Anda untuk memulai transfer data.

  Waktu yang dibutuhkan untuk menyelesaikan migrasi tergantung pada jenis replikasi dan ukuran database sumber.

## Prasyarat

### Saya SQL

Sebelum Anda mulai bekerja dengan database Saya SQL sebagai basis data sumber, pastikan Anda memiliki prasyarat berikut. Prasyarat ini berlaku untuk sumber yang dikelola. AWS

Anda harus memiliki akun AWS DMS yang memiliki peran Admin Replikasi. Peran itu memerlukan keistimewaan berikut:

- REPLICATIONCLIENT— Hak istimewa ini diperlukan untuk CDC tugas saja. Dengan kata lain, full-load-only tugas tidak memerlukan hak istimewa ini.
- REPLICATIONSLAVE— Hak istimewa ini diperlukan untuk CDC tugas saja. Dengan kata lain, full-load-only tugas tidak memerlukan hak istimewa ini.

AWS DMS Pengguna juga harus memiliki SELECT hak istimewa untuk tabel sumber yang ditunjuk untuk replikasi.

Berikan hak istimewa berikut jika Anda menggunakan penilaian premi SQL khusus Saya.

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

### Postgre SQL

Sebelum memigrasi data dari database SQL sumber Postgre yang AWS dikelola, lakukan hal berikut:

- Kami menyarankan Anda menggunakan akun AWS pengguna dengan izin minimum yang diperlukan untuk instans Postgre SQL DB sebagai akun pengguna untuk titik akhir sumber SQL Postgre. AWS DMS Menggunakan akun master tidak disarankan. Akun harus memiliki `rds_superuser` peran dan `rds_replication` peran. Peran `rds_replication` memberikan izin untuk mengelola slot logis dan mengalirkan data menggunakan slot logis.

> **Note**
>
> Beberapa AWS DMS transaksi menganggur selama beberapa waktu sebelum DMS mesin menggunakannya lagi. Dengan menggunakan parameter `idle_in_transaction_session_timeout` di Postgre SQL versi 9.6 dan yang lebih tinggi, Anda dapat menyebabkan transaksi idle menjadi time out dan gagal.

## Batasan

Batasan berikut berlaku untuk proses migrasi otomatis:

- Status basis data target Anda harus Tersedia untuk memulai migrasi basis data sumber.
- Saat bermigrasi dari database SQL sumber Saya, akun Aurora Anda harus memiliki peran Admin Replikasi. Anda juga harus memiliki hak istimewa yang tepat diterapkan untuk peran itu.
- Database EC2 instans dan target Anda harus samaVPC.
- Anda tidak dapat memigrasikan EC2 database ke database target berikut saat menggunakan tindakan Migrasi data dari EC2 database:
  - Aurora global database
  - Aurora Limitless database

- Aurora Serverless v1
- Database dengan SQL versi Saya lebih rendah dari 5.7
- Database dengan SQL versi Postgre lebih rendah dari 10.4

## Membuat sumber daya IAM untuk migrasi homogen

Aurora menggunakan AWS DMS untuk memigrasikan data Anda. Untuk mengakses database dan memigrasikan data, AWS DMS buat lingkungan tanpa server untuk migrasi data yang homogen. Dalam lingkungan ini, AWS DMS memerlukan akses ke peering VPC, tabel rute, grup keamanan, dan sumber daya lainnya. AWS Selain itu, AWS DMS menyimpan log, metrik, dan kemajuan untuk setiap migrasi data di Amazon CloudWatch. Untuk membuat proyek migrasi data, AWS DMS perlu akses ke layanan ini.

Juga, AWS DMS memerlukan akses ke rahasia yang respresent satu set kredensi pengguna untuk mengotentikasi koneksi database untuk sumber dan koneksi target.

> **Note**
>
> Dengan menggunakan tindakan Migrasi data dari EC2 instance, Anda dapat menggunakan konsol  RDS untuk menghasilkan sumber daya IAM ini. Lewati langkah ini jika Anda menggunakan sumber daya IAM yang dihasilkan konsol.

Anda memerlukan sumber daya IAM berikut untuk proses ini:

Topik

- Membuat kebijakan IAM untuk migrasi data homogen
- Membuat peran IAM untuk migrasi data homogen
- Membuat kebijakan dan peran akses rahasia
- Membuat IAM peran AWS DMS untuk mengelola Amazon VPC

### Membuat kebijakan IAM untuk migrasi data homogen

Pada langkah ini, Anda membuat kebijakan IAM yang menyediakan AWS DMS akses ke Amazon EC2 dan CloudWatch sumber daya. Selanjutnya, buat peran IAM dan lampirkan kebijakan ini.

Untuk membuat kebijakan IAM untuk migrasi data

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di halaman Buat kebijakan, pilih tab JSON.
5. Tempelkan JSON berikut ke editor.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribePrefixLists",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "arn:aws:servicequotas:*:*:vpc/L-0EA8095F"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute"
            ],
            "Resource": "arn:aws:ec2:*:*:route-table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:security-group-rule/*",
                "arn:aws:ec2:*:*:route-table/*",
                "arn:aws:ec2:*:*:vpc-peering-connection/*",
                "arn:aws:ec2:*:*:vpc/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group-rule/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AcceptVpcPeeringConnection",
                "ec2:ModifyVpcPeeringConnectionOptions"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-peering-connection/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:AcceptVpcPeeringConnection",
            "Resource": "arn:aws:ec2:*:*:vpc/*"
        }
    ]
}
```

6. Pilih Berikutnya: Tag dan Berikutnya: Tinjau.
7. Masukkan **`HomogeneousDataMigrationsPolicy`** Nama*, dan pilih Buat kebijakan.

## Membuat peran IAM untuk migrasi data homogen

Pada langkah ini, Anda membuat peran IAM yang menyediakan akses ke AWS Secrets Manager, Amazon EC2, dan CloudWatch.

Untuk membuat peran IAM untuk migrasi data

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Peran.
3. Pilih Buat peran.
4. Pada halaman Pilih entitas tepercaya, untuk jenis entitas tepercaya, pilih AWS Layanan. Untuk kasus penggunaan untuk AWS layanan lain, pilih DMS.

5. Pilih kotak centang DMS dan pilih Berikutnya.
6. Pada halaman Tambahkan izin, pilih HomogeneousDataMigrationsPolicyyang Anda buat sebelumnya. Pilih Berikutnya.
7. Pada halaman Nama, tinjau, dan buat, masukkan **`HomogeneousDataMigrationsRole`** nama Peran, dan pilih Buat peran.
8. Pada halaman Peran, masukkan nama **`HomogeneousDataMigrationsRole`** Peran. Pilih HomogeneousDataMigrationsRole.
9. Pada HomogeneousDataMigrationsRolehalaman, pilih tab Trust relationship. Pilih Edit kebijakan kepercayaan.
10. Pada halaman Edit kebijakan kepercayaan, tempelkan JSON berikut ke editor, ganti teks yang ada.

    JSON

    ```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "",
                "Effect": "Allow",
                "Principal": {
                    "Service": [
                        "dms-data-migrations.amazonaws.com",
                        "dms.your_region.amazonaws.com"
                    ]
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }
    ```

    Pada contoh sebelumnya, ganti *`your_region`* dengan nama Anda. Wilayah AWS

    Kebijakan berbasis sumber daya sebelumnya memberikan izin kepada prinsipal AWS DMS layanan untuk melakukan tugas sesuai dengan kebijakan yang dikelola pelanggan. HomogeneousDataMigrationsPolicy
11. Pilih Perbarui kebijakan.

## Membuat kebijakan dan peran akses rahasia

Ikuti prosedur di bawah ini untuk membuat kebijakan dan peran akses rahasia Anda yang memungkinkan DMS mengakses kredensi pengguna untuk basis data sumber dan target Anda.

Untuk membuat kebijakan dan peran akses rahasia, yang memungkinkan Amazon RDS mengakses AWS Secrets Manager untuk mengakses rahasia Anda yang sesuai

1. Masuk ke AWS Management Console dan buka konsol AWS Identity and Access Management (IAM) di https://console.aws.amazon.com/iam/.
2. Pilih Kebijakan, lalu pilih Buat kebijakan.
3. Pilih JSON dan masukkan kebijakan berikut untuk mengaktifkan akses ke dan dekripsi rahasia Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": secret_arn,
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
             "Resource": kms_key_arn,
        }
     ]
}
```

Di sini, *secret_arn* adalah ARN rahasia Anda, yang bisa Anda dapatkan dari yang `SecretsManagerSecretId` sesuai, dan *kms_key_arn* merupakan ARN dari AWS KMS kunci yang Anda gunakan untuk mengenkripsi rahasia Anda, seperti pada contoh berikut.

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
             "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
     ]
}
```

> **Note**
>
> Jika Anda menggunakan kunci enkripsi default yang dibuat oleh AWS Secrets Manager, Anda tidak perlu menentukan AWS KMS izin untuk*kms_key_arn*.
> Jika Anda ingin kebijakan Anda menyediakan akses ke kedua rahasia, cukup tentukan objek sumber daya JSON tambahan untuk yang lain*secret_arn*.

4. Tinjau dan buat kebijakan dengan nama yang dikenal dan deskripsi opsional.
5. Pilih Peran, lalu pilih Buat peran.
6. Pilih layanan AWS  sebagai jenis entitas terpercaya.
7. Pilih DMS dari daftar layanan sebagai layanan terpercaya, lalu pilih Berikutnya: Izin.
8. Cari dan lampirkan kebijakan yang Anda buat di langkah 4, lalu lanjutkan dengan menambahkan tag dan tinjau peran Anda. Pada titik ini, edit hubungan kepercayaan untuk peran tersebut untuk menggunakan prinsipal layanan regional Amazon RDS Anda sebagai entitas tepercaya. Prinsipal ini memiliki format berikut.

```
dms.region-name.amazonaws.com
```

Di sini, *region-name* adalah nama wilayah Anda, seperti `us-east-1`. Dengan demikian, kepala layanan regional Amazon RDS untuk wilayah ini mengikuti.

```
dms.us-east-1.amazonaws.com
dms-data-migrations.amazonaws.com
```

### Membuat IAM peran AWS DMS untuk mengelola Amazon VPC

Anda harus membuat IAM peran AWS DMS untuk mengelola VPC pengaturan untuk sumber daya Anda. Peran ini harus tersedia untuk migrasi yang berhasil.

Membuat **`dms-vpc-role`** untuk migrasi database

<result>

Ini menciptakan peran DMS untuk mengelola VPC pengaturan untuk migrasi.
</result>

1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Di panel navigasi konsol, pilih Peran dan kemudian pilih Buat peran.
3. Pilih Layanan AWSopsi untuk opsi Pilih entitas tepercaya.

   Untuk kasus penggunaan, pilih DMS.
4. Untuk langkah Tambahkan izin, pilih `AmazonDMSVPCManagementRole` dan pilih Berikutnya.
5. Di halaman Nama, tinjau, dan buat, setel nama Peran `dms-vpc-role` dan pilih Buat peran.

## Menyiapkan migrasi data untuk EC2 database

Untuk mulai memigrasikan data dari database EC2 sumber Anda, Anda harus membuat database Aurora yang setara. Untuk petunjuk tentang membuat database Anda, lihat Membuat klaster DB Amazon Aurora.

Setelah membuat database target, gunakan langkah-langkah berikut untuk mengatur migrasi data:

Siapkan proyek migrasi data

1. Pilih database target pada halaman Database di RDS konsol.

2. Pilih dropdown Tindakan dan pilih opsi Migrasi data dari EC2 database. Untuk melihat database target yang didukung, lihatBatasan.
3. Di bawah bagian Pilih EC2 database sumber:

    1. Periksa jenis Engine dan pastikan itu sama dengan database sumber Anda.

        Juga, periksa apakah versi mesinnya kompatibel.
    2. EC2Misalnya, pilih EC2 contoh di mana basis data sumber Anda berada.
    3. Untuk Port, masukkan port tempat database sumber Anda memungkinkan lalu lintas.
    4. Untuk Rahasia, pilih Buat dan gunakan rahasia baru jika Anda tidak memiliki rahasia yang ada. Masukkan Nama Pengguna dan Kata Sandi untuk basis data sumber Anda. Juga pilih KMS kunci yang digunakan untuk mengenkripsi rahasia Anda.

        Jika Anda memiliki rahasia yang ada, pilih Gunakan rahasia yang ada dan kemudian pilih rahasia dari dropdown.
    5. Untuk IAM peran rahasia, jika Anda memiliki IAM peran yang ada, pilih Gunakan IAM peran yang ada dan pilih IAM peran dari menu tarik-turun yang dapat mengakses ID rahasia dari langkah sebelumnya.

        Jika Anda tidak memiliki IAM peran yang ada, pilih Buat dan gunakan IAM peran baru. Masukkan nama baru untuk peran Anda untuk nama IAM peran. Anda dapat melihat izin yang terkait dengan peran ini di tautan di bawah ini.
4. Di bawah bagian Lihat RDS basis data target:

    1. Konfirmasikan pengaturan database target Anda di bagian atas bagian.
    2. Untuk Rahasia, pilih Buat dan gunakan rahasia baru jika Anda tidak memiliki rahasia yang ada yang menyimpan kredensi basis data target Anda.

        Jika Anda memiliki rahasia yang ada, pilih rahasia dari dropdown.
    3. Untuk IAM peran rahasia, pilih IAM peran yang dapat mengakses rahasia dari langkah sebelumnya. Anda juga dapat membuat IAM peran baru jika Anda tidak memiliki IAM peran yang ada.

        Jika dropdown tidak mengisi IAM peran, tentukan IAMperan ARN dalam format. `arn:aws:iam:`*`account_id`*`:role/`*`roleName`*
5. Di bawah bagian Konfigurasi migrasi data:

1. Pilih jenis migrasi data dengan memilih antara Full load, Full load dan change data capture (CDC), atau Change data capture (CDC). Untuk informasi selengkapnya tentang opsi ini, lihat Gambaran Umum.

   Anda tidak dapat mengubah jenis migrasi setelah migrasi dimulai.

2. Untuk IAM peran migrasi data, jika Anda memiliki IAM peran yang ada, pilih Gunakan peran yang ada IAM dan pilih IAM peran dari menu tarik-turun yang memberikan DMS izin untuk membuat sumber daya yang diperlukan untuk migrasi. Jika Anda tidak memiliki IAM peran yang ada, pilih Buat dan gunakan IAM peran baru.

6. Konfirmasikan bahwa tab Lihat pengaturan migrasi menampilkan setelan yang diperlukan agar migrasi data berhasil disiapkan.
7. Pilih Migrasi untuk menyelesaikan pengaturan migrasi.

Setelah menyelesaikan langkah-langkah ini, Anda dapat melihat sumber daya yang disiapkan untuk migrasi data dengan memilih Lihat detail di spanduk kemajuan di konsol. Setelah sumber daya yang diperlukan disiapkan, migrasi akan dimulai secara otomatis. Jika Anda membuat

Untuk memigrasikan beberapa database ke database target, mulai proses ini lagi dengan rincian tentang database baruEC2.

## Mengelola migrasi data

Setelah menggunakan tindakan Migrasi data dari EC2 database dari RDS konsol, Aurora memulai migrasi secara otomatis.

Jika Anda menggunakan AWS DMS konsol untuk membuat sumber daya migrasi, Anda dapat memulai proses migrasi.

### Memulai migrasi data

Ikuti langkah-langkah berikut untuk memulai migrasi data:

Memulai migrasi data

1. Pilih database target pada halaman Database di RDS konsol.
2. Di halaman detail database, pilih tab Migrasi data.
3. Di bawah tab Migrasi data, Migrasi data terkait mencantumkan migrasi data yang tersedia.

Migrasi yang disiapkan menggunakan konsol Aurora dimulai secara otomatis setelah sumber daya yang diperlukan disiapkan.

Migrasi yang disiapkan menggunakan DMS konsol diatur ke Siap.

Untuk memulai migrasi ini, pilih drop-down Tindakan dan pilih Mulai.

4. Ini memulai migrasi data untuk EC2 database Anda.

## Menghentikan migrasi data

Untuk migrasi data yang tipe replikasinya memuat penuh, menghentikan migrasi menyebabkan proses berhenti dan tidak dapat dilanjutkan. Setelah berhenti, Anda harus memulai ulang migrasi.

Untuk migrasi dengan tipe replikasi diatur untuk mengubah data capture (CDC) atau beban penuh danCDC, Anda dapat menghentikan proses replikasi berkelanjutan, dan melanjutkan proses nanti.

Menghentikan migrasi data

1. Pilih database target pada halaman Database di RDS konsol.
2. Di halaman detail database, pilih tab Migrasi data.
3. Di bawah tab Migrasi data, Migrasi data terkait mencantumkan migrasi data yang sedang berlangsung.

   Untuk menghentikan migrasi, pilih migrasi data dan pilih Berhenti di menu tarik-turun Tindakan.
4. Ini menghentikan migrasi data untuk EC2 database Anda.

## Melanjutkan migrasi data

Untuk migrasi data yang tipe replikasi adalah full load dan change data capture (CDC) atau change data capture (CDC) migrasi, Anda dapat melanjutkan CDC proses dari titik berhenti terakhir.

Melanjutkan migrasi data

1. Pilih database target pada halaman Database di RDS konsol.
2. Di halaman detail database, pilih tab Migrasi data.
3. Di bawah tab Migrasi data, Migrasi data terkait mencantumkan migrasi data yang dihentikan.

Untuk melanjutkan migrasi, pilih migrasi data dan pilih Lanjutkan pemrosesan di menu tarik-turun Tindakan.

4. Ini melanjutkan migrasi data untuk EC2 database Anda.

## Menghapus migrasi data

Untuk menghapus migrasi data terkait, gunakan petunjuk berikut

Menghapus migrasi data

1. Pilih database target pada halaman Database di RDS konsol.
2. Di halaman detail database, pilih tab Migrasi data.
3. Untuk menghapus migrasi, pilih migrasi data dan pilih Hapus di menu tarik-turun Tindakan.
4. Ini menghapus migrasi data.

Menghapus migrasi data yang sedang berlangsung tidak memengaruhi data apa pun yang telah dimuat ke database target.

## Memulai ulang migrasi data

Untuk memulai ulang migrasi data terkait dari titik CDC awal, gunakan petunjuk berikut

Memulai ulang migrasi data

1. Pilih database target pada halaman Database di RDS konsol.
2. Di halaman detail database, pilih tab Migrasi data.
3. Untuk memulai ulang migrasi, pilih migrasi data dan pilih Mulai ulang di menu tarik-turun Tindakan.
4. Ini memulai ulang migrasi data dari titik CDC awal.

Memulai ulang migrasi data yang sedang berlangsung tidak memengaruhi data apa pun yang telah dimuat ke database target.

# Memantau migrasi data Anda

Setelah migrasi data dimulai, Anda dapat memantau status dan kemajuannya. Migrasi data dari kumpulan data besar membutuhkan waktu berjam-jam untuk diselesaikan. Untuk menjaga keandalan, ketersediaan, dan kinerja tinggi migrasi data Anda, pantau perkembangannya secara teratur.

Untuk memeriksa status dan kemajuan migrasi data

1. Pilih database target pada halaman Database di RDS konsol.
2. Di halaman detail database, pilih tab Migrasi data.
3. Bagian Migrasi data terkait mencantumkan migrasi data Anda. Periksa kolom Status.
4. Untuk menjalankan migrasi data, kolom proses Migrasi menampilkan persentase data yang dimigrasi.
5. Untuk memantau proses di CloudWatch, gunakan tautan di CloudWatchkolom in.

## Status migrasi

Untuk setiap migrasi data yang Anda jalankan, konsol Aurora menampilkan Status. Daftar berikut mencakup status:

- `Ready`: Migrasi data siap untuk dimulai.
- `Starting`: Aurora menciptakan lingkungan tanpa server untuk migrasi data Anda.
- `Load running`: Aurora melakukan migrasi beban penuh.
- `Load complete, replication ongoing`: Aurora menyelesaikan beban penuh dan sekarang mereplikasi perubahan yang sedang berlangsung. Status ini hanya berlaku untuk migrasi muatan penuh dan CDC tipe.
- `Replication ongoing`: Aurora mereplikasi perubahan yang sedang berlangsung. Status ini hanya berlaku untuk migrasi CDC tipe.
- `Stopping`: Aurora menghentikan migrasi data. Status ini berlaku ketika Anda memilih untuk menghentikan migrasi data dari menu Tindakan.
- `Stopped`: Aurora telah menghentikan migrasi data.
- `Failed`: Migrasi data gagal. Untuk informasi selengkapnya, lihat file log.
- `Restarting`: Migrasi data telah memulai ulang replikasi data yang sedang berlangsung dari titik awal. CDC

# Tutorial: Membuat cluster dengan kustom dan grup opsi baru

Dalam tutorial ini, Anda membuat cluster dengan grup opsi baru. Untuk informasi selengkapnya tentang parameter , lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

> **Important**
>
> Tidak ada biaya untuk membuat AWS akun. Namun, dengan menyelesaikan tutorial ini, Anda mungkin dikenakan biaya untuk AWS sumber daya yang Anda gunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

Untuk membuat cluster DB dengan konfigurasi dan pengaturan khusus, Anda dapat menggunakan parameter kustom . Parameter khusus sangat membantu jika Anda bekerja dengan beberapa database dan ingin mengonfigurasi pengaturan secara seragam untuk mereka.

Dengan menyelesaikan langkah-langkah ini, Anda belajar:

- 
- Cara menggunakan parameter tertentu untuk .

Untuk menyelesaikan tutorial ini, lakukan tugas-tugas berikut:

1. 
2. Buat cluster dengan yang Anda buat.

Topik

- Prasyarat
- Buat grup parameter
- Ubah parameter dalam grup parameter kustom Anda
- Buat cluster dengan parameter cluster DB grup opsi baru

## Prasyarat

Tutorial ini mengharuskan Anda untuk memiliki Akun AWS dan pengguna dengan akses administratif. Jika Anda belum mengaturnya, selesaikan langkah-langkah di bagian berikut:

- Mendaftar untuk Akun AWS
- Buat pengguna dengan akses administratif

## Buat grup parameter

`default_password_lifetime`Parameter mengontrol jumlah hari sebelum kata sandi klien kedaluwarsa secara otomatis. Untuk informasi selengkapnya tentang parameter lain yang tersedia untuk cluster , lihat.

Untuk membuat grup parameter

1. Buka konsol Amazon RDS dan pilih Grup parameter.
2. Untuk grup parameter Kustom, pilih Buat grup parameter.
3. Mengatur rincian kelompok parameter.
    1. Masukkan nama untuk grup parameter.
    2. Masukkan deskripsi grup parameter.
    3. Untuk jenis Engine, pilih . Aurora MySQL.
    4. Untuk keluarga grup Parameter, pilih . aurora-mysql8.0.
    5. Untuk Type, pilih grup parameter cluster DB.
4. Pilih Buat.

Grup parameter cluster DB baru muncul di halaman grup Parameter di konsol Amazon RDS. Langkah-langkah berikut menggambarkan cara memodifikasi nilai parameter untuk menyesuaikan grup parameter Anda.

## Ubah parameter dalam grup parameter kustom Anda

Gunakan langkah-langkah berikut untuk memodifikasi  parameter dalam grup parameter yang Anda buatBuat grup parameter .

Untuk memodifikasi nilai parameter dalam grup parameter Anda

1. Buka konsol Amazon RDS dan pilih Grup parameter.
2. Untuk grup parameter Kustom, pilih nama grup parameter cluster DB yang Anda buat.
3. Pilih Edit.

4. Di kotak pencarian parameter Filter, cari parameter kustom`default_password_lifetime`.
5. Pilih kotak centang di sebelah parameter dan masukkan nilai jumlah hari yang akan ditetapkan untuk parameter seumur hidup kata sandi ini.
6. Pilih Simpan Perubahan.

Kluster DB.

## Buat cluster dengan parameter cluster DB grup opsi baru

Terakhir, buat cluster dengan yang Anda buat di langkah sebelumnya. Langkah-langkah berikut menunjukkan cara membuat cluster dengan grup opsi baru.

Untuk membuat cluster DB dengan parameter kustom dan grup opsi baru

1. Buka konsol Amazon RDS dan pilih Database.
2. Pilih Buat basis data.
3. Untuk Pilih metode pembuatan database, pilih Standard Create.
4. Untuk opsi Engine, pilih Compatible).
5. Pilih Konfigurasi Tambahan.
   - Untuk nama database Initial, pilih nama untuk cluster DB Anda.
   - Di bawah dropdown grup parameter cluster DB, pilih nama grup parameter cluster DB yang Anda buat sebelumnya.
6. Untuk tutorial ini, Anda dapat meninggalkan pengaturan default untuk pengaturan DB lainnya atau memodifikasinya sesuai kebutuhan.
7. Pilih Buat basis data.

RDS membuat cluster baru dengan opsi baru. Untuk melihat informasi selengkapnya tentang database ini, lihat halaman Database konsol Amazon RDS.

Dalam tutorial ini, Anda mengonfigurasi cluster dengan pengaturan yang disesuaikan menggunakan grup opsi baru. Cluster yang baru dibuat ini mengelola masa pakai kata sandi pengguna dengan menggunakan parameter. `default_password_lifetime` Untuk mengoptimalkan database Anda, Anda dapat menerapkan pengaturan tambahan di grup parameter kustom Anda dan menambahkan opsi.

Setelah Anda selesai membuat cluster  yang disesuaikan, Anda harus menghapus sumber daya Anda untuk menghindari biaya yang tidak diinginkan. Untuk menghapus cluster DB, ikuti instruksi di Menghapus instans DB atau klaster DB Aurora.

# Mengelola klaster DB Amazon Aurora

Bagian ini menunjukkan cara mengatur klaster DB Aurora Anda. Aurora melibatkan klaster server basis data yang terhubung dalam topologi replikasi. Dengan demikian, mengelola Aurora sering melibatkan penerapan perubahan ke beberapa server dan memastikan bahwa semua Replika Aurora mengikuti server sumber. Karena Aurora secara transparan menskalakan penyimpanan yang mendasari saat data Anda tumbuh, pengelolaan Aurora memerlukan manajemen penyimpanan disk yang relatif kecil. Demikian pula, karena Aurora secara otomatis melakukan pencadangan berkelanjutan, klaster Aurora tidak memerlukan perencanaan atau waktu henti yang ekstensif untuk melakukan pencadangan.

Topik


- Menghentikan dan memulai klaster DB Amazon Aurora
- Secara otomatis menghubungkan EC2 instance dan instans Aurora DB
- Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora
- Memodifikasi klaster DB Amazon Aurora
- Menambahkan Replika Aurora ke klaster DB
- Mengelola performa dan penskalaan untuk klaster DB Aurora
- Mengkloning volume untuk klaster DB Amazon Aurora
- Mengintegrasikan Aurora dengan layanan lain AWS
- Memelihara klaster DB Amazon Aurora
- Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora
- Gagal atas kluster DB Amazon Aurora
- Menghapus instans DB atau klaster DB Aurora
- Menandai sumber daya Amazon Aurora dan Amazon RDS
- Nama Sumber Daya Amazon (ARNs) di Amazon RDS
- Pembaruan Amazon Aurora

# Menghentikan dan memulai klaster DB Amazon Aurora

Menghentikan dan memulai klaster Aurora DB membantu Anda mengelola biaya untuk pengembangan dan pengujian lingkungan. Anda dapat menghentikan sementara semua instans DB dalam klaster Anda, dan tidak perlu menyiapkan dan merombak semua instans DB setiap kali menggunakan klaster.

Topik


- Ikhtisar menghentikan dan memulai klaster DB Aurora
- Pembatasan untuk menghentikan dan memulai klaster DB Aurora
- Menghentikan klaster DB Aurora
- Operasi yang mungkin dilakukan saat klaster DB Aurora dihentikan
- Memulai klaster DB Aurora


## Ikhtisar menghentikan dan memulai klaster DB Aurora

Selama periode di mana Anda tidak memerlukan cluster Aurora DB, Anda dapat menghentikan semua instance di cluster itu sekaligus. Anda dapat memulai klaster lagi kapan pun diperlukan. Memulai dan menghentikan akan menyederhanakan proses penyiapan dan perombakan pada klaster yang digunakan untuk pengembangan, pengujian, atau aktivitas serupa yang tidak memerlukan ketersediaan yang berkelanjutan. Anda dapat melakukan semua AWS Management Console prosedur yang terlibat hanya dengan satu tindakan, terlepas dari berapa banyak contoh yang ada di cluster.

Saat cluster DB Anda dihentikan, Anda hanya dikenakan biaya untuk penyimpanan cluster, snapshot manual, dan penyimpanan cadangan otomatis dalam jendela retensi yang Anda tentukan. Anda tidak dikenakan biaya untuk jam instans DB.

> **Important**
>
> Anda dapat menghentikan klaster DB hingga tujuh hari. Jika Anda tidak memulai klaster DB secara manual setelah tujuh hari, klaster DB Anda secara otomatis dimulai sehingga tidak tertinggal pembaruan pemeliharaan yang diperlukan.

Untuk meminimalkan biaya klaster Aurora yang bermuatan ringan, Anda dapat menghentikan klaster tersebut alih-alih menghapus semua Aurora Replicas. Untuk cluster dengan lebih dari satu atau

dua instance, sering menghapus dan membuat ulang instans DB hanya praktis menggunakan atau AWS CLI Amazon RDS API. Rangkaian operasi tersebut juga sulit dilakukan dalam urutan yang benar, misalnya untuk menghapus semua Aurora Replicas sebelum menghapus instans utama untuk menghindari pengaktifan mekanisme failover.

Jangan gunakan fungsi memulai dan menghentikan jika Anda perlu terus menjalankan klaster DB, tetapi klaster memiliki lebih banyak kapasitas dari yang Anda butuhkan. Jika klaster Anda terlalu mahal atau tidak terlalu sibuk, hapus satu instans DB atau lebih atau ubah semua instans DB Anda menjadi kelas instans kecil. Anda tidak dapat menghentikan instans DB Aurora secara terpisah.

Waktu untuk menghentikan cluster DB Anda bervariasi tergantung pada faktor-faktor seperti kelas instans DB, status jaringan, jenis mesin DB, dan status database. Prosesnya bisa memakan waktu beberapa menit. Layanan Amazon RDS melakukan tindakan berikut:

- Mematikan proses mesin database.
- Mematikan proses platform RDS.
- Mengakhiri EC2 instans Amazon yang mendasarinya.

Waktu untuk me-restart cluster DB Anda bervariasi tergantung pada faktor-faktor seperti ukuran database, kelas instans DB, status jaringan, tipe mesin DB, dan status database ketika cluster dimatikan. Proses startup dapat memakan waktu beberapa menit hingga berjam-jam, tetapi biasanya memakan waktu beberapa menit. Kami menyarankan Anda mempertimbangkan variabilitas dalam waktu startup saat membuat paket ketersediaan Anda.

Untuk memulai cluster DB, layanan melakukan tindakan seperti berikut:

- Menyediakan EC2 instans Amazon yang mendasarinya.
- Memulai proses platform RDS.
- Memulai proses mesin database.
- Memulihkan instans DB (pemulihan terjadi bahkan setelah shutdown normal).

## Pembatasan untuk menghentikan dan memulai klaster DB Aurora

Beberapa klaster Aurora tidak dapat dihentikan dan dimulai:

- Anda hanya dapat menghentikan dan memulai cluster yang merupakan bagian dari database global Aurora jika itu satu-satunya cluster dalam database global.

- Anda tidak dapat menghentikan dan memulai klaster yang memiliki replika baca lintas-Wilayah.
- Anda tidak dapat menghentikan dan memulai klaster yang merupakan bagian dari deployment blue/green.
- Anda tidak bisa berhenti dan memulai Aurora Serverless v1 klaster. dengan Aurora Serverless v2, Anda dapat berhenti dan memulai cluster.

## Menghentikan klaster DB Aurora

Untuk menggunakan klaster DB Aurora atau melakukan administrasi, selalu mulai dengan klaster DB Aurora yang berjalan, kemudian hentikan, lalu mulai lagi klaster tersebut. Saat klaster dihentikan, Anda akan dikenakan biaya untuk penyimpanan klaster, snapshot manual, dan penyimpanan cadangan otomatis dalam periode penyimpanan yang ditentukan, tetapi tidak untuk jam instans DB.

Operasi penghentian akan menghentikan instans Aurora Replica terlebih dahulu, lalu instans utama, untuk menghindari pengaktifan mekanisme failover.

Konsol

Untuk menghentikan klaster Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih salah satu klaster. Anda dapat melakukan operasi penghentian dari halaman ini, atau membuka halaman detail untuk klaster DB yang ingin Anda hentikan.
3. Untuk Tindakan, pilih Hentikan sementara.
4. Di jendela Hentikan sementara klaster DB, pilih pengakuan bahwa klaster DB akan dimulai ulang secara otomatis setelah 7 hari.
5. Pilih Hentikan sementara untuk menghentikan klaster DB, atau pilih Batalkan untuk membatalkan operasi.

AWS CLI

Untuk menghentikan instance DB dengan menggunakan AWS CLI, panggil stop-db-clusterperintah dengan parameter berikut:

- `--db-cluster-identifier` – nama klaster Aurora.

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

API RDS

Untuk menghentikan instans DB dengan menggunakan Amazon RDS API, panggil DBCluster operasi Stop dengan parameter berikut:

- `DBClusterIdentifier` – nama klaster Aurora.

## Operasi yang mungkin dilakukan saat klaster DB Aurora dihentikan

Sementara klaster Aurora dihentikan, Anda dapat melakukan point-in-time pemulihan ke titik mana pun dalam jendela retensi cadangan otomatis yang Anda tentukan. Untuk detail tentang melakukan point-in-time pemulihan, lihatMemulihkan data.

Anda tidak dapat memodifikasi konfigurasi klaster DB Aurora, atau salah satu instans DB-nya saat klaster dihentikan. Anda juga tidak dapat menambahkan atau menghapus instans DB dari klaster, atau menghapus klaster tersebut jika masih terdapat instans DB terkait. Anda harus memulai klaster sebelum melakukan tindakan administratif tersebut.

Menghentikan klaster DB akan menghapus tindakan yang tertunda, kecuali untuk grup parameter klaster DB atau untuk grup parameter DB dari instans klaster DB.

Aurora menerapkan pemeliharaan terjadwal apa pun pada klaster yang dihentikan setelah dimulai lagi. Perlu dicatat bahwa setelah tujuh hari, Aurora secara otomatis memulai klaster yang dihentikan sehingga klaster tersebut tidak terlalu jauh tertinggal dalam status pemeliharaannya.

Aurora juga tidak melakukan cadangan otomatis apa pun, karena data yang mendasari tidak berubah saat klaster dihentikan. Aurora tidak memperpanjang periode penyimpanan cadangan saat klaster dihentikan.

## Memulai klaster DB Aurora

Anda selalu memulai klaster DB Aurora diawali dengan klaster Aurora yang sudah berada dalam status terhenti. Saat Anda memulai klaster tersebut, semua instans DB menjadi tersedia lagi. Klaster tersebut menjaga pengaturan konfigurasi seperti titik akhir, grup parameter dan grup keamanan VPC.

Memulai cluster DB Anda biasanya membutuhkan waktu beberapa menit.

Konsol

Untuk memulai klaster Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih salah satu klaster. Anda dapat melakukan operasi mulai dari halaman ini, atau membuka halaman detail untuk klaster DB yang ingin Anda mulai.
3. Untuk Tindakan, pilih Mulai.

AWS CLI

Untuk memulai cluster DB dengan menggunakan AWS CLI, panggil start-db-clusterperintah dengan parameter berikut:

- `--db-cluster-identifier` – nama klaster Aurora. Nama ini adalah pengidentifikasi klaster tertentu yang Anda pilih saat membuat klaster, atau pengidentifikasi instans DB yang Anda pilih dengan `-cluster` yang ditambahkan ke bagian akhir.

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

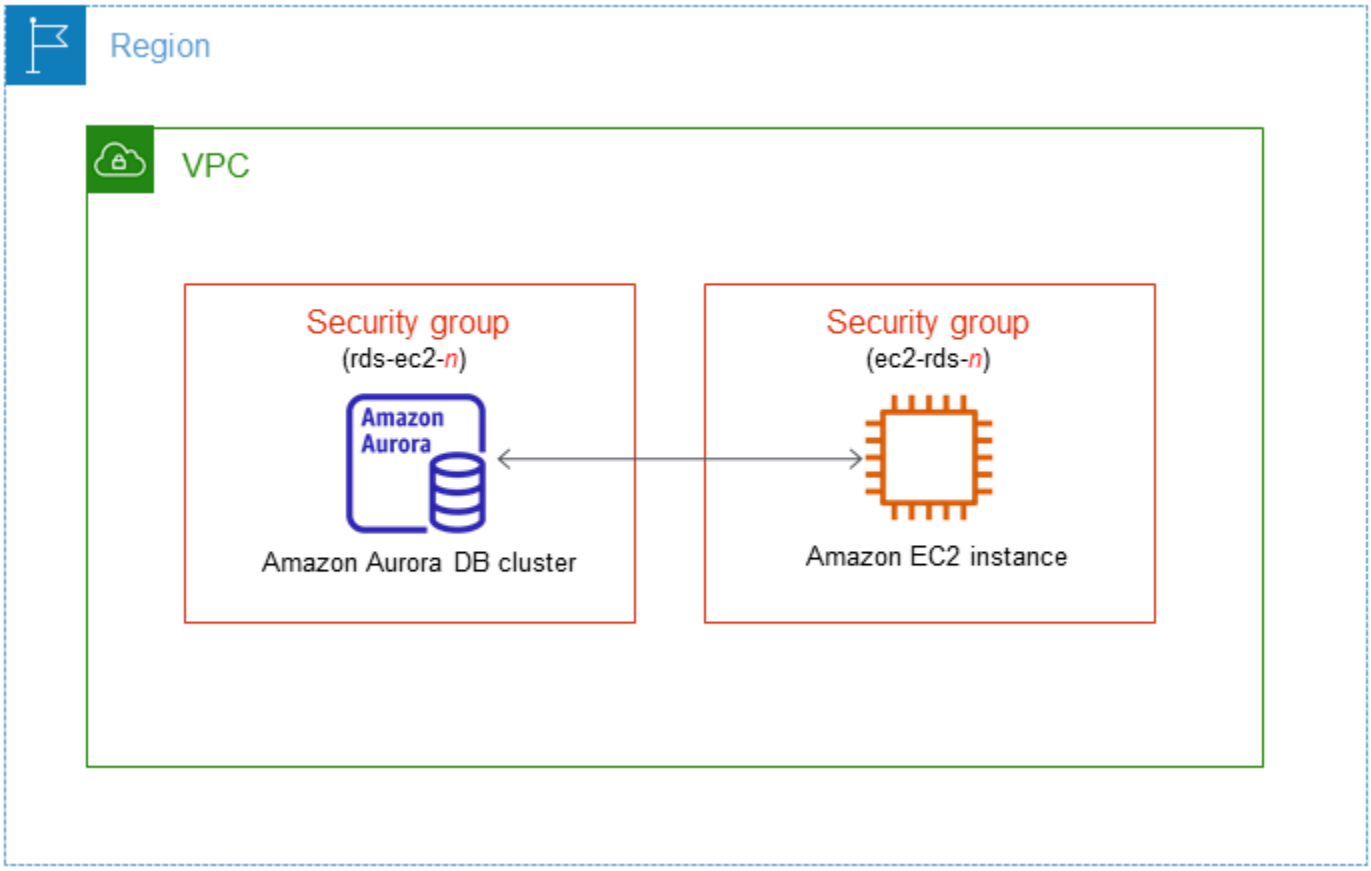
API RDS

Untuk memulai cluster Aurora DB dengan menggunakan Amazon RDS API, panggil DBCluster operasi Mulai dengan parameter berikut:

- `DBCluster` – nama klaster Aurora. Nama ini adalah pengidentifikasi klaster tertentu yang Anda pilih saat membuat klaster, atau pengidentifikasi instans DB yang Anda pilih dengan `-cluster` yang ditambahkan ke bagian akhir.

# Secara otomatis menghubungkan EC2 instance dan instans Aurora DB

Anda dapat menggunakan RDS konsol Amazon untuk menyederhanakan pengaturan koneksi antara instans Amazon Elastic Compute Cloud EC2 (Amazon) dan instans Aurora DB. Seringkali, cluster DB Anda berada di subnet pribadi dan EC2 instance Anda berada di subnet publik dalam file. VPC Anda dapat menggunakan SQL klien pada EC2 instans Anda untuk terhubung ke cluster DB Anda. EC2Instans ini juga dapat menjalankan server web atau aplikasi yang mengakses cluster pribadi Anda.




Jika Anda ingin terhubung ke EC2 instance yang tidak VPC sama dengan cluster Aurora DB, lihat skenario di. Skenario untuk mengakses cluster DB di VPC

Topik

- Ikhtisar konektivitas otomatis dengan sebuah EC2 instance
- Secara otomatis menghubungkan EC2 instance dan Aurora DB cluster
- Melihat sumber daya komputasi terhubung

- Menghubungi instans basis data yang menjalankan mesin basis data tertentu

## Ikhtisar konektivitas otomatis dengan sebuah EC2 instance

Saat Anda menyiapkan sambungan antara EC2 instans dan cluster DB Aurora RDS , Amazon RDSautomatically mengonfigurasi grup VPC keamanan untuk EC2 instans Anda dan untuk kluster DB Anda.

Berikut ini adalah persyaratan untuk menghubungkan EC2 instance dengan Aurora DB cluster:

- EC2Instance harus ada VPC sama dengan cluster DB RDS .

  Jika tidak ada EC2 instance yang samaVPC, maka konsol menyediakan tautan untuk membuatnya.
- Saat ini, klaster DB tidak dapat berupa klaster DB Aurora Serverless atau bagian dari basis data global Aurora.
- Pengguna yang mengatur konektivitas harus memiliki izin untuk melakukan EC2 operasi Amazon berikut:
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Jika instans dan EC2 instans DB berada di Availability Zone yang berbeda, akun Anda mungkin dikenakan biaya Cross-availability Zone.

Saat Anda menyiapkan sambungan ke EC2 instans, Amazon RDS bertindak sesuai dengan konfigurasi grup keamanan saat ini yang terkait dengan cluster dan EC2 instance DB RDS , seperti yang dijelaskan dalam tabel berikut.

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup EC2 keamanan saat ini | RDSaksi |
| --- | --- | --- |
| Ada satu atau lebih grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola `rds-ec2-`*n* (di *n* mana angka). Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan masuk dengan grup VPC keamanan EC2 instance sebagai sumbernya. | Ada satu atau lebih grup keamanan yang terkait dengan EC2 instance dengan nama yang cocok dengan pola `ec2-rds-`*n* (di *n* mana angka). Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan keluar dengan grup VPC keamanan cluster DB RDS sebagai sumbernya. | RDStidak mengambil tindakan.<br><br>Koneksi sudah dikonfigurasi secara otomatis antara EC2 instance dan DB cluster. Karena koneksi sudah ada antara EC2 instance dan RDS database, grup keamanan tidak dimodifikasi. |
| Salah satu syarat berikut berlaku:<br><br>• Tidak ada grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola`rds-ec2-`*n*.<br>• Ada satu atau lebih grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola`rds-ec2-`*n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan EC2 instans. Amazon tidak RDS dapat menggunak an grup keamanan yang tidak memiliki satu aturan | Salah satu syarat berikut dipenuhi:<br><br>• Tidak ada grup keamanan yang terkait dengan EC2 instance dengan nama yang cocok dengan pola`ec2-rds-`*n*.<br>• Ada satu atau lebih grup keamanan yang terkait dengan EC2 instance dengan nama yang cocok dengan pola`ec2-rds-`*n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster DB RDS . Amazon tidak RDS dapat menggunak an grup keamanan yang | RDS action: create new security groups |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup EC2 keamanan saat ini | RDSaksi |
| --- | --- | --- |
| masuk dengan grup VPC keamanan EC2 instance sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. Contoh-contoh perubahan meliputi penambahan aturan atau pengubahan port aturan yang ada. | tidak memiliki satu aturan keluar dengan grup VPC keamanan cluster DB RDS sebagai sumbernya . Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | |
| Ada satu atau lebih grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola `rds-ec2-`*n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan masuk dengan grup VPC keamanan EC2 instance sebagai sumbernya. | Ada satu atau lebih grup keamanan yang terkait dengan EC2 instance dengan nama yang cocok dengan pola `ec2-rds-`*n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster DB RDS . Amazon tidak RDS dapat menggunakan grup keamanan yang tidak memiliki satu aturan keluar dengan grup VPC keamanan cluster DB RDS sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | RDS action: create new security groups |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup EC2 keamanan saat ini | RDSaksi |
| --- | --- | --- |
| Ada satu atau lebih grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola`rds-ec2-`*n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan masuk dengan grup VPC keamanan EC2 instance sebagai sumbernya. | Grup EC2 keamanan yang valid untuk koneksi ada, tetapi tidak terkait dengan EC2 instance. Grup keamanan ini memiliki nama yang cocok dengan pola `ec2-rds-`*n*. Grup tersebut belum diubah. Ini hanya memiliki satu aturan keluar dengan grup VPC keamanan cluster DB RDS sebagai sumbernya. | RDS action: associate EC2 security group |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup EC2 keamanan saat ini | RDSaksi |
| --- | --- | --- |
| Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola`rds-ec2-`*n*.<br>• Ada satu atau lebih grup keamanan yang terkait dengan cluster DB RDS dengan nama yang cocok dengan pola`rds-ec2-`*n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan EC2 instans. Amazon tidak RDS dapat menggunak an grup keamanan yang tidak memiliki satu aturan masuk dengan grup VPC keamanan EC2 instance sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | Ada satu atau lebih grup keamanan yang terkait dengan EC2 instance dengan nama yang cocok dengan pola`ec2-rds-`*n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan keluar dengan grup VPC keamanan cluster DB RDS sebagai sumbernya. | RDS action: create new security groups |

RDStindakan: buat grup keamanan baru

Amazon RDS mengambil tindakan berikut:

- Membuat grup keamanan baru yang cocok dengan pola `rds-ec2-`*n*. Grup keamanan ini memiliki aturan masuk dengan grup VPC keamanan EC2 instance sebagai sumbernya. Grup keamanan ini dikaitkan dengan cluster DB RDS dan memungkinkan EC2 instance untuk mengakses cluster DB RDS .
- Membuat grup keamanan baru yang cocok dengan pola `ec2-rds-`*n*. Grup keamanan ini memiliki aturan keluar dengan grup VPC keamanan cluster DB RDS sebagai target. Grup keamanan ini dikaitkan dengan EC2 instance dan memungkinkan EC2 instance untuk mengirim lalu lintas ke cluster DB RDS .

RDStindakan: kelompok EC2 keamanan asosiasi

Amazon RDS mengaitkan grup EC2 keamanan yang valid dan sudah ada dengan EC2 instans. Grup keamanan ini memungkinkan EC2 instance untuk mengirim lalu lintas ke cluster DB RDS .

## Secara otomatis menghubungkan EC2 instance dan Aurora DB cluster

Sebelum menyiapkan koneksi antara EC2 instance dan Aurora DB cluster, pastikan Anda memenuhi persyaratan yang dijelaskan dalam. Ikhtisar konektivitas otomatis dengan sebuah EC2 instance

Jika Anda membuat perubahan pada grup keamanan setelah mengonfigurasi konektivitas, perubahan tersebut dapat memengaruhi koneksi antara EC2 instans dan cluster Aurora DB RDS .

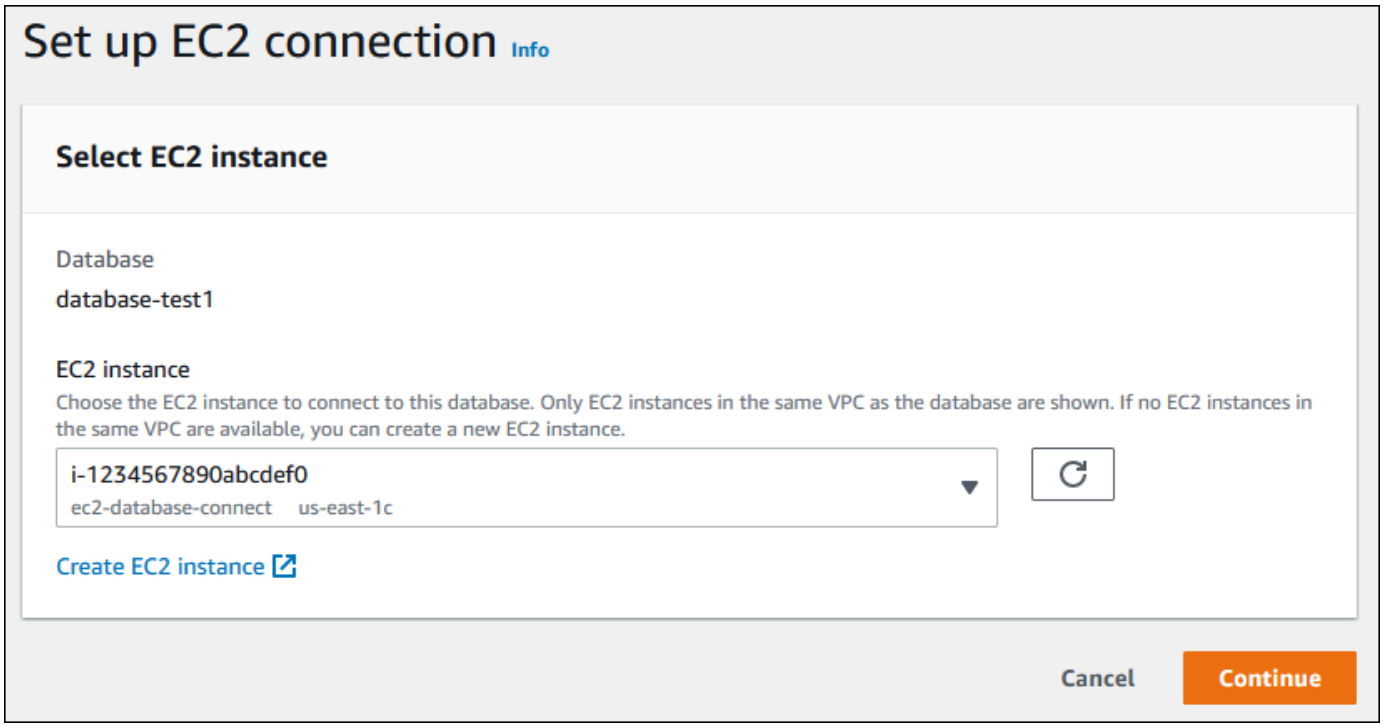
> **Note**
>
> Anda hanya dapat mengatur koneksi antara EC2 instance dan Aurora DB cluster secara otomatis dengan menggunakan. AWS Management Console Anda tidak dapat mengatur koneksi secara otomatis dengan AWS CLI atau RDSAPI.

Untuk menghubungkan EC2 instance dan Aurora DB cluster secara otomatis

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Databases, lalu pilih cluster DB RDS .
3. Dari Tindakan, pilih Siapkan EC2 koneksi.

   Halaman EC2Koneksi Siapkan muncul.

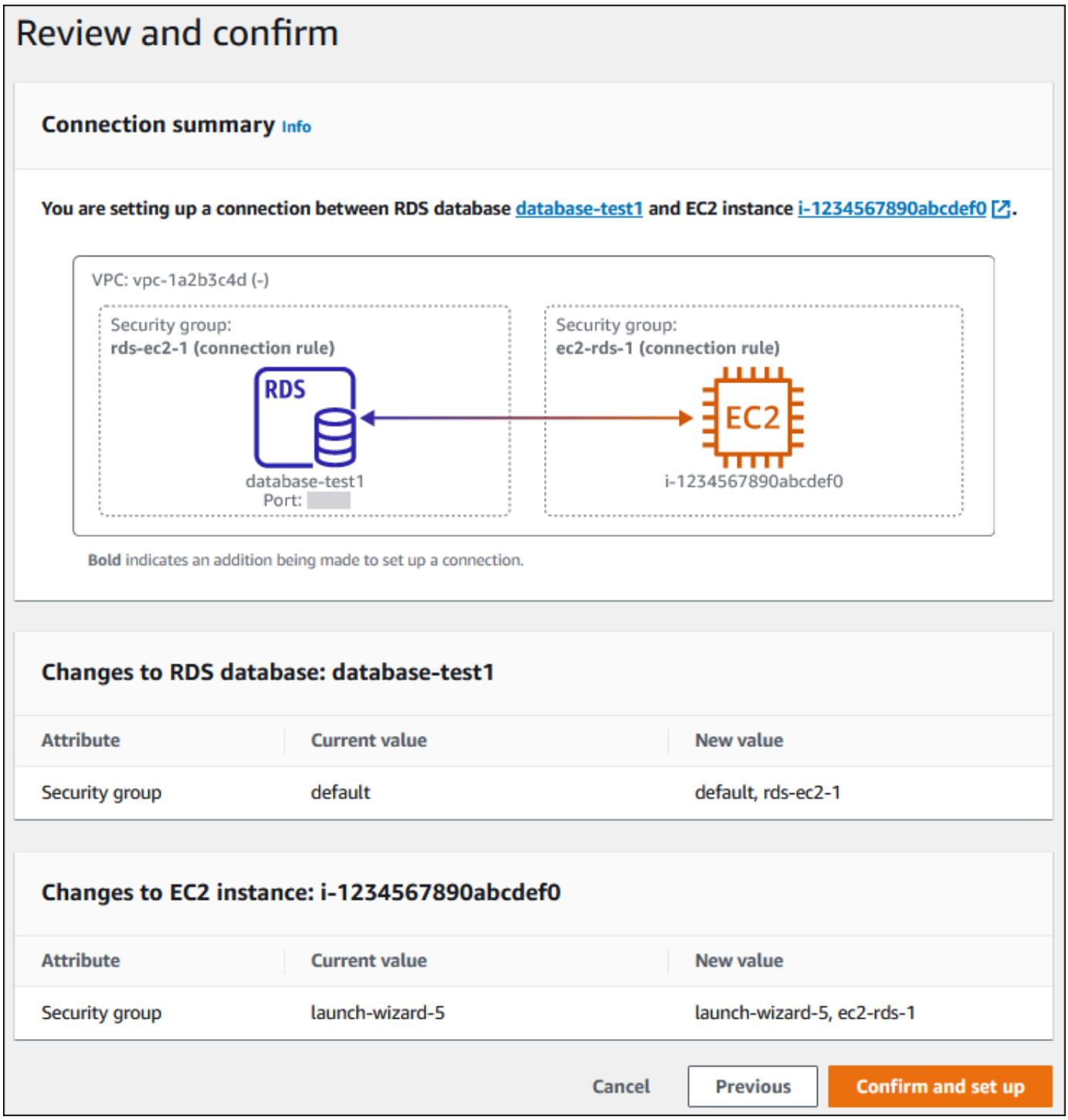
4. Pada halaman Siapkan EC2 koneksi, pilih EC2 instance.




Jika tidak ada EC2 instance yang samaVPC, pilih Buat EC2 instance untuk membuatnya. Dalam hal ini, pastikan EC2 instance baru VPC sama dengan cluster DB RDS .

5. Pilih Lanjutkan.

Halaman Tinjau dan tegaskan muncul.




6. Pada halaman Tinjau dan konfirmasi, tinjau perubahan yang RDS akan dilakukan untuk mengatur konektivitas dengan EC2 instans.

   Jika perubahan sudah benar, pilih Tegaskan dan siapkan.

   Jika perubahan masih salah, pilih Sebelumnya atau Batalkan.

# Melihat sumber daya komputasi terhubung

Anda dapat menggunakan AWS Management Console untuk melihat sumber daya komputasi yang terhubung ke Aurora DB cluster. Sumber daya yang ditampilkan meliputi koneksi sumber daya komputasi yang disiapkan secara otomatis. Anda dapat menyiapkan secara otomatis konektivitas dengan sumber daya komputasi dengan cara berikut:

- Anda dapat memilih sumber daya komputasi saat membuat basis data.

  Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.
- Anda dapat menyiapkan konektivitas antara basis data yang ada dan sumber daya komputasi.

  Untuk informasi selengkapnya, lihat Secara otomatis menghubungkan EC2 instance dan Aurora DB cluster.

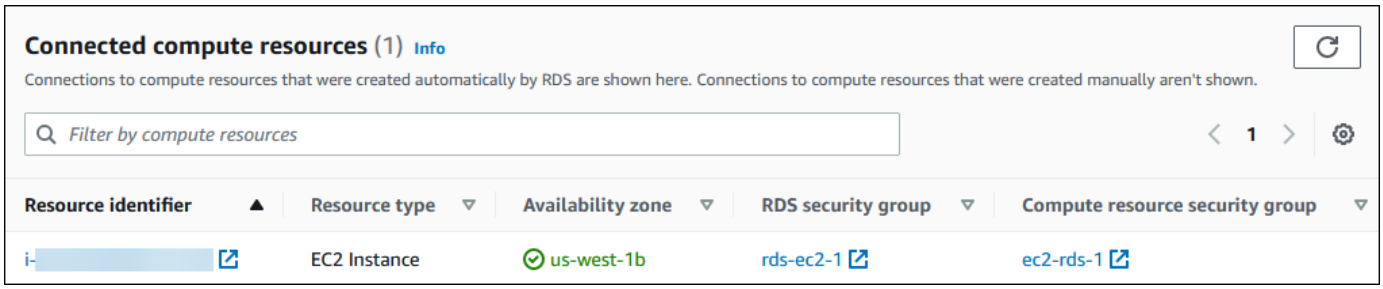
Sumber daya komputasi yang tercantum tidak menyertakan sumber daya yang dihubungkan secara manual dengan basis data. Misalnya, Anda dapat mengizinkan sumber daya komputasi untuk mengakses database secara manual dengan menambahkan aturan ke grup VPC keamanan yang terkait dengan database.

Agar sumber daya komputasi tercantum, syarat-syarat berikut harus dipenuhi:

- Nama grup keamanan yang terkait dengan sumber daya komputasi cocok dengan pola `ec2-rds-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan sumber daya komputasi memiliki aturan keluar dengan rentang port yang disetel ke port yang digunakan cluster DB RDS .
- Grup keamanan yang terkait dengan sumber daya komputasi memiliki aturan keluar dengan sumber yang disetel ke grup keamanan yang terkait dengan cluster DB RDS .
- Nama grup keamanan yang terkait dengan cluster DB RDS cocok dengan pola `rds-ec2-`*n* (di *n* mana angka).
- Grup keamanan yang terkait dengan cluster DB RDS memiliki aturan inbound dengan rentang port yang diatur ke port yang digunakan cluster DB RDS .
- Grup keamanan yang terkait dengan cluster DB RDS memiliki aturan masuk dengan sumber yang disetel ke grup keamanan yang terkait dengan sumber daya komputasi.

Untuk melihat sumber daya komputasi yang terhubung ke Aurora DB cluster

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Databases, lalu pilih nama cluster DB RDS .
3. Pada tab Konektivitas dan keamanan, lihat sumber daya komputasi di Sumber daya komputasi terhubung.




## Menghubungi instans basis data yang menjalankan mesin basis data tertentu

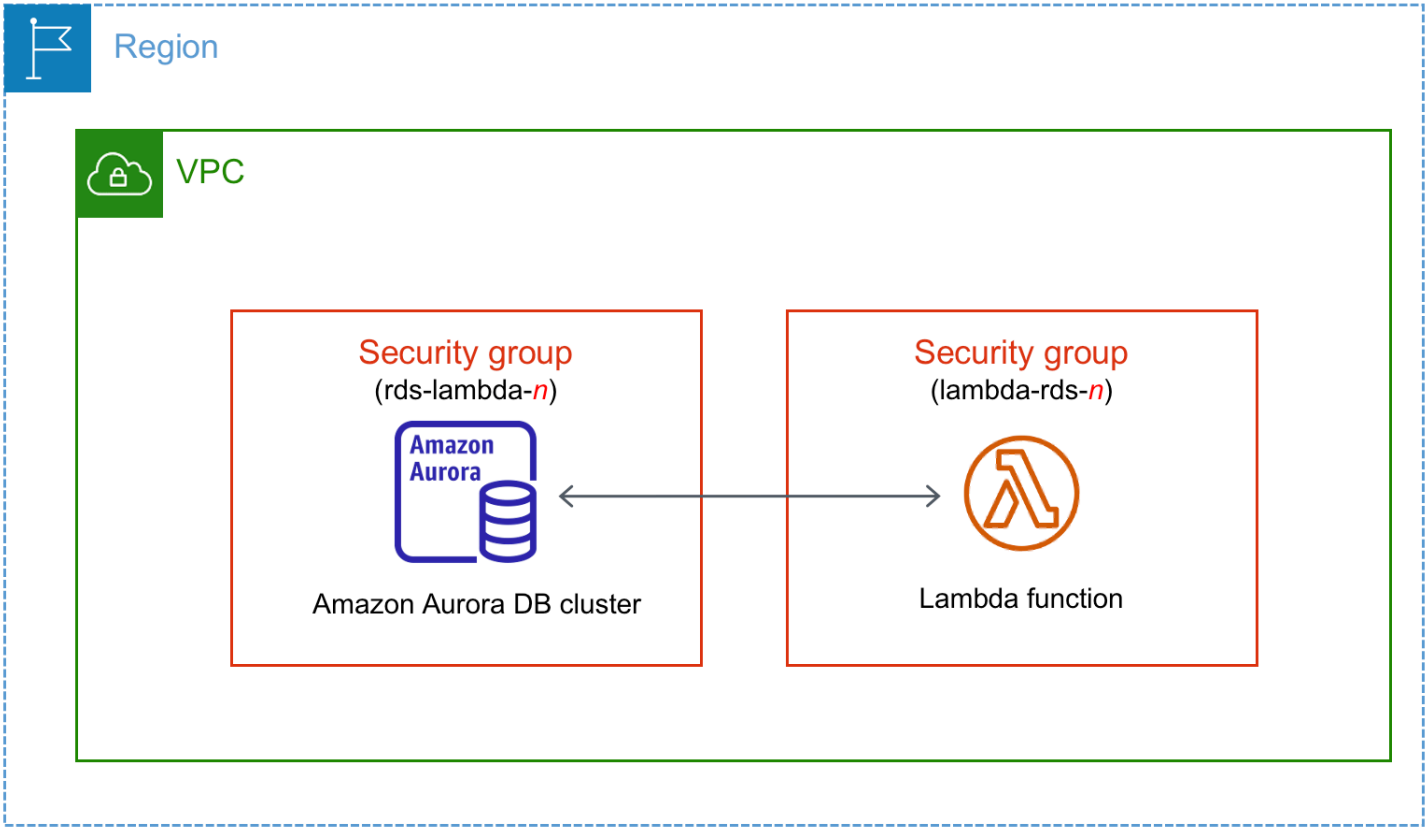
Untuk informasi tentang cara menghubungi instans basis data yang menjalankan mesin basis data tertentu, ikuti petunjuk untuk mesin basis data Anda:

- Menghubungkan ke klaster DB Amazon Aurora MySQL
- Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.

# Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora

Anda dapat menggunakan RDS konsol Amazon untuk menyederhanakan pengaturan koneksi antara fungsi Lambda . Seringkali, cluster DB Anda berada di subnet pribadi dalam file. VPC Fungsi Lambda dapat digunakan oleh aplikasi untuk mengakses klaster DB privat Anda.

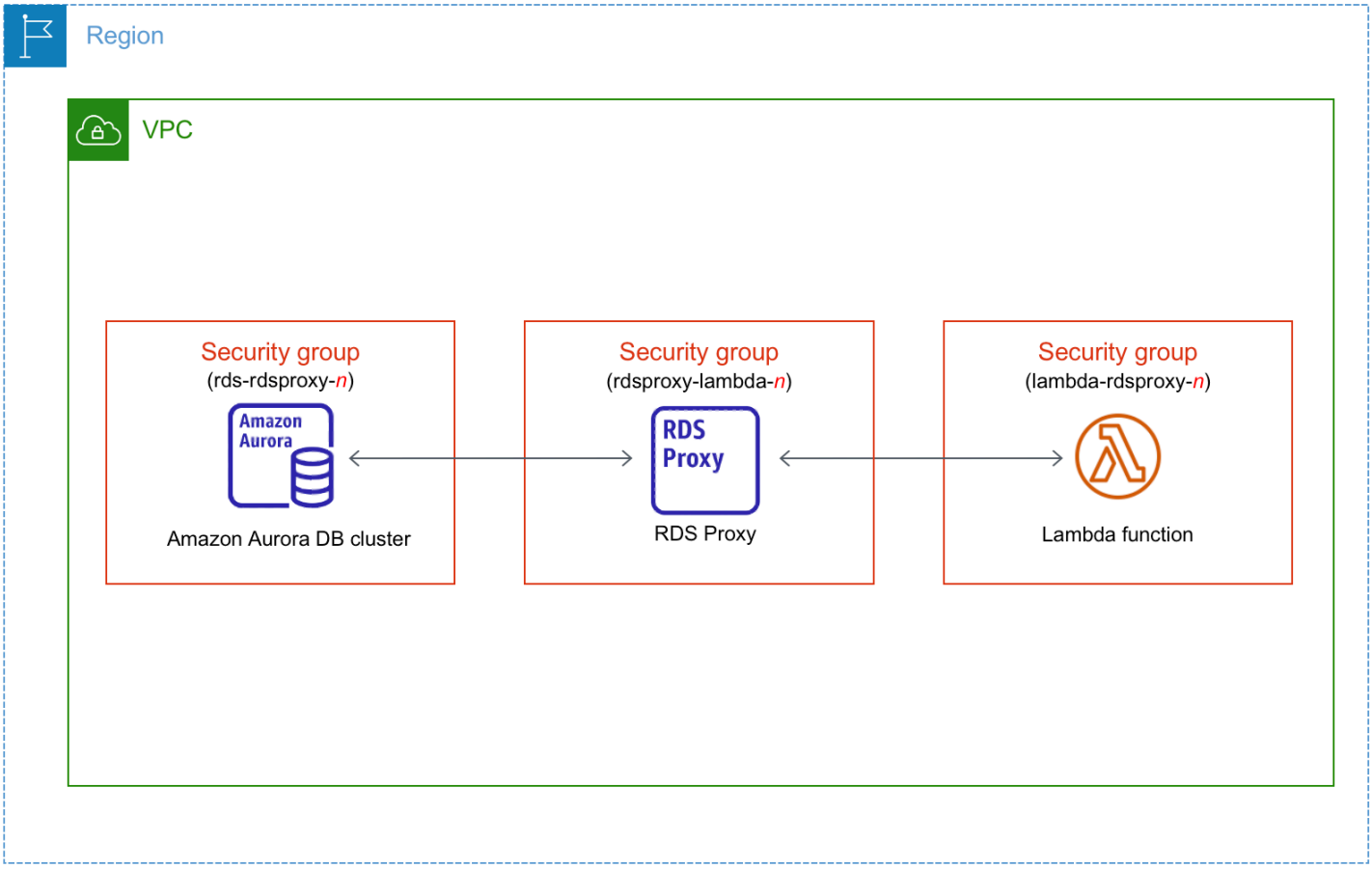
Gambar berikut menunjukkan koneksi langsung antara klaster DB dan fungsi Lambda.




Anda dapat mengatur koneksi antara fungsi Lambda dan cluster Anda melalui RDS Proxy untuk meningkatkan kinerja dan ketahanan database Anda. Seringkali, fungsi Lambda membuat koneksi database yang sering dan pendek yang mendapat manfaat dari penyatuan koneksi yang ditawarkan Proxy. RDS Anda dapat memanfaatkan otentikasi AWS Identity and Access Management (IAM) apa pun yang sudah Anda miliki untuk fungsi Lambda, alih-alih mengelola kredensi basis data dalam kode aplikasi Lambda Anda. Untuk informasi selengkapnya, lihat Proksi Amazon RDS untuk Aurora.

Saat Anda menggunakan konsol untuk terhubung dengan proxy yang ada, Amazon RDS memperbarui grup keamanan proxy untuk mengizinkan koneksi dari cluster dan fungsi Lambda Anda.

Anda juga dapat membuat proksi baru dari halaman konsol yang sama. Saat Anda membuat proxy di konsol, untuk mengakses cluster DB, Anda harus memasukkan kredenal database Anda atau memilih rahasia. AWS Secrets Manager




**Tip**

Untuk menghubungkan fungsi Lambda dengan cepat ke cluster Aurora DB, Anda juga dapat menggunakan panduan panduan dalam konsol. Untuk membuka wizard, lakukan hal berikut:

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih fungsi yang ingin Anda sambungkan ke database.
3. Pada tab Konfigurasi, pilih RDSdatabase.
4. Pilih Connect to RDS database.

Setelah menghubungkan fungsi ke database, Anda dapat membuat proxy dengan memilih Add proxy.

Topik


- Ikhtisar konektivitas otomatis dengan fungsi Lambda
- Secara otomatis menghubungkan fungsi Lambda dan database DB cluster
- Melihat sumber daya komputasi terhubung


## Ikhtisar konektivitas otomatis dengan fungsi Lambda

Berikut ini adalah persyaratan untuk menghubungkan fungsi Lambda dengan instance DB cluster :

- Fungsi Lambda harus ada VPC sama dengan cluster .
- Saat ini, cluster DB tidak bisa menjadi Aurora Serverless Cluster DB atau bagian dari database global Aurora.
- Pengguna yang mengatur konektivitas harus memiliki izin untuk melakukan AmazonRDS, Amazon, LambdaEC2, Secrets Manager, IAM dan operasi berikut:
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBClusters`
    - `rds:DescribeDBProxies`
    - `rds:ModifyDBCluster`
    - `rds:ModifyDBProxy`
    - `rds:RegisterProxyTargets`
  - Amazon EC2
    - `ec2:AuthorizeSecurityGroupEgress`
    - `ec2:AuthorizeSecurityGroupIngress`
    - `ec2:CreateSecurityGroup`
    - `ec2:DeleteSecurityGroup`
    - `ec2:DescribeSecurityGroups`
    - `ec2:RevokeSecurityGroupEgress`
    - `ec2:RevokeSecurityGroupIngress`
  - Lambda
    - `lambda:CreateFunctions`
    - `lambda:ListFunctions`

    - `lambda:UpdateFunctionConfiguration`
- Secrets Manager
    - `secretsmanager:CreateSecret`
    - `secretsmanager:DescribeSecret`
- IAM
    - `iam:AttachPolicy`
    - `iam:CreateRole`
    - `iam:CreatePolicy`
- AWS KMS
    - `kms:describeKey`

> **Note**
>
> Jika klaster DB dan fungsi Lambda berada di Zona Ketersediaan yang berbeda, akun Anda mungkin dikenakan biaya lintas Zona Ketersediaan.

Jika Anda menggunakan RDS Proxy, Amazon RDS juga mengonfigurasi grup VPC keamanan untuk proxy tersebut. Amazon RDS bertindak sesuai dengan konfigurasi grup keamanan saat ini yang terkait dengan cluster DB, fungsi Lambda, dan proxy, seperti yang dijelaskan dalam tabel berikut.

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
|---|---|---|---|
| Ada satu atau lebih grup keamanan yang terkait dengan cluster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika proxy sudah terhubung ke cluster DB Anda, RDS | Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rd` | Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n* (dengan *n* berupa angka). | Amazon tidak RDS mengambil tindakan.<br><br>Koneksi sudah dikonfigurasikan secara otomatis antara fungsi Lambda, proksi (opsional), dan klaster DB. Karena koneksi sudah ada |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
| --- | --- | --- | --- |
| periksa apakah proxy terkait`AVAILABLE` . `TargetHealth`<br><br>Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. | `sproxy-` *n* (dengan *n* berupa angka).<br><br>Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan keluar dengan grup VPC keamanan cluster DB atau proxy sebagai tujuan. | Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki aturan masuk dan keluar dengan grup VPC keamanan fungsi Lambda dan cluster DB instance . | antara fungsi, proksi, dan basis data, grup keamanan tidak diubah. |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
|---|---|---|---|
| Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. Namun, tidak satu pun grup keamanan ini dapat digunakan untuk koneksi dengan fungsi Lambda. | Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster DB.<br><br>Amazon tidak RDS dapat menggunak | Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan `rdsproxy-lambda-` *n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster atau fungsi Lambda.<br><br>Amazon tidak RDS dapat menggunak an grup keamanan | RDS action: create new security groups |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
| --- | --- | --- | --- |
| Amazon tidak RDS dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. Amazon RDS juga tidak dapat menggunak an grup keamanan yang telah dimodifik asi. Contoh-contoh perubahan meliputi penambahan aturan atau pengubahan port aturan yang ada. | an grup keamanan yang tidak memiliki satu aturan keluar dengan grup VPC keamanan cluster DB atau proxy sebagai tujuan. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | yang tidak memiliki aturan masuk dan keluar dengan grup VPC keamanan cluster dan fungsi Lambda. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
|---|---|---|---|
| Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE` .<br><br>Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. | Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*.<br><br>Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster DB. Amazon tidak RDS dapat menggunakan grup keamanan yang tidak memiliki satu aturan keluar dengan grup VPC keamanan cluster DB atau proxy sebagai tujuan. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*.<br><br>Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster atau fungsi Lambda. Amazon tidak RDS dapat menggunakan grup keamanan yang tidak memiliki aturan masuk dan keluar dengan grup VPC keamanan cluster dan fungsi Lambda. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | RDS action: create new security groups |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
|---|---|---|---|
| Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE` .<br><br>Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. | Ada grup keamanan Lambda yang valid untuk koneksi, tetapi tidak dikaitkan dengan fungsi Lambda. Grup keamanan ini memiliki nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Grup tersebut belum diubah. Ini hanya memiliki satu aturan keluar dengan grup VPC keamanan cluster DB atau proxy sebagai tujuan. | Ada grup keamanan proksi yang valid untuk koneksi, tetapi tidak dikaitkan dengan proksi. Grup keamanan ini memiliki nama yang cocok dengan pola `rdsproxy-lambda-` *n*. Grup tersebut belum diubah. Ini memiliki aturan masuk dan keluar dengan grup VPC keamanan cluster dan fungsi Lambda. | RDS action: associate Lambda security group |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
|---|---|---|---|
| Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan | Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*.<br>Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan keluar dengan grup VPC keamanan DB atau proxy sebagai tujuan. | Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*.<br>Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki aturan masuk dan keluar dengan grup VPC keamanan cluster dan fungsi Lambda. | RDS action: create new security groups |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
| --- | --- | --- | --- |
| fungsi atau proxy Lambda.<br><br>Amazon tidak RDS dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | | | |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
|---|---|---|---|
| Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan | Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster DB.<br><br>Amazon tidak RDS dapat menggunak | Salah satu syarat berikut berlaku:<br>• Tidak ada grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*.<br>• Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan `rdsproxy-lambda-` *n*. Namun, Amazon tidak RDS dapat menggunakan salah satu grup keamanan ini untuk koneksi dengan cluster atau fungsi Lambda.<br><br>Amazon tidak RDS dapat menggunak an grup keamanan | RDS action: create new security groups |

| Konfigurasi grup RDS keamanan saat ini | Konfigurasi grup keamanan Lambda saat ini | Konfigurasi grup keamanan proksi saat ini | Tindakan RDS |
| --- | --- | --- | --- |
| fungsi atau proxy Lambda.<br><br>Amazon tidak RDS dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | an grup keamanan yang tidak memiliki satu aturan keluar dengan grup VPC keamanan cluster atau proxy sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | yang tidak memiliki aturan masuk dan keluar dengan grup VPC keamanan cluster dan fungsi Lambda. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah dimodifikasi. | |

RDStindakan: buat grup keamanan baru

Amazon RDS mengambil tindakan berikut:

- Membuat grup keamanan baru yang cocok dengan pola `rds-lambda-`*n* atau `rds-rdsproxy-`*n* (jika Anda memilih untuk menggunakan RDS Proxy). Grup keamanan ini memiliki aturan masuk dengan grup VPC keamanan fungsi Lambda atau proxy sebagai sumbernya. Grup keamanan ini dikaitkan dengan klaster DB dan memungkinkan fungsi atau proksi mengakses klaster DB.
- Membuat grup keamanan baru yang cocok dengan pola `lambda-rds-`*n* atau `lambda-rdsproxy-`*n*. Grup keamanan ini memiliki aturan keluar dengan grup VPC keamanan cluster DB atau proxy sebagai tujuan. Grup keamanan ini dikaitkan dengan fungsi Lambda dan memungkinkan fungsi mengirim lalu lintas ke klaster DB atau mengirim lalu lintas melalui proksi.
- Membuat grup keamanan baru yang cocok dengan pola `rdsproxy-lambda-`*n*. Grup keamanan ini memiliki aturan masuk dan keluar dengan grup VPC keamanan cluster dan fungsi Lambda.

RDStindakan: asosiasi kelompok keamanan Lambda

Amazon RDS mengaitkan grup keamanan Lambda yang valid dan sudah ada dengan fungsi Lambda. Grup keamanan ini memungkinkan fungsi mengirim lalu lintas ke klaster DB atau mengirim lalu lintas melalui proksi.

## Secara otomatis menghubungkan fungsi Lambda dan database DB cluster

Anda dapat menggunakan RDS konsol Amazon untuk secara otomatis menghubungkan fungsi Lambda ke cluster Anda. Ini menyederhanakan proses penyiapan koneksi di antara sumber daya-sumber daya ini.

Anda juga dapat menggunakan RDS Proxy untuk menyertakan proxy dalam koneksi Anda. Fungsi Lambda sering membuat koneksi database pendek yang mendapat manfaat dari penyatuan koneksi yang ditawarkan Proxy. RDS Anda juga dapat menggunakan IAM otentikasi apa pun yang telah Anda siapkan untuk fungsi Lambda Anda, alih-alih mengelola kredensi database dalam kode aplikasi Lambda Anda.

Anda dapat menghubungkan klaster DB yang ada dengan fungsi Lambda baru dan lama dengan menggunakan halaman Siapkan koneksi Lambda. Proses penyiapan menyiapkan secara otomatis grup keamanan yang Anda perlukan.

Sebelum menyiapkan koneksi antara fungsi Lambda dan klaster DB, pastikan bahwa:

- Fungsi Lambda dan cluster  Anda berada dalam keadaan yang sama. VPC
- Anda memiliki izin-izin yang tepat untuk akun pengguna Anda. Lihat informasi lebih lanjut tentang persyaratan di Ikhtisar konektivitas otomatis dengan fungsi Lambda.

Jika Anda mengubah grup keamanan setelah mengonfigurasikan konektivitas, perubahan itu dapat memengaruhi koneksi antara fungsi Lambda dan klaster DB.

> **Note**
>
> Anda dapat secara otomatis mengatur koneksi antara cluster DB dan fungsi Lambda hanya di. AWS Management Console Untuk menghubungkan fungsi Lambda, semua instans di klaster DB harus dalam keadaan Tersedia.

Untuk menghubungkan secara otomatis fungsi Lambda dan klaster DB

<result>

Setelah mengonfirmasi penyiapan, Amazon RDS memulai proses menghubungkan fungsi Lambda, RDS Proxy (jika Anda menggunakan proxy), dan cluster . Konsol menampilkan kotak dialog Detail koneksi, yang mencantumkan perubahan grup keamanan yang memungkinkan koneksi di antara sumber daya-sumber daya Anda.
</result>

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB yang ingin Anda hubungkan dengan fungsi Lambda.
3. Untuk Tindakan, pilih Siapkan koneksi Lambda.
4. Pada halaman Siapkan koneksi Lambda, di bawah Pilih fungsi Lambda, lakukan salah satu hal berikut:
   - Jika Anda memiliki fungsi Lambda yang ada VPC sama dengan cluster  Anda, pilih Pilih fungsi yang ada, lalu pilih fungsinya.
   - Jika Anda tidak memiliki fungsi Lambda yang samaVPC, pilih Buat fungsi baru, lalu masukkan nama Fungsi. Runtime default diatur ke Nodejs.18. Anda dapat mengubah setelan untuk fungsi Lambda baru di konsol Lambda setelah menyelesaikan penyiapan koneksi.
5. (Opsional) Di bawah RDSProxy, pilih Connect using RDS Proxy, lalu lakukan salah satu hal berikut:
   - Jika Anda sudah memiliki proksi yang ingin Anda gunakan, pilih Pilih proksi yang ada, lalu pilih proksi itu.
   - Jika Anda tidak memiliki proxy, dan Anda ingin Amazon membuatnya secara otomatis RDS untuk Anda, pilih Buat proxy baru. Lalu, untuk Kredensial basis data, lakukan salah satu langkah berikut:

     a. Pilih Nama pengguna dan kata sandi basis data, lalu masukkan Nama pengguna dan Kata sandi untuk klaster DB Anda.

     b. Pilih Rahasia Secrets Manager. Kemudian, untuk Pilih rahasia, pilih AWS Secrets Manager rahasia. Jika Anda tidak memiliki rahasia Secrets Manager, pilih Buat rahasia Secrets Manager baru untuk membuat rahasia baru. Setelah Anda membuat rahasia, untuk Pilih rahasia, pilih rahasia baru.

Setelah Anda membuat proksi baru, pilih Pilih proksi yang ada, lalu pilih proksi. Perhatikan bahwa mungkin perlu beberapa waktu sebelum proksi Anda tersedia untuk koneksi.

6. (Opsional) Perluas Ringkasan koneksi dan periksa pembaruan yang disorot untuk sumber daya Anda.
7. Pilih Siapkan.

## Melihat sumber daya komputasi terhubung

Anda dapat menggunakan AWS Management Console untuk melihat fungsi Lambda yang terhubung ke cluster Anda. Sumber daya yang ditampilkan mencakup koneksi sumber daya komputasi yang RDS disiapkan Amazon secara otomatis.

Sumber daya komputasi tercantum yang tidak menyertakan sumber daya yang dihubungkan secara manual dengan klaster DB. Misalnya, Anda dapat mengizinkan sumber daya komputasi untuk mengakses cluster DB Anda secara manual dengan menambahkan aturan ke grup VPC keamanan yang terkait dengan database.

Agar konsol menampilkan suatu fungsi Lambda, kondisi-kondisi berikut harus terpenuhi:

- Nama grup keamanan yang terkait dengan sumber daya komputasi cocok dengan pola `lambda-rds-`*n* atau `lambda-rdsproxy-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan sumber daya komputasi memiliki aturan keluar dengan rentang port yang diatur ke port klaster DB atau proksi terkait. Tujuan untuk aturan keluar harus diatur ke grup keamanan yang terkait dengan klaster DB atau proksi terkait.
- Jika konfigurasi menyertakan proksi, nama grup keamanan yang dilampirkan pada proksi yang terkait dengan basis data Anda cocok dengan pola `rdsproxy-lambda-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan fungsi memiliki aturan keluar dengan port yang diatur ke port yang digunakan oleh klaster DB atau proksi terkait. Tujuan harus diatur ke grup keamanan yang terkait dengan klaster DB atau proksi terkait.

Untuk melihat sumber daya komputasi yang dihubungkan secara otomatis dengan klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB.
3. Pada tab Konektivitas & keamanan, lihat sumber daya komputasi di bawah Sumber daya komputasi terhubung.

# Memodifikasi klaster DB Amazon Aurora

Anda dapat mengubah pengaturan klaster DB untuk menyelesaikan tugas seperti mengubah periode retensi cadangan atau port basis datanya. Anda juga dapat memodifikasi instans DB dalam klaster DB untuk menyelesaikan tugas seperti mengubah kelas instans DB atau mengaktifkan Wawasan Performa untuknya. Topik ini memandu Anda dalam memodifikasi klaster DB Aurora dan instans DB-nya, serta menjelaskan masing-masing pengaturannya.

Kami menyarankan Anda menguji setiap perubahan pada instans klaster DB uji atau instans DB sebelum memodifikasi instans DB atau klaster DB produksi, sehingga Anda sepenuhnya memahami dampak dari setiap perubahan. Hal ini sangat penting ketika meningkatkan versi basis data.

Topik


- Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API
- Memodifikasi instans DB dalam klaster DB
- Mengubah kata sandi untuk pengguna master database
- Pengaturan untuk Amazon Aurora
- Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora
- Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora


## Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API

Anda dapat memodifikasi cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

> **Note**
>
> Sebagian besar modifikasi dapat diterapkan segera atau selama periode pemeliharaan terjadwal berikutnya. Beberapa modifikasi, seperti mengaktifkan perlindungan penghapusan, akan diterapkan segera, terlepas dari kapan Anda memilih untuk menerapkannya. Mengubah kata sandi utama di AWS Management Console selalu diterapkan segera. Jika Anda menggunakan titik akhir SSL dan mengubah pengidentifikasi klaster DB, hentikan dan mulai ulang klaster DB untuk memperbarui titik akhir SSL. Untuk informasi selengkapnya, lihat Menghentikan dan memulai klaster DB Amazon Aurora.

Konsol

Untuk memodifikasi klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin Anda modifikasi.
3. Pilih Ubah. Halaman Modifikasi klaster DB akan muncul.
4. Ubah pengaturan apa pun yang Anda inginkan. Lihat informasi tentang setiap setelan di Pengaturan untuk Amazon Aurora.

   > **Note**
   >
   > Dalam AWS Management Console, beberapa perubahan tingkat instance hanya berlaku untuk instans DB saat ini, sementara yang lain berlaku untuk seluruh cluster DB. Untuk informasi tentang apakah pengaturan berlaku pada instans DB atau klaster DB, lihat cakupan untuk pengaturan tersebut dalam Pengaturan untuk Amazon Aurora. Untuk mengubah pengaturan yang memodifikasi seluruh cluster DB pada tingkat instans di AWS Management Console, ikuti instruksi diMemodifikasi instans DB dalam klaster DB.

5. Jika semua perubahan sudah sesuai dengan keinginan Anda, pilih Lanjutkan dan periksa ringkasan modifikasi.
6. Untuk menerapkan perubahan dengan segera, pilih Terapkan segera.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Ubah klaster untuk menyimpan perubahan Anda.

   Alternatifnya, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan Anda.

AWS CLI

Untuk memodifikasi cluster DB menggunakan AWS CLI, panggil modify-db-clusterperintah. Tentukan pengidentifikasi klaster DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan hanya berlaku untuk instans DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi instans DB dalam klaster DB.

Example

Perintah berikut memodifikasi `mydbcluster` dengan mengatur periode retensi pencadangan menjadi 1 minggu (7 hari).

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 7
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

API RDS

Untuk memodifikasi cluster DB menggunakan Amazon RDS API, panggil DBCluster operasi Modify. Tentukan pengidentifikasi klaster DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap parameter, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan hanya berlaku untuk instans DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi instans DB dalam klaster DB.

## Memodifikasi instans DB dalam klaster DB

Anda dapat memodifikasi instans DB di cluster DB menggunakan AWS CLI,, atau RDS API. AWS Management Console

Saat Anda memodifikasi instans DB, Anda dapat langsung menerapkan perubahan. Untuk segera menerapkan perubahan, Anda memilih opsi Terapkan Segera di AWS Management Console, Anda menggunakan `--apply-immediately` parameter saat memanggil AWS CLI, atau Anda mengatur `ApplyImmediately` parameter `true` saat menggunakan Amazon RDS API.

Jika Anda tidak memilih untuk segera menerapkan perubahan, perubahan akan ditunda hingga periode pemeliharaan berikutnya. Selama periode pemeliharaan berikutnya, perubahan yang ditunda ini akan diterapkan. Jika Anda memilih untuk segera menerapkan perubahan, perubahan baru dan setiap perubahan yang sebelumnya ditunda akan diterapkan.

Untuk melihat modifikasi yang tertunda untuk jendela pemeliharaan berikutnya, gunakan describe-db-clusters AWS CLI perintah dan periksa `PendingModifiedValues` bidangnya.

> **Important**
>
> Jika salah satu dari modifikasi yang ditunda memerlukan waktu henti, memilih Terapkan segera dapat menyebabkan waktu henti yang tidak terduga untuk instans DB. Tidak ada waktu henti untuk instans DB lain dalam klaster DB.
> Modifikasi yang Anda tunda tidak tercantum dalam output dari perintah CLI `describe-pending-maintenance-actions`. Tindakan pemeliharaan hanya mencakup peningkatan sistem yang Anda jadwalkan untuk periode pemeliharaan berikutnya.

Konsol

Untuk memodifikasi instans DB dalam klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin Anda modifikasi.
3. Untuk Tindakan, pilih Modifikasi. Halaman Modifikasi instans DB muncul.
4. Ubah pengaturan apa pun yang Anda inginkan. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan berlaku untuk keseluruhan klaster DB dan harus diubah pada tingkat klaster. Untuk mengubah pengaturan tersebut, ikuti petunjuk di Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.
> Dalam AWS Management Console, beberapa perubahan tingkat instance hanya berlaku untuk instans DB saat ini, sementara yang lain berlaku untuk seluruh cluster DB. Untuk informasi tentang apakah pengaturan berlaku pada instans DB atau klaster DB, lihat cakupan untuk pengaturan tersebut dalam Pengaturan untuk Amazon Aurora.

5. Jika semua perubahan sudah sesuai dengan keinginan Anda, pilih Lanjutkan dan periksa ringkasan modifikasi.
6. Untuk menerapkan perubahan dengan segera, pilih Terapkan segera.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Modifikasi instans DB untuk menyimpan perubahan Anda.

   Alternatifnya, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan Anda.

AWS CLI

Untuk memodifikasi instans DB di cluster DB dengan menggunakan AWS CLI, panggil modify-db-instanceperintah. Tentukan pengidentifikasi instans DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap parameter, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan berlaku untuk seluruh klaster DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

Example

Kode berikut memodifikasi `mydbinstance` dengan menetapkan kelas instans DB untuk `db.r4.xlarge`. Perubahan akan diterapkan selama periode pemeliharaan berikutnya dengan

menggunakan `--no-apply-immediately`. Gunakan `--apply-immediately` untuk segera menerapkan perubahan.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

API RDS

Untuk memodifikasi instans DB dengan menggunakan Amazon RDS API, panggil DBInstance operasi Modify. Tentukan pengidentifikasi instans DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap parameter, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan berlaku untuk seluruh klaster DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

## Mengubah kata sandi untuk pengguna master database

Anda dapat menggunakan AWS Management Console atau AWS CLI untuk mengubah kata sandi pengguna utama.

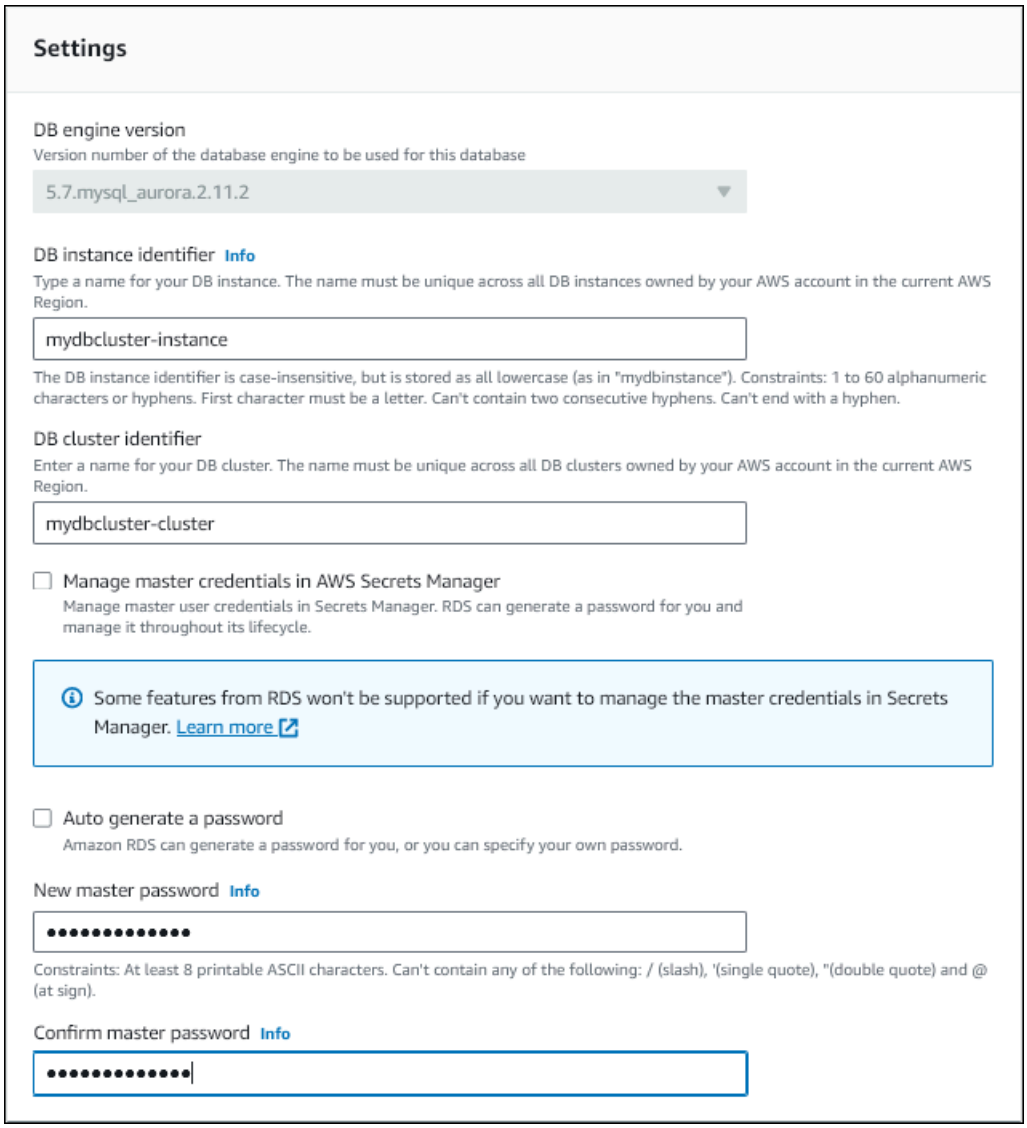
Konsol

Anda memodifikasi instance DB penulis untuk mengubah kata sandi pengguna master menggunakan AWS Management Console.

Untuk mengubah kata sandi pengguna utama

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin Anda modifikasi.
3. Untuk Tindakan, pilih Modifikasi.

   Halaman Modifikasi instans DB muncul.
4. Masukkan kata sandi master baru.
5. Untuk Konfirmasi kata sandi utama, masukkan kata sandi baru yang sama.




6. Pilih Lanjutkan dan periksa ringkasan modifikasi.

> **Note**
>
> Perubahan kata sandi selalu diterapkan segera.

7. Di halaman konfirmasi, pilih Modifikasi instans DB.

CLI

Anda memanggil modify-db-clusterperintah untuk mengubah kata sandi pengguna master menggunakan file AWS CLI. Tentukan pengidentifikasi cluster DB dan kata sandi baru, seperti yang ditunjukkan pada contoh berikut.

Anda tidak perlu menentukan`--apply-immediately|--no-apply-immediately`, karena perubahan kata sandi selalu diterapkan segera.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

## Pengaturan untuk Amazon Aurora

Tabel berikut berisi detail tentang pengaturan mana yang dapat Anda modifikasi, metode untuk memodifikasi pengaturan, dan cakupan pengaturan. Cakupan menentukan apakah pengaturan berlaku untuk keseluruhan klaster DB atau apakah pengaturan hanya dapat diatur untuk instans DB tertentu.

> **Note**
>
> Pengaturan tambahan tersedia jika Anda memodifikasi klaster DB Aurora Serverless v1 atau Aurora Serverless v2. Untuk informasi tentang pengaturan ini, lihat Memodifikasi Aurora Serverless v1 Klaster DB dan Mengelola Aurora Serverless v2 Klaster DB.

Beberapa pengaturan tidak tersedia untuk Aurora Serverless v1 dan Aurora Serverless v2 karena batasannya. Untuk informasi selengkapnya, lihat Pembatasan Aurora Serverless v1 dan Persyaratan dan batasan untuk Aurora Serverless v2.

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
|---|---|---|---|
| Peningkatan versi minor otomatis<br><br>Hal ini menentukan apakah Anda ingin instans DB menerima peningkatan versi mesin minor pilihan secara otomatis ketika peningkatan tersebut tersedia. Peningkatan hanya diinstal selama periode pemeliharaan terjadwal Anda.<br><br>Untuk informasi selengkapnya tentang pembaruan mesin, lihat Pembaruan mesin database untuk Amazon Aurora Postgre SQL dan Pembaruan mesin database untuk Amazon Aurora My SQL. Untuk informasi selengkapnya tentang pengatura n Peningkatan versi | **Note**<br>Pengaturan ini diaktifka n secara default. Untuk setiap klaster baru, pilih nilai yang sesuai untuk pengaturan ini berdasark an tingkat kepenting annya, masa penggunaa n yang diharapkan, dan jumlah pengujian verifikasi yang Anda lakukan setelah setiap peningkatan.<br><br>Saat Anda mengubah pengaturan ini, | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. Pemadaman terjadi selama periode pemeliharaan mendatang saat Aurora menerapkan peningkatan otomatis. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| minor otomatis untuk Aurora MySQL, lihat Mengaktifkan upgrade otomatis antara Aurora minor versi Saya SQL. | lakukan modifikasi ini untuk setiap instans DB di klaster Aurora Anda. Jika ada instans DB di klaster Anda yang menonaktifkan pengaturan ini, klaster tersebut tidak ditingkatkan secara otomatis.<br><br>Menggunakan AWS Management Console,Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan modify-db-instancedan atur `--auto-minor-version-upgrade\|--no-auto-minor-version-upgrade` opsi.<br><br>Menggunakan RDS API, panggil Modify DBInstance dan atur `AutoMinorVersionUpgrade` parameternya. | | |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Periode retensi cadangan<br><br>Jumlah hari retensi cadangan otomatis. Nilai minimum adalah 1.<br><br>Untuk informasi selengkapnya, lihat Cadangan. | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan modify-db-clusterdan atur `--backup-retention-period` opsi.<br><br>Menggunakan RDS API, panggil Modify DBCluster dan atur `BackupRetentionPeriod` parameternya. | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Jendela cadangan (Waktu mulai)<br><br>Rentang waktu saat pencadangan otomatis basis data Anda terjadi. Jendela cadangan adalah waktu mulai dalam Waktu Universal (UTC) Terkoordinasi, dan durasi dalam jam.<br><br>Cadangan Aurora bersifat terus-men erus dan bertahap, tetapi periode pencadangan digunakan untuk membuat cadangan sistem harian yang dipertahankan dalam periode retensi cadangan. Anda dapat menyalin cadangan ini untuk mempertahankannya di luar periode retensi.<br><br>Jendela pemeliharaan dan jendela cadangan untuk klaster DB tidak dapat tumpang tindih. | Menggunakan AWS Management Console,Memodifik asi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan modify-db-clusterdan atur `--preferred-backup-window` opsi.<br><br>Menggunakan RDS API, panggil Modify DBCluster dan atur `Preferred BackupWindow` parameternya. | Seluruh klaster DB. | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Untuk informasi selengkapnya, lihat Jendela cadangan. | | | |
| Pengaturan kapasitas<br><br>Properti penskalaan klaster DB Aurora Serverless v1. Anda hanya dapat memodifikasi properti penskalaan untuk klaster DB dalam mode mesin DB `serverless` .<br><br>Untuk informasi tentang Aurora Serverless v1, lihat Menggunakan Amazon Aurora Serverless v1. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan modify-db-cluster dan atur `--scaling-configuration` opsi.<br><br>Menggunakan RDS API, panggil Modify DBCluster dan atur `ScalingConfiguration` parameternya. | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini.<br><br>Perubahan terjadi dengan segera. Pengaturan ini mengabaikan pengaturan "Terapkan segera". |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Otoritas sertifikat<br><br>Otoritas sertifikat (CA) untuk sertifikat server yang digunakan oleh instans DB. | Menggunakan AWS Management Console,Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan modify-db-instance dan atur `--ca-certificate-identifier` opsi.<br><br>Menggunakan RDS API, panggil Modify DBInstance dan atur `CACertificateIdentifier` parameternya. | Hanya instans DB yang ditentukan | Pemadaman hanya terjadi jika mesin DB tidak mendukung rotasi tanpa pengaktifan ulang. Anda dapat menggunakan describe-db-engine-versions AWS CLI perintah untuk menentukan apakah mesin DB mendukung rotasi tanpa restart. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Konfigurasi penyimpanan klaster<br><br>Jenis penyimpanan untuk klaster DB: Aurora I/O-Optimized atau Aurora Standard.<br><br>Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan modify-db-clusterdan atur `--storage-type` opsi.<br><br>Menggunakan RDS API, panggil Modify DBCluster dan atur `StorageType` parameternya. | Seluruh klaster DB | Mengubah jenis penyimpanan cluster Aurora PostgreSQL DB dengan kelas instance Optimized Reads menyebabkan pemadaman. Ini tidak terjadi saat mengubah tipe penyimpanan untuk cluster dengan tipe kelas instance lainnya. Untuk informasi lebih lanjut tentang jenis kelas instans DB, lihatJenis kelas instans DB. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Salin tanda ke snapshot<br><br>Pilih untuk menentukan agar tanda/tag yang ditentukan untuk klaster DB ini disalin ke snapshot DB yang dibuat dari klaster DB ini. Untuk informasi selengkapnya, lihat Menandai sumber daya Amazon Aurora dan Amazon RDS. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html dan atur `--no-copy-tags-to-snapshot` opsi `--copy-tags-to-snapshot` or.<br><br>Menggunakan API RDS, panggil https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ModifyDBCluster.html dan tetapkan parameter `CopyTagsToSnapshot` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Data API<br><br>Anda dapat mengakses Aurora Serverless v1 dengan aplikasi berbasis layanan web, termasuk dan. AWS Lambda AWS AppSync<br><br>Pengaturan ini hanya berlaku untuk klaster DB Aurora Serverless v1.<br><br>Untuk informasi selengkapnya, lihat Menggunakan Amazon RDS Data API. | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--enable-http-endpoint` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `EnableHttpEndpoint` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Autentikasi basis data<br><br>Autentikasi basis data yang ingin Anda gunakan.<br><br>Untuk MySQL:<br><br>• Pilih Autentikasi kata sandi untuk mengautentikasi pengguna basis data dengan kata sandi basis data saja.<br>• Pilih Kata sandi dan autentikasi basis data IAM untuk mengautentikasi pengguna basis data dengan kata sandi basis data dan kredensial pengguna melalui pengguna dan peran IAM. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.<br><br>Untuk PostgreSQL:<br><br>• Pilih Autentikasi basis data IAM | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan modify-db-clusterdan atur opsi berikut:<br><br>• Untuk autentikasi IAM, atur opsi `--enable-iam-database-authentication\|--no-enable-iam-database-authentication` .<br>• Untuk autentikasi Kerberos, atur opsi `--domain` dan `--domain-iam-role-name` .<br><br>Menggunakan RDS API, panggil Modify DBCluster dan atur parameter berikut: | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| untuk mengauten tikasi pengguna basis data dengan kata sandi basis data dan kredensia l pengguna melalui pengguna dan peran. Untuk informasi selengkap nya, lihat Autentika si basis data IAM.<br>• Pilih Autentikasi Kerberos untuk mengautentikasi kata sandi basis data dan kredensia l pengguna menggunak an autentika si Kerberos. Untuk informasi selengkapnya, lihat Menggunak an autentikasi Kerberos dengan Aurora PostgreSQ L. | • Untuk autentikasi IAM, atur parameter `EnableIAM DatabaseA uthentica tion` .<br>• Untuk autentika si Kerberos, atur parameter `Domain` dan `DomainIAM RoleName` . | | |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Port basis data<br><br>Port yang ingin Anda gunakan untuk mengakses klaster DB. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--port` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `Port`. | Seluruh klaster DB | Pemadaman terjadi selama perubahan ini. Semua instans DB dalam klaster DB langsung di-boot ulang. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Pengidentifikasi klaster DB<br><br>Pengidentifikasi klaster DB. Nilai ini disimpan sebagai string huruf kecil.<br><br>Saat Anda mengubah pengidentifikasi klaster DB, titik akhir klaster DB berubah. Titik akhir instans DB di klaster DB tidak berubah. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--new-db-cluster-identifier` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `NewDBClusterIdentifier`. | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Grup parameter klaster DB<br><br>Grup parameter klaster DB yang ingin Anda kaitkan dengan klaster DB.<br><br>Untuk informasi selengkapnya, lihat . | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--db-cluster-parameter-group-name` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `DBClusterParameterGroupName` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. Ketika Anda mengubah grup parameter, perubahan pada beberapa parameter akan diterapkan ke instans DB dalam klaster DB segera tanpa boot ulang. Perubahan pada parameter lain diterapkan hanya setelah instans DB dalam klaster DB di-boot ulang. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Kelas instans DB<br><br>Kelas instans DB yang ingin Anda gunakan.<br><br>Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--db-instance-class` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `DBInstanceClass` . | Hanya instans DB yang ditentukan | Pemadaman terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Pengidentifikasi instans DB<br><br>Pengidentifikasi instans DB. Nilai ini disimpan sebagai string huruf kecil. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--new-db-instance-identifier` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `NewDBInstanceIdentifier` . | Hanya instans DB yang ditentukan | Waktu henti terjadi selama perubahan ini.<br><br>RDS memulai ulang instans DB untuk memperbarui yang berikut ini:<br><br>• Aurora MySQL - kolom dalam tabel `SERVER_ID` `information_schema.replica_host_status`<br>• Aurora PostgreSQL — kolom dalam fungsi `server_id` `aurora_replica_status()` |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Grup parameter DB<br><br>Grup parameter DB yang ingin Anda kaitkan dengan instans DB.<br><br>Untuk informasi selengkapnya, lihat . | Menggunakan AWS Management Console,Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--db-parameter-group-name` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `DBParameterGroupName` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini.<br><br>Ketika Anda mengaitkan grup parameter DB baru dengan instans DB, parameter statis dan dinamis yang dimodifikasi diterapkan hanya setelah instans DB di boot ulang. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini diterapkan segera tanpa boot ulang.<br><br>Untuk informasi lebih lanjut, lihat dan Memboot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Perlindungan penghapusan<br><br>Aktifkan perlindungan penghapusan agar klaster DB tidak terhapus. Untuk informasi selengkapnya, lihat Perlindungan penghapusan untuk klaster Aurora. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--deletion-protection\|--no-deletion-protection` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `DeletionProtection` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Versi mesin<br><br>Versi mesin DB yang ingin Anda gunakan. Sebelum meningkatkan klaster DB produksi Anda, sebaiknya Anda menguji proses peningkatan pada klaster DB uji untuk memverifikasi durasinya dan untuk memvalidasi aplikasi Anda. | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--engine-version` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `EngineVersion` . | Seluruh klaster DB | Pemadaman terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Pemantauan yang ditingkatkan<br><br>Aktifkan Pemantauan yang ditingkatkan untuk memungkinkan pengumpulan metrik secara waktu nyata untuk sistem operasi yang dijalankan oleh instans DB Anda.<br><br>Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, menjalankan `modify-db-instance` dan mengatur `--monitoring-role-arn` dan `--monitoring-interval` pilihan.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `MonitoringRoleArn` dan `MonitoringInterval` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Ekspor log<br><br>Pilih jenis log untuk dipublikasikan ke Amazon CloudWatch Logs.<br><br>Untuk informasi selengkapnya, lihat File log basis data Aurora MySQL. | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--cloudwatch-logs-export-configuration` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `CloudwatchLogsExportConfiguration` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Jendela pemeliharaan<br><br>Rentang waktu saat pemeliharaan sistem berlangsung. Pemeliharaan sistem mencakup peningkatan, jika ada. Jendela pemeliharaan adalah waktu mulai dalam Waktu Terkoordinasi Universal (UTC), dan durasi dalam jam.<br><br>Jika Anda mengatur jendela ke waktu saat ini, harus ada minimal 30 menit antara waktu saat ini dan akhir jendela untuk memastikan setiap perubahan yang tertunda diterapkan.<br><br>Anda dapat mengatur jendela pemeliharaan secara terpisah untuk klaster DB dan untuk setiap instans DB dalam klaster DB. Jika cakupan modifikasi adalah seluruh klaster DB, modifikasi dilakukan | Untuk mengubah jendela pemeliharaan untuk cluster DB menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Untuk mengubah jendela pemeliharaan untuk instans DB menggunakan AWS Management Console,Memodifikasi instans DB dalam klaster DB.<br><br>Untuk mengubah jendela pemeliharaan untuk cluster DB menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--preferred-maintenance-window` opsi.<br><br>Untuk mengubah jendela pemeliharaan untuk instans DB menggunakan | Seluruh klaster DB atau satu instans DB | Jika ada satu atau beberapa tindakan tertunda yang menyebabkan pemadaman, dan periode pemeliharaan diubah untuk menyertakan waktu saat ini, tindakan tertunda tersebut akan segera diterapkan, dan pemadaman pun terjadi. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| selama jendela pemeliharaan klaster DB. Jika cakupan modifikasi adalah instans DB, modifikasi dilakukan selama jendela pemeliharaan instans DB tersebut.<br><br>Jendela pemeliharaan dan jendela cadangan untuk klaster DB tidak dapat tumpang tindih.<br><br>Untuk informasi selengkapnya, lihat Jendela pemeliharaan Amazon RDS. | AWS CLI, jalankan `modify-db-instance` dan atur `--preferred-maintenance-window` opsi.<br><br>Untuk mengubah jendela pemeliharaan untuk klaster DB menggunakan API RDS, panggil `ModifyDBCluster` dan atur parameter `PreferredMaintenanceWindow` .<br><br>Untuk mengubah jendela pemeliharaan untuk instans DB menggunakan API RDS, panggil `ModifyDBInstance` dan atur parameter `PreferredMaintenanceWindow` . | | |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Kelola kredensi master di AWS Secrets Manager<br><br>Pilih Kelola kredensial master di AWS Secrets Manager untuk mengelola kata sandi pengguna master dalam rahasia di Secrets Manager.<br><br>Anda juga dapat memilih kunci KMS yang akan digunakan untuk melindungi rahasia. Pilih dari kunci KMS di akun Anda, atau masukkan kunci dari akun yang berbeda.<br><br>Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.<br><br>Jika Aurora sudah mengelola kata sandi pengguna master untuk klaster DB, Anda dapat merotasi kata sandi pengguna | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, menjalankan `modify-db-cluster` dan mengatur `--manage-master-user-password \| --no-manage-master-user-password` dan `--master-user-secret-kms-key-id` pilihan. Untuk segera merotasi kata sandi pengguna master, atur opsi `--rotate-master-user-password` .<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `ManageMasterUserPassword` dan `MasterUse` | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| master dengan memilih Putar rahasia segera.<br><br>Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. | `rSecretKmsKeyId` . Untuk segera merotasi kata sandi pengguna master, atur parameter `RotateMasterUserPassword` ke `true`. | | |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Jenis jaringan<br><br>Protokol alamat IP yang didukung oleh klaster DB.<br><br>IPv4untuk menentukan bahwa sumber daya dapat berkomunikasi dengan cluster DB hanya melalui protokol IPv4 pengalamatan.<br><br>Mode dual-stack untuk menentukan bahwa sumber daya dapat berkomunikasi dengan cluster DB over IPv4, IPv6, atau keduanya. Gunakan mode dual-stack jika Anda memiliki sumber daya yang harus berkomunikasi dengan cluster DB Anda melalui protokol IPv6 pengalamatan. Untuk menggunakan mode dual-stack, pastikan setidaknya dua subnet yang mencakup dua Availability Zone yang | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--network-type` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `NetworkType` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| mendukung protokol dan jaringan. IPv4 IPv6 Juga, pastikan Anda mengaitkan blok IPv6 CIDR dengan subnet di grup subnet DB yang Anda tentukan.<br><br>Untuk informasi selengkapnya, lihat Penentuan alamat IP Amazon Aurora. | | | |
| Kata sandi master baru<br><br>Kata sandi untuk pengguna master Anda.<br><br>• Untuk Aurora MySQL, kata sandi harus berisi 8–41 karakter ASCII yang dapat dicetak.<br>• Untuk Aurora PostgreSQL, kata sandi harus berisi 8–99 karakter ASCII yang dapat dicetak.<br>• Kata sandi tidak dapat berisi /, ", @, atau spasi. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--master-user-password` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `MasterUserPassword` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Wawasan Performa<br><br>Opsi untuk mengaktifkan Wawasan Performa, alat yang memantau beban instans DB Anda sehingga Anda dapat menganalisis dan memecahkan masalah performa basis data Anda.<br><br>Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--enable-performance-insights\|--no-enable-performance-insights` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `EnablePerformanceInsights` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Wawasan Performa AWS KMS key<br><br>AWS KMS key Pengidentifikasi untuk enkripsi data Performance Insights. Pengidentifikasi kunci KMS adalah Amazon Resource Name (ARN), pengident ifikasi kunci, atau alias kunci untuk kunci KMS.<br><br>Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora. | Menggunakan AWS Management Console,Memodifik asi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db- instance` dan atur `--performance- insights-kms- key-id` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBI nstance` dan tetapkan parameter `Performan ceInsight sKMSKeyId` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Periode retensi Wawasan Performa<br><br>Jumlah waktu, dalam hari, untuk mempertahankan data Wawasan Performa. Pengaturan retensi di tingkat gratis adalah Default (7 hari). Untuk mempertahankan data performa Anda lebih lama, tetapkan 1–24 bulan. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.<br><br>Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--performance-insights-retention-period` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `PerformanceInsightsRetentionPeriod` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Tingkat promosi<br><br>Nilai yang menentukan urutan Replika Aurora dipromosikan ke instans primer dalam klaster DB, setelah kegagalan instans primer yang ada.<br><br>Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora. | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--promotion-tier` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `PromotionTier` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Akses publik<br><br>Dapat diakses publik untuk memberikan alamat IP publik pada instans DB, yang berarti bahwa instans DB ini dapat diakses di luar VPC. Agar dapat diakses oleh publik, instans DB juga harus berada di subnet publik di VPC.<br><br>Tidak dapat diakses publik agar instans DB hanya dapat diakses dari dalam VPC.<br><br>Untuk informasi selengkapnya, lihat Menyembunyikan klaster DB dalam VPC dari internet.<br><br>Untuk terhubung ke instans DB dari luar Amazon VPC-nya, instans DB harus dapat diakses secara publik, akses harus diberikan menggunakan aturan masuk grup keamanan | Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB.<br><br>Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--publicly-accessible\|--no-publicly-accessible` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `PubliclyAccessible` . | Hanya instans DB yang ditentukan | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| instans DB, dan persyaratan lain harus dipenuhi. Untuk mengetahui informasi nya, lihat Tidak dapat terhubung ke instans DB Amazon RDS.<br><br>Jika instans DB Anda tidak dapat diakses publik, Anda juga dapat menggunak an koneksi AWS Site-to-Site VPN atau AWS Direct Connect koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan. | | | |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Pengaturan kapasitas Serverless v2<br><br>Kapasitas database cluster Aurora Serverless v2 DB, diukur dalam Unit Kapasitas Aurora ()ACUs.<br><br>Untuk informasi selengkapnya, lihat Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster. | Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--serverless-v2-scaling-configuration` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `ServerlessV2ScalingConfiguration` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini.<br><br>Perubahan terjadi dengan segera. Pengaturan ini mengabaikan pengaturan "Terapkan segera". |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Grup keamanan<br><br>Grup keamanan yang ingin Anda kaitkan dengan klaster DB.<br><br>Untuk informasi selengkapnya, lihat Mengontrol akses dengan grup keamanan. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--vpc-security-group-ids` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `VpcSecurityGroupIds` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

| Pengaturan dan deskripsi | Metode | Cakupan | Catatan waktu henti |
| --- | --- | --- | --- |
| Jendela Backtrack Target<br><br>Jumlah waktu yang Anda inginkan untuk melakukan backtrack klaster DB Anda, dalam hitungan detik. Pengaturan ini hanya tersedia untuk Aurora MySQL dan hanya jika klaster DB dibuat dengan Backtrack diaktifkan. | Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.<br><br>Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--backtrack-window` opsi.<br><br>Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `BacktrackWindow` . | Seluruh klaster DB | Pemadaman tidak akan terjadi selama perubahan ini. |

## Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `modify-db-cluster` dan operasi RDS API `ModifyDBCluster` tidak berlaku untuk cluster Amazon Aurora DB.

> **Note**
>
> Anda tidak dapat menggunakan AWS Management Console untuk memodifikasi pengaturan ini untuk cluster Aurora DB.

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| `--allocated-storage` | `AllocatedStorage` |
| `--auto-minor-version-upgrade \| --no-auto-minor-version-upgrade` | `AutoMinorVersionUpgrade` |
| `--db-cluster-instance-class` | `DBClusterInstanceClass` |
| `--enable-performance-insights \| --no-enable-performance-insights` | `EnablePerformanceInsights` |
| `--iops` | `Iops` |
| `--monitoring-interval` | `MonitoringInterval` |
| `--monitoring-role-arn` | `MonitoringRoleArn` |
| `--option-group-name` | `OptionGroupName` |
| `--performance-insights-kms-key-id` | `PerformanceInsightsKMSKeyId` |
| `--performance-insights-retention-period` | `PerformanceInsightsRetentionPeriod` |

## Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `modify-db-instance`dan operasi RDS API `ModifyDBInstance`tidak berlaku untuk instans Amazon Aurora DB.

> **Note**
>
> Anda tidak dapat menggunakan AWS Management Console untuk memodifikasi pengaturan ini untuk instans Aurora DB.

| AWS CLI pengaturan | Pengaturan API RDS |
| --- | --- |
| `--allocated-storage` | `AllocatedStorage` |
| `--allow-major-version-upgrade\|--no-allow-major-version-upgrade` | `AllowMajorVersionUpgrade` |
| `--copy-tags-to-snapshot\|--no-copy-tags-to-snapshot` | `CopyTagsToSnapshot` |
| `--domain` | `Domain` |
| `--db-security-groups` | `DBSecurityGroups` |
| `--db-subnet-group-name` | `DBSubnetGroupName` |
| `--domain-iam-role-name` | `DomainIAMRoleName` |
| `--multi-az\|--no-multi-az` | `MultiAZ` |
| `--iops` | `Iops` |
| `--license-model` | `LicenseModel` |
| `--network-type` | `NetworkType` |
| `--option-group-name` | `OptionGroupName` |
| `--processor-features` | `ProcessorFeatures` |
| `--storage-type` | `StorageType` |
| `--tde-credential-arn` | `TdeCredentialArn` |
| `--tde-credential-password` | `TdeCredentialPassword` |
| `--use-default-processor-features\|--no-use-default-processor-features` | `UseDefaultProcessorFeatures` |

# Menambahkan Replika Aurora ke klaster DB

Klaster DB Aurora dengan replikasi memiliki satu instans DB primer dan hingga 15 Replika Aurora. Instans DB primer mendukung operasi baca dan tulis, dan melakukan semua modifikasi data pada volume klaster. Replika Aurora terhubung ke volume penyimpanan yang sama dengan instans DB primer, tetapi hanya mendukung operasi baca. Anda menggunakan Replika Aurora untuk mengalihkan beban kerja baca dari instans DB primer. Untuk informasi selengkapnya, lihat Replika Aurora.

Replika Amazon Aurora memiliki batasan berikut ini:

- Anda tidak dapat membuat Replika Aurora untuk Aurora Serverless v1 klaster DB. Aurora Serverless v1 memiliki instans DB tunggal yang menskalakan naik dan turun secara otomatis untuk mendukung semua operasi baca dan tulis database.

  Namun, Anda dapat menambahkan instance pembaca ke Aurora Serverless v2 klaster DB. Untuk informasi selengkapnya, lihat Menambahkan Aurora Serverless v2 pembaca.

Kami menyarankan agar Anda mendistribusikan instans primer dan Replika Aurora dari klaster DB Anda ke beberapa Zona Ketersediaan untuk meningkatkan ketersediaan klaster DB Anda. Untuk informasi selengkapnya, lihat Ketersediaan wilayah.

Untuk menghapus Replika Aurora dari klaster DB Aurora, hapus Replika Aurora dengan mengikuti petunjuk dalam Menghapus instans dari klaster DB Aurora.

> **Note**
>
> Amazon Aurora juga mendukung replikasi dengan database eksternal, seperti instans DB. RDS Instans RDS DB harus sama AWS Wilayah sebagai Amazon Aurora. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

Anda dapat menambahkan Aurora Replicas ke cluster DB menggunakan AWS Management Console, AWS CLI, atau RDSAPI.

## Konsol

Untuk menambahkan Replika Aurora ke klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB tempat Anda ingin menambahkan instans DB baru.
3. Pastikan klaster dan instans primer berada dalam status Tersedia. Jika klaster DB atau instans primer berada dalam status transisi seperti Membuat, Anda tidak dapat menambahkan replika.

   Jika cluster tidak memiliki instance utama, buat satu menggunakan create-db-instance AWS CLI perintah. Situasi ini dapat muncul jika Anda menggunakan CLI untuk memulihkan snapshot cluster DB dan kemudian melihat cluster di AWS Management Console.
4. Untuk Tindakan, pilih Tambahkan pembaca.

   Halaman Tambahkan pembaca muncul.
5. Di halaman Tambahkan pembaca, tentukan opsi untuk Replika Aurora Anda. Tabel berikut menunjukkan pengaturan untuk Replika Aurora.

| Untuk opsi ini | Lakukan hal berikut |
| --- | --- |
| Zona ketersediaan | Tentukan apakah Anda ingin menentukan Zona Ketersediaan tertentu. Daftar ini hanya mencakup Zona Ketersediaan yang dipetakan ke grup subnet DB yang Anda pilih saat Anda membuat klaster DB. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan. |
| Dapat diakses publik | Pilih `Yes` untuk memberikan alamat IP publik pada Replika Aurora; jika tidak, pilih `No`. Untuk informasi selengkapnya tentang menyembunyikan Replika Aurora dari akses publik, lihat Menyembunyikan klaster DB dalam VPC dari internet. |
| Enkripsi | Pilih `Enable encryption` untuk mengaktifkan enkripsi saat diam untuk Replika Aurora ini. Untuk |

| Untuk opsi ini | Lakukan hal berikut |
| --- | --- |
| | informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora. |
| Kelas instans DB | Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk Replika Aurora. Untuk informasi selengkapnya tentang opsi kelas instans DB, lihat Kelas instans Amazon Aurora DB. |
| Sumber replika Aurora | Pilih pengidentifikasi instans primer yang akan dibuatkan Replika Aurora. |
| Pengidentifikasi instans DB | Masukkan nama untuk contoh yang unik untuk akun Anda di AWS Wilayah yang Anda pilih. Anda mungkin memilih untuk menambahkan beberapa kecerdasan ke nama seperti memasukkan AWS Region dan DB engine yang Anda pilih, misalnya**`aurora-read-instance1`** . |
| Prioritas | Pilih prioritas failover untuk instans. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora. |
| Port basis data | Port untuk Replika Aurora sama dengan port untuk klaster DB. |
| Grup parameter DB | Pilih grup parameter. Aurora memiliki grup parameter default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter Anda sendiri. Untuk informasi selengkapnya tentang grup parameter, lihat . |

| Untuk opsi ini | Lakukan hal berikut |
| --- | --- |
| Wawasan Performa | Kotak centang Aktifkan Wawasan Performa dipilih secara default. Nilai ini tidak diwarisi dari instans penulis. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. |
| Pemantauan yang ditingkatkan | Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |
| Peran Pemantauan | Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Pilih IAM peran yang Anda buat untuk mengizinkan Amazon RDS berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default untuk RDS membuat peran untuk nama Andards-monitoring-role . Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |
| Granularitas | Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda. |

| Untuk opsi ini | Lakukan hal berikut |
| --- | --- |
| Peningkatan versi minor otomatis | Pilih Aktifkan peningkatan versi minor otomatis jika Anda ingin klaster DB Aurora Anda menerima peningkatan versi Mesin DB minor secara otomatis saat tersedia.<br><br>Pengaturan pemutakhiran versi minor Otomatis berlaku untuk cluster Aurora Postgre SQL dan Aurora My DB. SQL Untuk klaster Aurora My SQL 2.x, pengaturan ini meningkatkan cluster ke versi maksimum 2.07.2.<br><br>Untuk informasi lebih lanjut tentang pembaruan mesin untuk Aurora SQL Postgre, lihat. Pembaruan mesin database untuk Amazon Aurora Postgre SQL<br><br>Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat. Pembaruan mesin database untuk Amazon Aurora My SQL |

6. Pilih Tambahkan pembaca untuk membuat Replika Aurora.

## AWS CLI

Untuk membuat Replika Aurora di cluster DB Anda, jalankan create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai opsi `--db-cluster-identifier`. Anda dapat menentukan Zona Ketersediaan secara opsional untuk Replika Aurora menggunakan parameter `--availability-zone` sebagaimana ditunjukkan dalam contoh berikut.

Misalnya, perintah berikut membuat Replika Aurora My SQL 5.7 yang kompatibel dengan nama. `sample-instance-us-west-2a`

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

Perintah berikut membuat Replika Aurora My SQL 5.7 yang kompatibel dengan nama. `sample-instance-us-west-2a`

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

Perintah berikut membuat Postgre SQL -compatible Aurora Replica baru bernama. `sample-instance-us-west-2a`

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
    --availability-zone us-west-2a
```

## RDS API

Untuk membuat Replika Aurora di cluster DB Anda, panggil operasi C. reateDBInstance Sertakan nama klaster DB sebagai parameter `DBClusterIdentifier`. Anda dapat menentukan Zona Ketersediaan secara opsional untuk Replika Aurora menggunakan parameter `AvailabilityZone`.

Untuk informasi tentang Auto Scaling Amazon Aurora dengan replika Aurora, lihat bagian berikut.

Topik

- Auto Scaling Amazon Aurora dengan Replika Aurora
- Menambahkan kebijakan penskalaan otomatis ke klaster DB Amazon Aurora
- Mengedit kebijakan penskalaan otomatis untuk klaster DB Amazon Aurora
- Menghapus kebijakan penskalaan otomatis dari klaster DB Amazon Aurora Anda

# Auto Scaling Amazon Aurora dengan Replika Aurora

Untuk memenuhi persyaratan konektivitas dan beban kerja Anda, Aurora Auto Scaling secara dinamis menyesuaikan jumlah Replika Aurora (instans DB pembaca) terprovisi untuk klaster DB Aurora. Aurora Auto Scaling tersedia untuk Aurora MySQL dan Aurora PostgreSQL. Aurora Auto Scaling memungkinkan klaster DB Aurora Anda menangani peningkatan mendadak dalam konektivitas atau beban kerja. Ketika konektivitas atau beban kerja berkurang, Aurora Auto Scaling menghapus Replika Aurora yang tidak perlu sehingga Anda tidak membayar untuk instans DB terprovisi yang tidak digunakan.

Anda menentukan dan menerapkan kebijakan penskalaan untuk klaster DB Aurora. Kebijakan penskalaan menentukan jumlah minimum dan maksimum Replika Aurora yang dapat dikelola oleh Aurora Auto Scaling. Berdasarkan kebijakan tersebut, Auto Scaling Aurora menyesuaikan jumlah Replika Aurora naik atau turun sebagai respons terhadap beban kerja aktual, yang ditentukan dengan menggunakan metrik Amazon dan nilai target. CloudWatch

> **Note**
>
> Auto Scaling Aurora tidak berlaku untuk beban kerja pada instans DB penulis. Auto Scaling Aurora membantu beban kerja hanya pada instans pembaca.

Anda dapat menggunakan AWS Management Console untuk menerapkan kebijakan penskalaan berdasarkan metrik yang telah ditentukan sebelumnya. Atau, Anda dapat menggunakan API Auto Scaling AWS CLI atau Aurora Auto untuk menerapkan kebijakan penskalaan berdasarkan metrik yang telah ditentukan atau kustom.

Topik


- Sebelum Anda mulai
- Kebijakan Aurora Auto Scaling
- Instans DB IDs dan penandaan
- Aurora Auto Scaling dan Wawasan Performa


## Sebelum Anda mulai

Sebelum Anda dapat menggunakan Aurora Auto Scaling dengan klaster DB Aurora, Anda terlebih dahulu harus membuat sebuah klaster DB Aurora dengan instans DB utama (penulis). Untuk

informasi selengkapnya tentang cara membuat klaster DB Aurora, lihat Membuat klaster DB Amazon Aurora.

Aurora Auto Scaling hanya menskalakan klaster DB jika klaster DB berada dalam status tersedia.

Saat Aurora Auto Scaling menambahkan Replika Aurora baru, Replika Aurora yang baru adalah kelas instans DB yang sama dengan yang digunakan oleh instans primer. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB. Selain itu, tingkat promosi untuk Replika Aurora yang baru akan ditetapkan sesuai prioritas terakhir, yaitu 15 secara default. Ini berarti bahwa selama failover, replika dengan prioritas lebih baik, seperti yang dibuat secara manual, akan dipromosikan terlebih dahulu. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.

Aurora Auto Scaling hanya menghapus Replika Aurora yang dibuatnya.

Untuk memanfaatkan Aurora Auto Scaling, aplikasi Anda harus mendukung koneksi ke Replika Aurora baru. Untuk melakukannya, sebaiknya gunakan titik akhir pembaca Aurora. Anda dapat menggunakan driver seperti Driver AWS JDBC. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Amazon Aurora.

> **Note**
>
> Basis data global Aurora saat ini tidak mendukung Aurora Auto Scaling untuk klaster DB sekunder.

## Kebijakan Aurora Auto Scaling

Aurora Auto Scaling menggunakan kebijakan penskalaan untuk menyesuaikan jumlah Replika Aurora dalam klaster DB Aurora. Aurora Auto Scaling memiliki komponen berikut:

- Peran terkait layanan
- Metrik target
- Kapasitas minimum dan maksimum
- Periode pendinginan

Topik

- Peran terkait layanan

- Metrik target
- Kapasitas minimum dan maksimum
- Periode pendinginan
- Mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam
- Menambahkan, mengedit, atau menghapus kebijakan penskalaan otomatis

Peran terkait layanan

Aurora Auto Scaling menggunakan peran yang berkaitan dengan layanan `AWSServiceRoleForApplicationAutoScaling_RDSCluster`. Untuk informasi selengkapnya, lihat Peran yang ditautkan dengan layanan untuk Application Auto Scaling dalam Panduan Pengguna Application Auto Scaling.

Metrik target

Dalam jenis kebijakan ini, metrik yang sudah ditentukan sebelumnya atau khusus dan nilai target untuk metrik tersebut ditentukan dalam konfigurasi kebijakan penskalaan pelacakan target. Auto Scaling Aurora membuat dan CloudWatch mengelola alarm yang memicu kebijakan penskalaan dan menghitung penyesuaian penskalaan berdasarkan metrik dan nilai target. Kebijakan penskalaan menambah atau menghapus Replika Aurora yang diperlukan untuk menjaga metrik pada, atau mendekati, nilai target yang ditentukan. Selain menjaga metrik tetap dekat dengan nilai target, kebijakan penskalaan pelacakan target juga disesuaikan menurut fluktuasi metrik karena pola beban kerja yang berubah. Kebijakan tersebut juga meminimalkan fluktuasi cepat dalam jumlah Replika Aurora yang tersedia untuk klaster DB Anda.

Misalnya, ambil kebijakan penskalaan yang menggunakan metrik penggunaan CPU rata-rata standar. Kebijakan tersebut dapat menjaga pemanfaatan CPU pada, atau mendekati, persentase penggunaan tertentu, seperti 40 persen.

> **Note**
>
> Untuk setiap klaster DB Aurora, Anda hanya dapat membuat satu kebijakan Penskalaan Otomatis untuk setiap metrik target.

Saat Anda mengonfigurasi Auto Scaling Aurora, nilai metrik target dihitung sebagai rata-rata semua instance pembaca di cluster. Perhitungan ini dilakukan sebagai berikut:

- Termasuk semua instance pembaca di klaster Aurora, terlepas dari apakah mereka dikelola oleh Auto Scaling atau ditambahkan secara manual.
- Termasuk contoh yang terkait dengan titik akhir kustom. Titik akhir khusus tidak memengaruhi perhitungan metrik target.
- Tidak termasuk instance penulis cluster.

Metrik berasal dari CloudWatch menggunakan dimensi berikut:

- `DBClusterIdentifier`
- `Role=READER`

Misalnya, pertimbangkan cluster Aurora MySQL dengan pengaturan berikut:

- Instans manual (tidak dikontrol oleh Auto Scaling):
  - Penulis dengan pemanfaatan CPU 50%
  - Pembaca 1 (titik akhir khusus:`custom-reader-1`) dengan pemanfaatan CPU 90%
  - Pembaca 2 (titik akhir khusus:`custom-reader-2`) dengan pemanfaatan CPU 90%
- Contoh Auto Scaling:
  - Reader 3 (ditambahkan menggunakan Auto Scaling) dengan pemanfaatan CPU 10%

Dalam skenario ini, metrik target untuk kebijakan Auto Scaling dihitung sebagai berikut:

```
Target metric = (CPU utilization of reader 1 + reader 2 + reader 3) / total number of
 readers

Target metric = (90 + 90 + 10) / 3 = 63.33%
```

Kebijakan Auto Scaling menggunakan nilai ini untuk mengevaluasi apakah akan menskalakan atau mengurangi skala berdasarkan ambang batas yang ditentukan.

Pertimbangkan hal berikut:

- Meskipun titik akhir khusus menentukan bagaimana lalu lintas diarahkan ke pembaca tertentu, mereka tidak mengecualikan pembaca dari perhitungan metrik.
- Instans manual selalu disertakan dalam perhitungan metrik target.

- Untuk menghindari perilaku penskalaan yang tidak terduga, pastikan konfigurasi Auto Scaling memperhitungkan semua instance pembaca di klaster.
- Jika klaster tidak memiliki pembaca, metrik tidak dihitung, dan alarm kebijakan Auto Scaling tetap tidak aktif. Agar kebijakan Auto Scaling berfungsi secara efektif, setidaknya satu pembaca harus hadir setiap saat.

### Kapasitas minimum dan maksimum

Anda dapat menentukan jumlah maksimum Replika Aurora yang akan dikelola oleh Application Auto Scaling. Nilai ini harus diatur ke 0–15, dan harus sama dengan atau lebih besar dari nilai yang ditentukan untuk jumlah minimum Replika Aurora.

Anda dapat menentukan jumlah minimum Replika Aurora yang akan dikelola oleh Application Auto Scaling. Nilai ini harus diatur ke 0–15, dan harus sama dengan atau kurang dari nilai yang ditentukan untuk jumlah maksimum Replika Aurora.

Harus ada setidaknya satu instans DB pembaca agar Auto Scaling Aurora berfungsi. Jika cluster DB tidak memiliki instance pembaca, dan Anda menyetel kapasitas minimum ke 0, maka Auto Scaling Aurora tidak akan berfungsi.

> **Note**
>
> Kapasitas minimum dan maksimum ditetapkan untuk klaster DB Aurora. Nilai yang ditentukan berlaku untuk semua kebijakan yang terkait dengan klaster DB Aurora.

### Periode pendinginan

Anda dapat menyesuaikan daya respons kebijakan penskalaan pelacakan target dengan menambahkan periode pendinginan yang memengaruhi penskalaan klaster DB Aurora Anda ke dalam dan ke luar. Periode pendinginan memblokmintaan penskalaan ke dalam atau ke luar berikutnya hingga periode ini berakhir. Pemblokiran ini memperlambat penghapusan Replika Aurora dalam klaster DB Aurora Anda untuk permintaan penskalaan ke dalam, dan pembuatan Replika Aurora untuk permintaan penskalaan ke luar.

Anda dapat menentukan periode pendinginan berikut:

- Aktivitas penskalaan ke dalam mengurangi jumlah Replika Aurora dalam klaster DB Aurora Anda. Periode pendinginan penskalaan ke dalam menentukan jumlah waktu, dalam detik, setelah

aktivitas penskalaan ke dalam selesai sebelum aktivitas penskalaan ke dalam lainnya dapat dimulai.

- Aktivitas penskalaan ke luar menambah jumlah Replika Aurora dalam klaster DB Aurora Anda. Periode pendinginan penskalaan ke luar menentukan jumlah waktu, dalam detik, setelah aktivitas penskalaan ke luar selesai sebelum aktivitas penskalaan ke luar lainnya dapat dimulai.

> **Note**
>
> Periode pendinginan penskalaan ke luar diabaikan jika permintaan penskalaan ke luar berikutnya ditujukan untuk jumlah Replika Aurora yang lebih besar daripada permintaan pertama.

Jika Anda tidak mengatur periode pendinginan penskalaan masuk atau ke luar, nilai default untuk masing-masing adalah 300 detik.

Mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam

Anda dapat mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam untuk sebuah kebijakan. Mengaktifkan aktivitas penskalaan ke dalam memungkinkan kebijakan penskalaan untuk menghapus Replika Aurora. Saat aktivitas penskalaan ke dalam diaktifkan, periode pendinginan penskalaan ke dalam di kebijakan penskalaan berlaku untuk aktivitas penskalaan ke dalam. Menonaktifkan aktivitas penskalaan ke dalam akan mencegah kebijakan penskalaan menghapus Replika Aurora.

> **Note**
>
> Aktivitas penskalaan ke luar selalu diaktifkan sehingga kebijakan penskalaan dapat membuat Replika Aurora sesuai kebutuhan.

Menambahkan, mengedit, atau menghapus kebijakan penskalaan otomatis

Anda dapat menambahkan, mengedit, atau menghapus kebijakan penskalaan otomatis menggunakan AWS Management Console, AWS CLI, atau Application Auto Scaling API. Untuk informasi selengkapnya tentang menambahkan, mengedit, atau menghapus kebijakan penskalaan otomatis, lihat bagian berikut.

- Menambahkan kebijakan penskalaan otomatis ke klaster DB Amazon Aurora

- Mengedit kebijakan penskalaan otomatis untuk klaster DB Amazon Aurora
- Menghapus kebijakan penskalaan otomatis dari klaster DB Amazon Aurora Anda

### Instans DB IDs dan penandaan

Ketika replika ditambahkan oleh Aurora Auto Scaling, ID instans DB-nya diberi awalan `application-autoscaling-`, misalnya, `application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123`.

Tag berikut secara otomatis ditambahkan ke instans DB. Anda dapat melihatnya di tab Tanda pada halaman detail instans DB.

| Tag | Nilai |
| --- | --- |
| application-autoscaling:resourceId | cluster:mynewcluster-cluster |

Untuk informasi selengkapnya tentang tag sumber daya Amazon RDS, lihat Menandai sumber daya Amazon Aurora dan Amazon RDS.

### Aurora Auto Scaling dan Wawasan Performa

Anda dapat menggunakan Wawasan Performa untuk memantau replika yang telah ditambahkan oleh Aurora Auto Scaling, sama seperti instans DB pembaca Aurora lainnya.

Untuk informasi selengkapnya tentang penggunaan Wawasan Performa untuk memantau klaster DB Aurora, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

## Menambahkan kebijakan penskalaan otomatis ke klaster DB Amazon Aurora

Anda dapat menambahkan kebijakan penskalaan menggunakan AWS Management Console, Application Auto Scaling AWS CLI, atau Application Auto API Scaling.

> **Note**
>
> Untuk contoh yang menambahkan kebijakan penskalaan menggunakan AWS CloudFormation, lihat Mendeklarasikan kebijakan penskalaan untuk klaster DB Aurora di Panduan Pengguna.AWS CloudFormation

Konsol

Anda dapat menambahkan kebijakan penskalaan ke cluster Aurora DB dengan menggunakan. AWS Management Console

Untuk menambahkan kebijakan penskalaan otomatis ke klaster DB Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB Aurora yang ingin Anda tambahi kebijakan.
4. Pilih tab Log & peristiwa.
5. Pada bagian Kebijakan auto scaling, pilih Tambahkan.

   Kotak dialog Tambahkan kebijakan Auto Scaling muncul.
6. Untuk Nama Kebijakan, ketik nama kebijakan.
7. Untuk metrik target, pilih salah satu dari berikut:

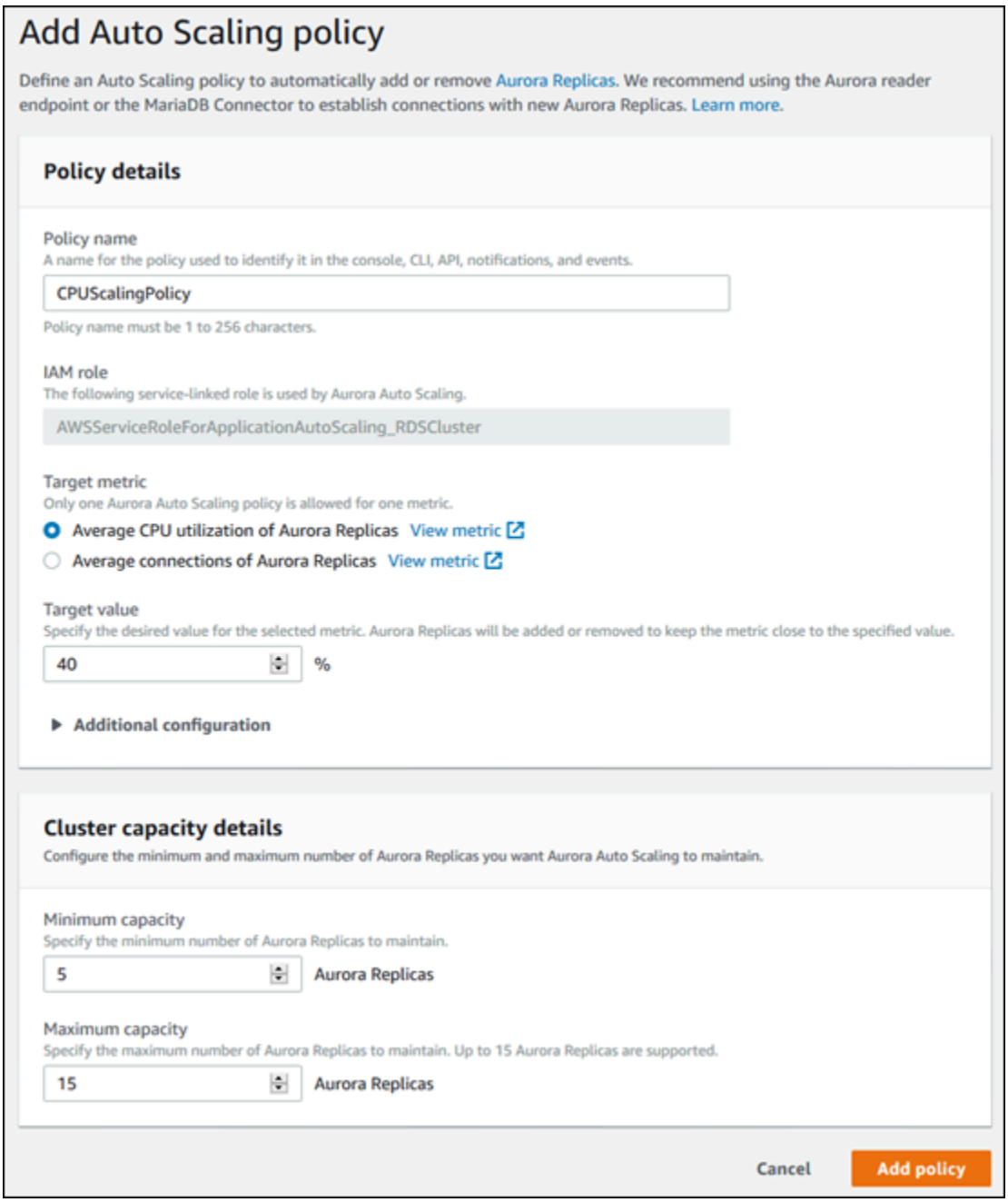
   - Rata-rata CPU pemanfaatan Replika Aurora untuk membuat kebijakan berdasarkan pemanfaatan rata-rata. CPU
   - Koneksi rata-rata dari Aurora Replicas untuk membuat kebijakan berdasarkan jumlah koneksi rata-rata ke Replika Aurora.
8. Untuk nilai target, ketik salah satu dari berikut ini:

   - Jika Anda memilih CPUpenggunaan Rata-rata Replika Aurora pada langkah sebelumnya, ketikkan persentase CPU pemanfaatan yang ingin Anda pertahankan pada Replika Aurora.
   - Jika Anda memilih Koneksi rata-rata dari Aurora Replicas pada langkah sebelumnya, ketik jumlah koneksi yang ingin Anda pertahankan.

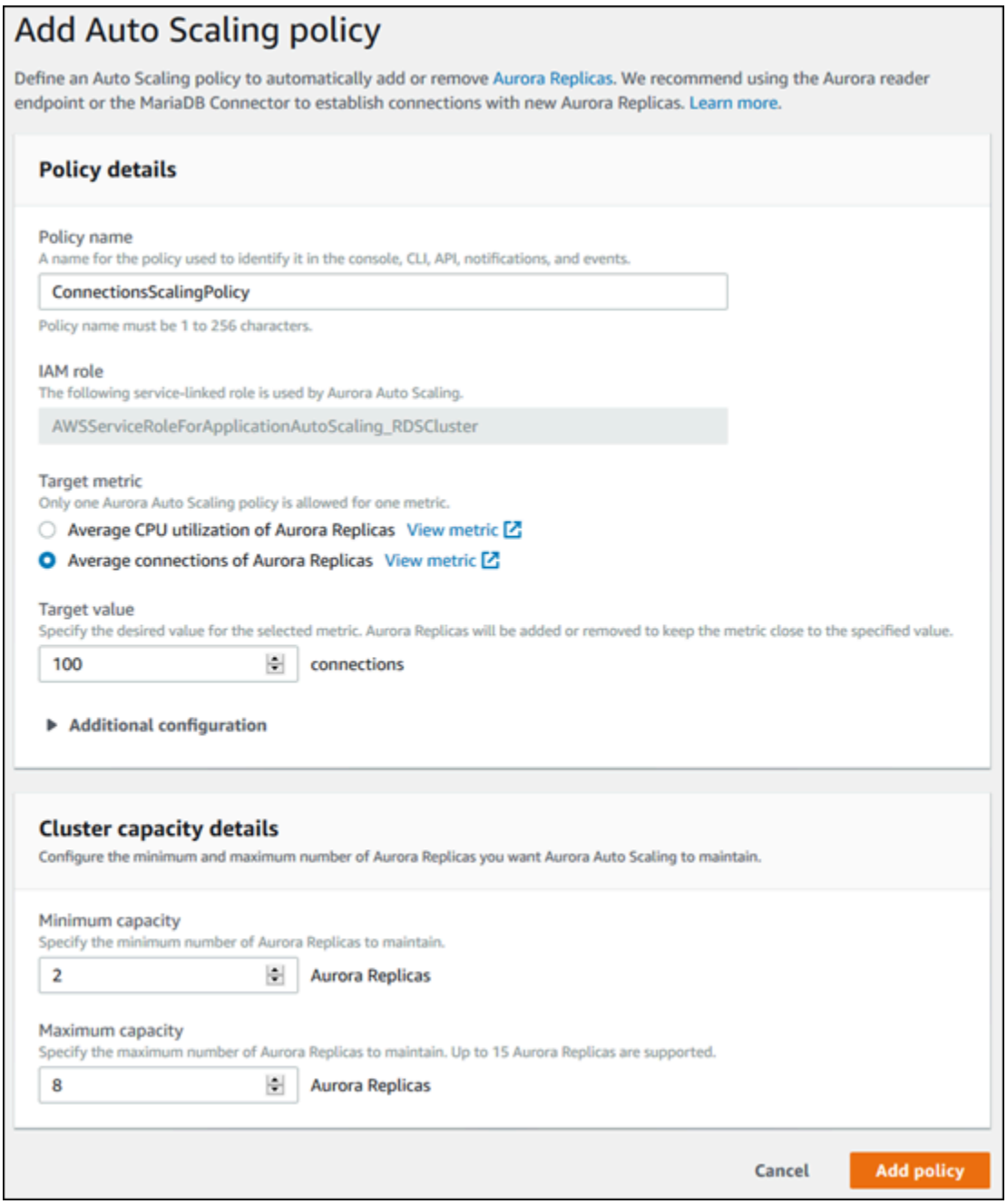
Replika Aurora ditambahkan atau dihapus untuk menjaga metrik tetap mendekati nilai yang ditentukan.

9. (Opsional) Perluas Konfigurasi Tambahan untuk membuat periode pendinginan penskalaan ke dalam atau ke luar.
10. Untuk Kapasitas minimum, ketik jumlah minimum Replika Aurora yang diwajibkan oleh kebijakan Aurora Auto Scaling untuk dipertahankan.
11. Untuk Kapasitas maksimum, ketik jumlah maksimum Replika Aurora yang diwajibkan oleh kebijakan Aurora Auto Scaling untuk dipertahankan.
12. Pilih Tambahkan kebijakan.

Kotak dialog berikut membuat kebijakan Auto Scaling berdasarkan CPU pemanfaatan rata-rata 40 persen. Kebijakan tersebut menentukan minimum 5 Replika Aurora dan maksimum 15 Replika Aurora.




Kotak dialog berikut membuat kebijakan penskalaan otomatis berdasarkan penggunaan koneksi rata-rata sebesar 100. Kebijakan tersebut menentukan minimum dua Replika Aurora dan maksimum delapan Replika Aurora.




AWS CLI atau Application Auto Scaling API

Anda dapat menerapkan kebijakan penskalaan berdasarkan metrik standar atau kustom. Untuk melakukannya, Anda dapat menggunakan AWS CLI atau Application Auto API Scaling. Langkah pertama adalah untuk mendaftarkan klaster DB Aurora Anda dengan Application Auto Scaling.

### Mendaftarkan klaster DB Aurora

Sebelum Anda dapat menggunakan Aurora Auto Scaling dengan klaster DB Aurora, Anda perlu mendaftarkan klaster DB Aurora dengan Application Auto Scaling. Anda melakukannya untuk mendefinisikan dimensi dan batasan penskalaan yang akan diterapkan pada klaster tersebut. Application Auto Scaling menskalakan klaster DB Aurora secara dinamis bersama dengan dimensi `rds:cluster:ReadReplicaCount` yang dapat diskalakan, yang merepresentasikan jumlah Replika Aurora.

Untuk mendaftarkan cluster Aurora DB Anda, Anda dapat menggunakan Application Auto Scaling AWS CLI atau Application Auto Scaling. API

#### AWS CLI

Untuk mendaftarkan cluster Aurora DB Anda, gunakan `register-scalable-target` AWS CLI perintah dengan parameter berikut:

- `--service-namespace` – Atur nilai ini ke `rds`.
- `--resource-id` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `--scalable-dimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `--min-capacity` – Jumlah minimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `--min-capacity`, `--max-capacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.
- `--max-capacity` – Jumlah maksimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `--min-capacity`, `--max-capacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.

#### Example

Dalam contoh berikut, Anda mendaftarkan klaster DB Aurora bernama `myscalablecluster`. Pendaftaran ini menunjukkan bahwa klaster DB harus diskalakan secara dinamis untuk memiliki satu hingga delapan Replika Aurora.

Untuk Linux, macOS, atau Unix:

```
aws application-autoscaling register-scalable-target \
```

```
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
    --max-capacity 8 \
```

Untuk Windows:

```
aws application-autoscaling register-scalable-target ^
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
    --max-capacity 8 ^
```

Application Auto Scaling API

Untuk mendaftarkan cluster Aurora DB Anda dengan Application Auto Scaling, gunakan operasi Application Auto `RegisterScalableTarget`Scaling dengan API parameter berikut:

- `ServiceNamespace` – Atur nilai ini ke `rds`.
- `ResourceID` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `ScalableDimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `MinCapacity` – Jumlah minimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `MinCapacity`, `MaxCapacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.
- `MaxCapacity` – Jumlah maksimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `MinCapacity`, `MaxCapacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.

Example

Dalam contoh berikut, Anda mendaftarkan cluster Aurora DB bernama `myscalablecluster` dengan Application Auto Scaling. API Pendaftaran ini menunjukkan bahwa klaster DB harus diskalakan secara dinamis untuk memiliki satu hingga delapan Replika Aurora.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Menetapkan kebijakan penskalaan untuk klaster DB Aurora

Konfigurasi kebijakan penskalaan pelacakan target diwakili oleh JSON blok tempat metrik dan nilai target ditentukan. Anda dapat menyimpan konfigurasi kebijakan penskalaan sebagai JSON blok dalam file teks. Anda menggunakan file teks tersebut saat menjalankan AWS CLI atau Application Auto API Scaling. Untuk informasi selengkapnya tentang sintaks konfigurasi kebijakan, lihat `TargetTrackingScalingPolicyConfiguration`di Referensi APIPenskalaan Otomatis Aplikasi.

Opsi berikut tersedia untuk menetapkan konfigurasi kebijakan penskalaan pelacakan target.

Topik

- Menggunakan metrik standar
- Menggunakan metrik khusus
- Menggunakan periode pendinginan
- Menonaktifkan aktivitas penskalaan ke dalam

## Menggunakan metrik standar

Dengan menggunakan metrik standar, Anda dapat dengan cepat menentukan kebijakan penskalaan pelacakan target untuk klaster DB Aurora yang berfungsi baik, dengan pelacakan target dan penskalaan dinamis dalam Aurora Auto Scaling.

Saat ini, Aurora mendukung metrik standar berikut dalam Aurora Auto Scaling:

- RDSReaderAverageCPUUtilization— Nilai rata-rata `CPUUtilization` metrik di CloudWatch seluruh Replika Aurora di cluster Aurora DB.
- RDSReaderAverageDatabaseConnections— Nilai rata-rata `DatabaseConnections` metrik di CloudWatch seluruh Replika Aurora di cluster Aurora DB.

Untuk informasi selengkapnya tentang metrik `CPUUtilization` dan `DatabaseConnections`, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

Untuk menggunakan metrik standar dalam kebijakan penskalaan, Anda membuat konfigurasi pelacakan target untuk kebijakan penskalaan Anda. Konfigurasi ini harus menyertakan `PredefinedMetricSpecification` untuk metrik yang sudah ditentukan sebelumnya dan `TargetValue` untuk nilai target metrik tersebut.

Example

Contoh berikut menjelaskan konfigurasi kebijakan umum untuk penskalaan pelacakan target untuk klaster DB Aurora. Dalam konfigurasi ini, metrik yang `RDSReaderAverageCPUUtilization` telah ditentukan digunakan untuk menyesuaikan cluster Aurora DB berdasarkan pemanfaatan CPU rata-rata 40 persen di semua Replika Aurora.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

Menggunakan metrik khusus

Dengan menggunakan metrik kustom, Anda dapat menentukan kebijakan penskalaan pelacakan target yang memenuhi persyaratan kustom Anda. Anda dapat menentukan metrik kustom berdasarkan metrik Aurora apa pun yang berubah secara berbanding lurus dengan penskalaan.

Tidak semua metrik Aurora berfungsi untuk pelacakan target. Metrik harus berupa metrik pemanfaatan yang valid dan menjelaskan tingkat kesibukan sebuah instans. Nilai metrik harus meningkat atau menurun secara berbanding lurus dengan jumlah Replika Aurora dalam klaster DB

Aurora. Peningkatan atau penurunan proporsional ini diperlukan untuk menggunakan data metrik untuk menskalakan ke luar atau ke dalam jumlah Replika Aurora secara proporsional.

Example

Contoh berikut menjelaskan konfigurasi pelacakan target untuk kebijakan penskalaan. Dalam konfigurasi ini, metrik khusus menyesuaikan cluster Aurora DB berdasarkan pemanfaatan CPU rata-rata 50 persen di semua Replika Aurora dalam cluster Aurora DB bernama. `my-db-cluster`

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

Menggunakan periode pendinginan

Anda dapat menentukan nilai, dalam detik, untuk `ScaleOutCooldown` guna menambahkan periode pendinginan untuk menskalakan ke luar klaster DB Aurora Anda. Demikian pula, Anda dapat menentukan nilai, dalam detik, untuk `ScaleInCooldown` guna menambahkan periode pendinginan untuk menskalakan ke dalam klaster DB Aurora Anda. Untuk informasi selengkapnya tentang `ScaleInCooldown` dan`ScaleOutCooldown`, lihat `TargetTrackingScalingPolicyConfiguration`di Referensi Application Auto Scaling. API

Example

Contoh berikut menjelaskan konfigurasi pelacakan target untuk kebijakan penskalaan. Dalam konfigurasi ini, metrik yang `RDSReaderAverageCPUUtilization` telah ditentukan digunakan untuk menyesuaikan cluster Aurora DB berdasarkan pemanfaatan CPU rata-rata 40 persen di semua Replika Aurora di cluster Aurora DB itu. Konfigurasi ini menyediakan periode pendinginan penskalaan ke dalam selama 10 menit dan periode pendinginan penskalaan ke luar selama 5 menit.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

Menonaktifkan aktivitas penskalaan ke dalam

Anda dapat mencegah konfigurasi kebijakan penskalaan pelacakan target agar tidak menskalakan klaster DB Aurora ke dalam dengan menonaktifkan aktivitas penskalaan ke dalam. Menonaktifkan aktivitas penskalaan ke dalam akan mencegah kebijakan penskalaan menghapus Replika Aurora, tetapi masih memungkinkan kebijakan penskalaan membuat Replika Aurora sesuai kebutuhan.

Anda dapat menentukan nilai Boolean untuk `DisableScaleIn` guna mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam untuk klaster DB Aurora Anda. Untuk informasi selengkapnya`DisableScaleIn`, lihat `TargetTrackingScalingPolicyConfiguration`di Referensi Application Auto Scaling. API

Example

Contoh berikut menjelaskan konfigurasi pelacakan target untuk kebijakan penskalaan. Dalam konfigurasi ini, metrik yang `RDSReaderAverageCPUUtilization` telah ditentukan menyesuaikan cluster Aurora DB berdasarkan CPU pemanfaatan rata-rata 40 persen di semua Replika Aurora di cluster Aurora DB itu. Konfigurasi ini menonaktifkan aktivitas skala masuk untuk kebijakan penskalaan.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
}
```

### Menerapkan kebijakan penskalaan untuk klaster DB Aurora

Setelah mendaftarkan klaster DB Aurora Anda dengan Application Auto Scaling dan menentukan kebijakan penskalaan, Anda perlu menerapkan kebijakan penskalaan ini ke klaster DB Aurora yang terdaftar. Untuk menerapkan kebijakan penskalaan ke klaster Aurora DB, Anda dapat menggunakan Application Auto Scaling atau AWS CLI Application Auto Scaling. API

#### AWS CLI

Untuk menerapkan kebijakan penskalaan ke cluster Aurora DB Anda, gunakan `put-scaling-policy` AWS CLI perintah dengan parameter berikut:

- `--policy-name` – Nama kebijakan penskalaan.
- `--policy-type` – Atur nilai ini ke `TargetTrackingScaling`.
- `--resource-id` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `--service-namespace` – Atur nilai ini ke `rds`.
- `--scalable-dimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `--target-tracking-scaling-policy-configuration` – Konfigurasi kebijakan penskalaan pelacakan target untuk digunakan pada klaster DB Aurora.

#### Example

Dalam contoh berikut, Anda menerapkan kebijakan penskalaan pelacakan target yang disebut `myscalablepolicy` ke klaster DB Aurora bernama `myscalablecluster` dengan Application Auto Scaling. Untuk melakukannya, Anda menggunakan konfigurasi kebijakan yang disimpan dalam file bernama `config.json`.

Untuk Linux, macOS, atau Unix:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Untuk Windows:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

Application Auto Scaling API

Untuk menerapkan kebijakan penskalaan ke cluster Aurora DB Anda dengan Application API Auto Scaling, gunakan operasi Application PutScalingPolicyAuto Scaling dengan API parameter berikut:

- `PolicyName` – Nama kebijakan penskalaan.
- `ServiceNamespace` – Atur nilai ini ke `rds`.
- `ResourceID` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `ScalableDimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `PolicyType` – Atur nilai ini ke `TargetTrackingScaling`.
- `TargetTrackingScalingPolicyConfiguration` – Konfigurasi kebijakan penskalaan pelacakan target untuk digunakan pada klaster DB Aurora.

Example

Dalam contoh berikut, Anda menerapkan kebijakan penskalaan pelacakan target yang disebut `myscalablepolicy` ke klaster DB Aurora bernama `myscalablecluster` dengan Application Auto Scaling. Anda menggunakan konfigurasi kebijakan berdasarkan pada metrik standar `RDSReaderAverageCPUUtilization`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
```

```
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

## Mengedit kebijakan penskalaan otomatis untuk klaster DB Amazon Aurora

Anda dapat mengedit kebijakan penskalaan menggunakan AWS Management Console, Application Auto Scaling AWS CLI, atau Application Auto API Scaling.

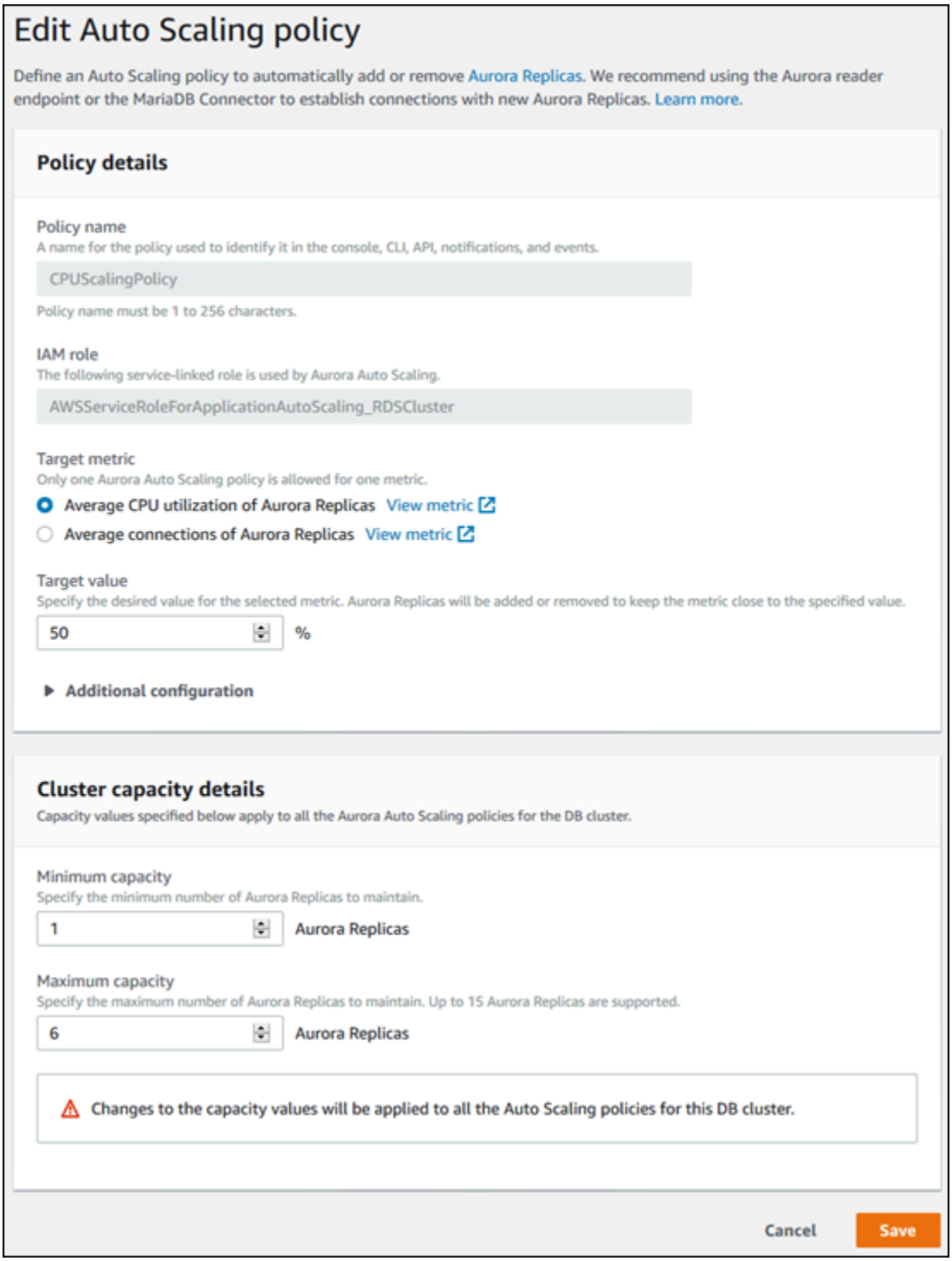
Konsol

Anda dapat mengedit kebijakan penskalaan dengan menggunakan. AWS Management Console

Untuk mengedit kebijakan penskalaan untuk klaster DB Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB Aurora yang kebijakan penskalaan otomatisnya ingin Anda edit.
4. Pilih tab Log & peristiwa.
5. Pada bagian Kebijakan auto scaling, pilih kebijakan penskalaan otomatis, lalu pilih Edit.
6. Buat perubahan pada kebijakan.
7. Pilih Simpan.

Berikut ini adalah contoh kotak dialog Edit kebijakan Auto Scaling.



AWS CLI atau Application Auto Scaling API

Anda dapat menggunakan Application Auto Scaling AWS CLI atau Application Auto Scaling API untuk mengedit kebijakan penskalaan dengan cara yang sama seperti Anda menerapkan kebijakan penskalaan:

- Saat menggunakan AWS CLI, tentukan nama kebijakan yang ingin Anda edit di `--policy-name` parameter. Tentukan nilai baru untuk parameter yang ingin Anda ubah.
- Saat menggunakan Application Auto ScalingAPI, tentukan nama kebijakan yang ingin Anda edit dalam parameter. `PolicyName` Tentukan nilai baru untuk parameter yang ingin Anda ubah.

Untuk informasi selengkapnya, lihat Menerapkan kebijakan penskalaan untuk klaster DB Aurora.

## Menghapus kebijakan penskalaan otomatis dari klaster DB Amazon Aurora Anda

Anda dapat menghapus kebijakan penskalaan menggunakan AWS Management Console, Application Auto Scaling AWS CLI, atau Application Auto API Scaling.

Konsol

Anda dapat menghapus kebijakan penskalaan menggunakan AWS Management Console.

Untuk menghapus kebijakan penskalaan untuk klaster DB Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB Aurora yang kebijakan penskalaan otomatisnya ingin Anda hapus.
4. Pilih tab Log & peristiwa.
5. Pada bagian Kebijakan auto scaling, pilih kebijakan penskalaan otomatis, lalu pilih Hapus.

AWS CLI

Untuk menghapus kebijakan penskalaan dari klaster Aurora DB Anda, gunakan `delete-scaling-policy` AWS CLI perintah dengan parameter berikut:

- `--policy-name` – Nama kebijakan penskalaan.

- `--resource-id` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `--service-namespace` – Atur nilai ini ke `rds`.
- `--scalable-dimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.

Example

Dalam contoh berikut, Anda menghapus kebijakan penskalaan pelacakan target bernama `myscalablepolicy` dari klaster DB Aurora bernama `myscalablecluster`.

Untuk Linux, macOS, atau Unix:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

Untuk Windows:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

Application Auto Scaling API

Untuk menghapus kebijakan penskalaan dari klaster Aurora DB Anda, gunakan `DeleteScalingPolicy`operasi Application API Auto Scaling dengan parameter berikut:

- `PolicyName` – Nama kebijakan penskalaan.
- `ServiceNamespace` – Atur nilai ini ke `rds`.

- `ResourceID` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `ScalableDimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.

Example

Dalam contoh berikut, Anda menghapus kebijakan penskalaan pelacakan target yang dinamai dari klaster Aurora DB `myscalablepolicy` yang diberi nama dengan Application `myscalablecluster` Auto Scaling. API

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

# Mengelola performa dan penskalaan untuk klaster DB Aurora

Anda dapat menggunakan opsi berikut untuk mengelola performa dan penskalaan untuk klaster DB dan instans DB Aurora:

Topik


- Penskalaan penyimpanan
- Penskalaan instans
- Penskalaan baca
- Mengelola koneksi
- Mengelola rencana eksekusi kueri


## Penskalaan penyimpanan

Penyimpanan Aurora secara otomatis diskalakan dengan data dalam volume klaster Anda. Seiring pertumbuhan data Anda, penyimpanan volume cluster Anda bertambah tergantung pada versi mesin DB. Untuk informasi tentang ukuran volume cluster Aurora maksimum untuk setiap versi mesin, lihat. Batas ukuran Amazon Aurora Untuk mempelajari data yang termasuk dalam volume klaster, lihat Penyimpanan Amazon Aurora. Untuk detail tentang ukuran maksimum untuk versi tertentu, lihat Batas ukuran Amazon Aurora.

Ukuran volume klaster Anda dievaluasi setiap jam untuk menentukan biaya penyimpanan Anda. Untuk informasi harga, lihat halaman Harga Aurora.

Meskipun volume klaster Aurora dapat menaikkan skala ukurannya menjadi banyak tebibyte, Anda hanya dikenai biaya untuk ruang yang Anda gunakan dalam volume. Mekanisme untuk menentukan ruang penyimpanan yang ditagih bergantung pada versi klaster Aurora Anda.

- Ketika data Aurora dihapus dari volume klaster, keseluruhan ruang yang ditagih berkurang sebesar jumlah yang sebanding. Perilaku pengubahan ukuran dinamis ini terjadi saat ruang tabel dasar dihapus atau disusun ulang agar membutuhkan lebih sedikit ruang. Dengan demikian, Anda dapat mengurangi biaya penyimpanan dengan menghapus tabel dan basis data yang tidak lagi Anda butuhkan. Pengubahan ukuran dinamis berlaku untuk versi Aurora tertentu. Berikut adalah versi Aurora yang memungkinkan volume klaster berubah ukurannya secara dinamis saat Anda menghapus data:

| Mesin basis data | Versi dengan pengubahan ukuran dinamis |
| --- | --- |
| Aurora MySQL | • Versi 3 (kompatibel dengan MySQL 8.0): semua versi didukung<br>• Versi 2 (kompatibel dengan MySQL 5.7): 2.11 dan lebih tinggi |
| Aurora PostgreSQL | Versi yang Didukung |
| Aurora Serverless v2 | Versi yang Didukung |
| Aurora Serverless v1 | Tidak tersedia |

- Dalam versi Aurora yang lebih rendah daripada yang ada di daftar sebelumnya, volume cluster dapat menggunakan kembali ruang yang dibebaskan saat Anda menghapus data, tetapi volume itu sendiri tidak pernah berkurang ukurannya.

Pengubahan ukuran dinamis berlaku untuk operasi yang secara fisik menghapus atau mengubah ukuran ruang tabel dalam volume klaster. Dengan demikian, hal ini berlaku untuk pernyataan SQL seperti `DROP TABLE`, `DROP DATABASE`, `TRUNCATE TABLE`, dan `ALTER TABLE ... DROP PARTITION`. Hal ini tidak berlaku untuk penghapusan baris menggunakan pernyataan `DELETE`. Jika Anda menghapus sejumlah besar baris dari tabel, Anda dapat menjalankan pernyataan `OPTIMIZE TABLE` Aurora MySQL atau menggunakan ekstensi `pg_repack` Aurora PostgreSQL setelahnya untuk menyusun ulang tabel dan mengubah ukuran volume klaster secara dinamis.

Untuk Aurora MySQL, pertimbangan berikut berlaku:

- Setelah Anda memutakhirkan cluster DB Anda ke versi mesin DB yang mendukung pengubahan ukuran dinamis, dan ketika fitur diaktifkan secara spesifik Wilayah AWS, ruang apa pun yang nantinya dibebaskan oleh pernyataan SQL tertentu, seperti`DROP TABLE`, dapat direklamasi kembali.

  Jika fitur dinonaktifkan secara eksplisit pada suatu fitur tertentu Wilayah AWS, ruang mungkin hanya dapat digunakan kembali—dan tidak dapat direklamasi—bahkan pada versi yang mendukung pengubahan ukuran dinamis.

  Fitur ini diaktifkan untuk versi mesin DB tertentu (1.23.0—1.23.4, 2.09.0—2.09.3, dan 2.10.0) antara November 2020 dan Maret 2022, dan diaktifkan secara default pada versi berikutnya.

- Sebuah tabel disimpan secara internal dalam satu atau lebih fragmen yang berdekatan dengan berbagai ukuran. Saat menjalankan `TRUNCATE TABLE` operasi, ruang yang sesuai dengan fragmen pertama dapat digunakan kembali dan tidak dapat direklamasi. Fragmen lainnya dapat direklamasi. Selama `DROP TABLE` operasi, ruang yang sesuai dengan seluruh tablespace dapat direklamasi kembali.
- `innodb_file_per_table`Parameter mempengaruhi bagaimana penyimpanan tabel diatur. Jika tabelnya adalah bagian dari ruang tabel sistem, menghapus tabel tidak akan mengurangi ukuran ruang tabel sistem. Jadi, pastikan untuk mengatur `innodb_file_per_table` ke 1 untuk klaster DB Aurora MySQL agar mengambil keuntungan penuh dari pengubahan ukuran dinamis.
- Di versi 2.11 dan yang lebih tinggi, ruang meja sementara InnoDB dijatuhkan dan dibuat ulang saat restart. Hal ini akan melepaskan ruang yang ditempati oleh ruang tabel sementara ke sistem, lalu volume klaster akan diubah ukurannya. Untuk memanfaatkan sepenuhnya fitur pengubahan ukuran dinamis, kami sarankan Anda meningkatkan cluster DB Anda ke versi 2.11 atau lebih tinggi.

> **Note**
>
> Fitur pengubahan ukuran dinamis tidak segera merebut kembali ruang saat tabel di ruang tabel dijatuhkan, tetapi secara bertahap dengan kecepatan sekitar 10 TB per hari. Spasi di tablespace sistem tidak direklamasi, karena tablespace sistem tidak pernah dihapus. Ruang kosong yang tidak diklaim ulang di ruang tabel akan digunakan kembali saat operasi membutuhkan ruang di ruang tabel tersebut. Fitur pengubahan ukuran dinamis dapat mengklaim kembali ruang penyimpanan hanya ketika klasternya dalam keadaan tersedia.

Anda dapat memeriksa berapa banyak ruang penyimpanan yang digunakan klaster dengan memantau metrik `VolumeBytesUsed` pada CloudWatch. Untuk informasi selengkapnya tentang penagihan penyimpanan, lihat Cara penyimpanan data Aurora ditagih.

- Di dalam AWS Management Console, Anda dapat melihat gambar ini dalam bagan dengan melihat `Monitoring` tab pada halaman detail untuk cluster.
- Dengan AWS CLI, Anda dapat menjalankan perintah yang mirip dengan contoh Linux berikut. Ganti contoh ini dengan nilai Anda sendiri untuk waktu mulai dan berakhir serta nama klaster.

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
```

```
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

Perintah tersebut menghasilkan output seperti yang berikut ini.

```
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

Contoh berikut menunjukkan bagaimana Anda dapat melacak penggunaan penyimpanan untuk cluster Aurora dari waktu ke waktu menggunakan AWS CLI perintah pada sistem Linux. Parameter `--start-time` dan `--end-time` menentukan interval waktu keseluruhan sebagai satu hari. Parameter `--period` meminta pengukuran pada interval satu jam. Tidak masuk akal untuk memilih nilai `--period` yang kecil karena metrik dikumpulkan dalam interval dan tidak terus-menerus. Selain itu, operasi penyimpanan Aurora terkadang berlanjut selama beberapa waktu di latar belakang setelah pernyataan SQL yang relevan selesai.

Contoh pertama menghasilkan output dalam format default JSON. Titik data dihasilkan dalam urutan arbitrer, tidak diurutkan berdasarkan stempel waktu. Anda dapat mengimpor data JSON ini ke alat bagan untuk melakukan pengurutan dan visualisasi.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
```

```
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

Contoh ini menghasilkan data yang sama seperti sebelumnya. Parameter `--output` merepresentasikan data dalam format teks biasa yang ringkas. Perintah `aws cloudwatch` mengirimkan output ke perintah `sort`. Parameter `-k` dari perintah `sort` mengurutkan output berdasarkan bidang ketiga, yang merupakan stempel waktu dalam format Waktu Universal Terkoordinasi (UTC).

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
```

```
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

Output yang diurutkan menunjukkan berapa banyak penyimpanan yang digunakan pada awal dan akhir periode pemantauan. Anda juga dapat menemukan titik selama periode tersebut saat Aurora mengalokasikan lebih banyak penyimpanan untuk klaster. Contoh berikut menggunakan perintah Linux untuk memformat ulang nilai `VolumeBytesUsed` awal dan akhir sebagai gigabyte (GB) dan gibibyte (GiB). Gigabyte merepresentasikan unit yang diukur dalam pangkat 10 dan umumnya digunakan dalam pembahasan penyimpanan untuk hard drive rotasional. Gibibyte merepresentasikan unit yang diukur dalam pangkat 2. Pengukuran dan batas penyimpanan Aurora biasanya dinyatakan dalam unit pangkat 2, seperti gibibyte dan tebibyte.

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

Metrik `VolumeBytesUsed` memberi tahu Anda berapa banyak penyimpanan dalam klaster tersebut yang menimbulkan biaya. Jadi, jika memungkinkan sebaiknya minimalkan angka ini. Namun, metrik ini tidak mencakup beberapa penyimpanan yang digunakan Aurora secara internal di klaster dan tidak dikenai biaya. Jika klaster Anda mendekati batas penyimpanan dan mungkin kehabisan ruang, sebaiknya pantau metrik `AuroraVolumeBytesLeftTotal` dan coba maksimalkan angka tersebut. Contoh berikut menjalankan perhitungan yang sama seperti yang sebelumnya, tetapi untuk `AuroraVolumeBytesLeftTotal`, bukan `VolumeBytesUsed`.

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
```

```
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

Untuk klaster yang menjalankan Aurora MySQL versi 2.09 atau yang lebih tinggi, atau Aurora PostgreSQL, ukuran kosong yang dilaporkan meningkat sebesar `VolumeBytesUsed` saat data ditambahkan dan berkurang saat data dihapus. Contoh berikut menunjukkan cara melakukannya. Laporan ini menunjukkan ukuran penyimpanan maksimum dan minimum untuk klaster dengan interval 15 menit saat tabel dengan data sementara dibuat dan dihapus. Laporan ini mencantumkan nilai maksimum sebelum nilai minimum. Jadi, untuk memahami bagaimana penggunaan penyimpanan berubah dalam interval 15 menit, tafsirkan angkanya dari kanan ke kiri.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

Contoh berikut menunjukkan bagaimana dengan cluster yang menjalankan Aurora PostgreSQL, ukuran bebas yang dilaporkan oleh mencerminkan batas ukuran 256-Tib. `AuroraVolumeBytesLeftTotal`

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
```

```
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
256 TiB remaining for this cluster
```

## Penskalaan instans

Anda dapat menskalakan klaster DB Aurora sesuai kebutuhan dengan memodifikasi kelas instans DB untuk setiap instans DB dalam klaster DB ini. Aurora mendukung beberapa kelas instans DB yang dioptimalkan untuk Aurora, bergantung pada kompatibilitas mesin basis data.

| Mesin basis data | Penskalaan instans |
| --- | --- |
| Amazon Aurora MySQL | Lihat Penskalaan instans DB Aurora MySQL |
| Amazon Aurora PostgreSQL | Lihat Penskalaan instans DB Aurora PostgreSQL |

## Penskalaan baca

Anda dapat melakukan penskalaan baca untuk klaster DB Aurora dengan membuat hingga 15 Replika Aurora dalam klaster DB. Setiap Replika Aurora menghasilkan data yang sama dari volume klaster dengan sedikit lag replika—biasanya kurang dari 100 milidetik setelah instans primer menulis pembaruan. Saat lalu lintas baca meningkat, Anda dapat membuat Replika Aurora tambahan dan terhubung langsung ke replika ini untuk mendistribusikan beban baca untuk klaster DB Anda. Replika Aurora tidak harus berasal dari kelas instans DB yang sama dengan instans primer.

Untuk informasi tentang menambahkan Replika Aurora ke klaster DB, lihat Menambahkan Replika Aurora ke klaster DB.

## Mengelola koneksi

Jumlah maksimum koneksi yang diizinkan untuk instans DB Aurora ditentukan oleh parameter `max_connections` dalam grup parameter tingkat instans untuk instans DB. Nilai default parameter tersebut bervariasi bergantung pada kelas instans DB yang digunakan untuk instans DB dan kompatibilitas mesin basis data.

| Mesin basis data | Nilai default max_connections |
|---|---|
| Amazon Aurora MySQL | Lihat Koneksi maksimum ke instans DB Aurora MySQL |
| Amazon Aurora PostgreSQL | Lihat Koneksi maksimum ke instans DB Aurora PostgreSQL |

> **Tip**
>
> Jika aplikasi Anda sering membuka dan menutup koneksi, atau membiarkan sejumlah besar koneksi yang lama aktif tetap terbuka, kami menyarankan agar Anda menggunakan Proksi Amazon RDS. Proksi RDS adalah proksi basis data yang sepenuhnya terkelola dengan ketersediaan tinggi yang menggunakan pooling koneksi untuk berbagi koneksi basis data dengan aman dan efisien. Untuk mempelajari tentang RDS, lihat Proksi Amazon RDS untuk Aurora.

## Mengelola rencana eksekusi kueri

Jika Anda menggunakan manajemen rencana kueri untuk Aurora PostgreSQL, Anda dapat mengontrol rencana mana yang dijalankan pengoptimisasi. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL.

# Mengkloning volume untuk klaster DB Amazon Aurora

Dengan menggunakan kloning Aurora, Anda dapat membuat klaster baru yang memiliki halaman data yang sama dengan aslinya, tetapi merupakan volume terpisah dan independen. Prosesnya dirancang agar cepat dan hemat biaya. Klaster baru dengan volume data terkaitnya dikenal sebagai Klon. Membuat klon lebih cepat dan lebih hemat ruang daripada menyalin data secara fisik menggunakan teknik lain, seperti memulihkan snapshot.

Topik


- Gambaran umum kloning Aurora
- Batasan kloning Aurora
- Cara kerja kloning Aurora
- Membuat klon Amazon Aurora
- VPCKloning silang dengan Amazon Aurora
- Kloning lintas akun dengan AWS RAM dan Amazon Aurora


## Gambaran umum kloning Aurora

Aurora menggunakan copy-on-write protokol untuk membuat klon. Mekanisme ini menggunakan ruang tambahan minimal untuk membuat klon awal. Ketika klon pertama kali dibuat, Aurora mempertahankan satu salinan data yang digunakan oleh klaster DB Aurora sumber dan klaster DB Aurora yang baru (klon). Penyimpanan tambahan dialokasikan hanya ketika perubahan dibuat pada data (pada volume penyimpanan Aurora) oleh klaster DB Aurora sumber atau klon klaster DB Aurora. Untuk mempelajari lebih lanjut tentang copy-on-write protokol, lihatCara kerja kloning Aurora.

Kloning Aurora sangat berguna untuk menyiapkan lingkungan pengujian dengan cepat menggunakan data produksi Anda, tanpa risiko kerusakan data. Anda dapat menggunakan klon untuk berbagai jenis aplikasi, seperti berikut:

- Bereksperimen dengan potensi perubahan (misalnya, perubahan skema dan perubahan grup parameter) untuk menilai semua dampak.
- Menjalankan operasi sarat beban kerja, seperti mengekspor data atau menjalankan kueri analitis.
- Membuat salinan dari klaster DB produksi Anda untuk pengembangan, pengujian, atau tujuan lainnya.

Anda dapat membuat lebih dari satu klon dari klaster DB Aurora yang sama. Anda juga dapat membuat beberapa klon dari klon lain.

Setelah membuat klon Aurora, Anda dapat mengonfigurasi instans DB Aurora secara berbeda dari klaster DB Aurora sumber. Misalnya, Anda mungkin tidak memerlukan klon untuk tujuan pengembangan guna memenuhi persyaratan ketersediaan tinggi yang sama dengan klaster DB Aurora produksi. Dalam hal ini, Anda dapat mengonfigurasi klon dengan satu instans DB Aurora dan bukan beberapa instans DB yang digunakan oleh klaster DB Aurora.

Saat Anda membuat klon menggunakan konfigurasi penyebaran yang berbeda dari sumbernya, klon dibuat menggunakan versi minor terbaru dari mesin Aurora DB sumber.

Saat Anda membuat klon dari cluster Aurora DB Anda, klon dibuat di AWS akun Anda—akun yang sama yang memiliki cluster Aurora DB sumber. Namun, Anda juga dapat berbagi Aurora Serverless v2 dan menyediakan cluster dan klon Aurora DB dengan akun lain. AWS Untuk informasi selengkapnya, lihat Kloning lintas akun dengan AWS RAM dan Amazon Aurora.

Setelah selesai menggunakan klon untuk pengujian, pengembangan, atau tujuan lainnya, Anda dapat menghapusnya.

## Batasan kloning Aurora

Kloning Aurora saat ini memiliki batasan berikut:

- Anda dapat membuat klon sebanyak yang Anda inginkan, hingga jumlah maksimum klaster DB yang diizinkan di Wilayah AWS.
- Anda dapat membuat hingga 15 klon dengan copy-on-write protokol. Setelah Anda memiliki 15 klon, klon berikutnya yang Anda buat adalah salinan lengkap. Protokol salinan lengkap bertindak seperti point-in-time pemulihan.
- Anda tidak dapat membuat klon di AWS Wilayah yang berbeda dari cluster Aurora DB sumber.
- Anda tidak dapat membuat klon dari klaster DB Aurora tanpa fitur kueri paralel ke klaster yang menggunakan kueri paralel. Untuk memasukkan data ke dalam klaster yang menggunakan kueri paralel, buat snapshot dari klaster asli dan pulihkan ke klaster yang menggunakan fitur opsi kueri pararel.
- Anda tidak dapat membuat klon dari klaster DB Aurora yang tidak memiliki instans DB. Anda hanya dapat mengkloning klaster DB Aurora yang memiliki setidaknya satu instans DB.
- Anda dapat membuat klon di cloud privat virtual (VPC) yang berbeda dari klaster DB Aurora. Jika Anda melakukannya, subnet VPCs harus dipetakan ke Availability Zones yang sama.

- Anda dapat membuat klon terprovisi Aurora dari klaster Aurora DB terprovisi.
- Cluster dengan Aurora Serverless v2 instance mengikuti aturan yang sama dengan cluster yang disediakan.
- Untuk Aurora Serverless v1:
  - Anda dapat membuat klon yang disediakan dari Aurora Serverless v1 klaster DB.
  - Anda dapat membuat Aurora Serverless v1 klon dari sebuah Aurora Serverless v1 atau cluster DB yang disediakan.
  - Anda tidak dapat membuat Aurora Serverless v1 kloning dari cluster Aurora DB yang tidak terenkripsi dan disediakan.
  - Kloning lintas akun saat ini tidak mendukung kloning Aurora Serverless v1 klaster DB. Untuk informasi selengkapnya, lihat Batasan kloning lintas akun.
  - Sebuah kloning Aurora Serverless v1 Cluster DB memiliki perilaku dan keterbatasan yang sama seperti apa pun Aurora Serverless v1 klaster DB. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.
  - Aurora Serverless v1 selalu terenkripsi. Saat Anda mengkloning sebuah Aurora Serverless v1 Cluster DB menjadi cluster Aurora DB yang disediakan, cluster Aurora DB yang disediakan dienkripsi. Anda dapat memilih kunci enkripsi, tetapi tidak dapat menonaktifkan enkripsi. Untuk mengkloning dari cluster Aurora DB yang disediakan ke Aurora Serverless v1, Anda harus mulai dengan cluster Aurora DB yang disediakan terenkripsi.

## Cara kerja kloning Aurora

Kloning Aurora beroperasi pada lapisan penyimpanan klaster DB Aurora. Ini menggunakan copy-on-writeprotokol yang cepat dan hemat ruang dalam hal media tahan lama yang mendasarinya yang mendukung volume penyimpanan Aurora. Anda dapat mempelajari selengkapnya tentang volume klaster Aurora dalam Gambaran umum penyimpanan Amazon Aurora.

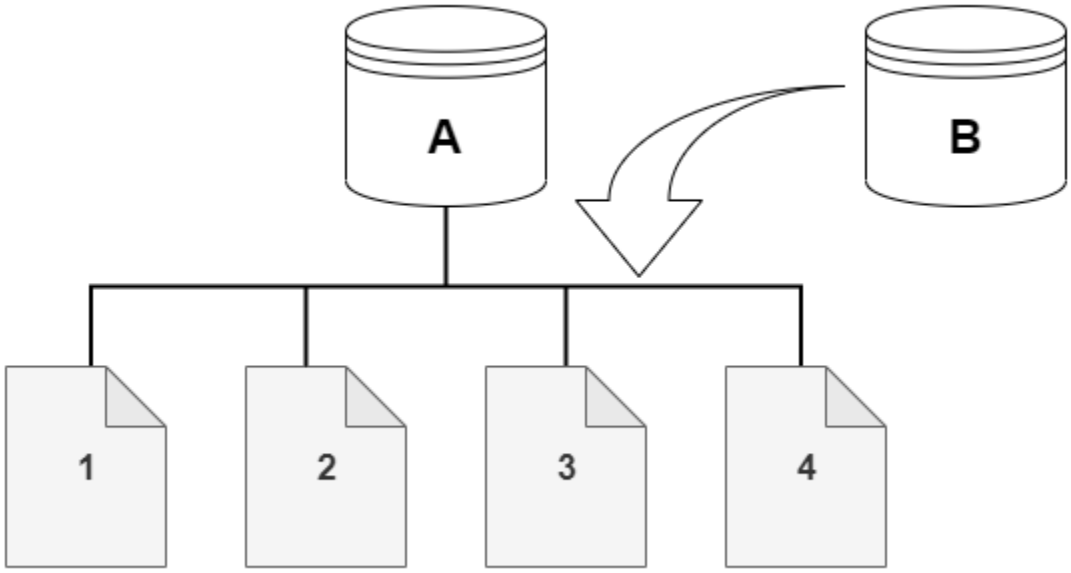
Topik

- Memahami copy-on-write protokol
- Menghapus volume klaster sumber

### Memahami copy-on-write protokol

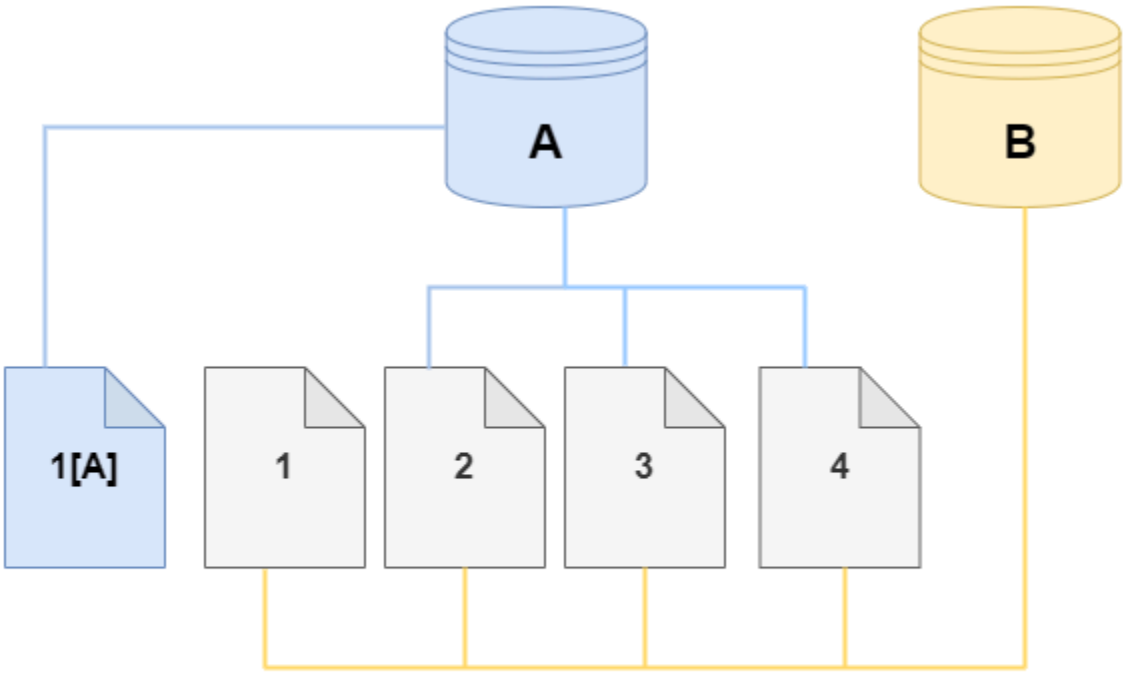
Sebuah klaster DB Aurora menyimpan data dalam halaman di volume penyimpanan Aurora yang mendasarinya.

Misalnya, dalam diagram berikut, Anda dapat menemukan klaster DB Aurora (A) yang memiliki empat halaman data, 1, 2, 3, dan 4. Bayangkan bahwa sebuah klon, B, dibuat dari klaster DB Aurora. Saat klon dibuat, tidak ada data yang disalin. Sebagai gantinya, klon tersebut menunjuk ke kumpulan halaman yang sama sebagai sumber klaster DB Aurora.

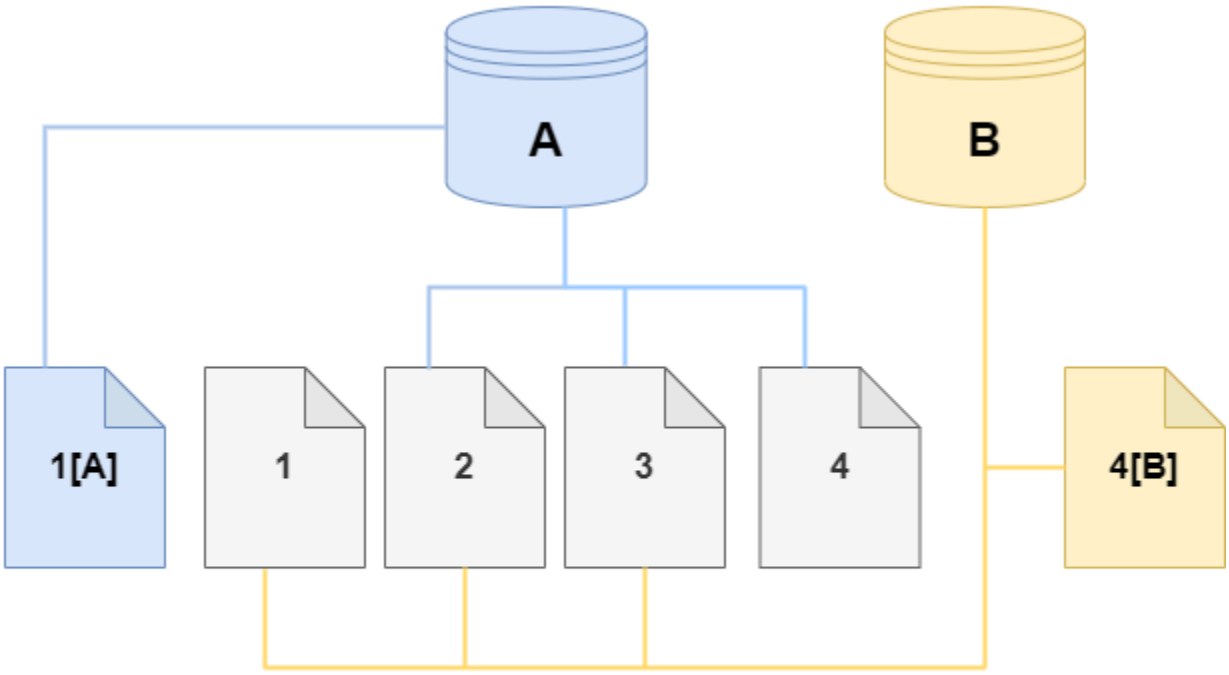



Saat klon dibuat, tidak ada penyimpanan tambahan yang biasanya diperlukan. copy-on-writeProtokol menggunakan segmen yang sama pada media penyimpanan fisik sebagai segmen sumber. Penyimpanan tambahan hanya diperlukan jika kapasitas segmen sumber tidak cukup untuk seluruh segmen klon. Jika demikian, segmen sumber disalin ke perangkat fisik lain.

Dalam diagram berikut, Anda dapat menemukan contoh copy-on-write protokol yang sedang beraksi menggunakan cluster A yang sama dan klonnya, B, seperti yang ditunjukkan sebelumnya. Katakanlah bahwa Anda membuat perubahan pada klaster DB Aurora (A) yang menghasilkan perubahan pada data yang disimpan di halaman 1. Alih-alih menulis ke halaman 1 asli, Aurora membuat halaman baru 1[A]. Volume klaster DB Aurora untuk klaster (A) sekarang menunjuk ke halaman 1[A], 2, 3, dan 4, sedangkan klon (B) masih merujuk ke halaman asli.




Pada klon, perubahan dibuat pada halaman 4 di volume penyimpanan. Alih-alih menulis ke halaman 4 asli, Aurora membuat halaman baru 4[B]. Klon sekarang menunjuk ke halaman 1, 2, 3, dan halaman 4[B], sementara klaster (A) terus menunjuk ke 1[A], 2, 3, dan 4.




Saat lebih banyak perubahan terjadi seiring waktu di volume klaster DB Aurora sumber dan klon, diperlukan lebih banyak penyimpanan untuk menangkap dan menyimpan perubahan tersebut.

## Menghapus volume klaster sumber

Awalnya, volume klon berbagi halaman data yang sama dengan volume asli dari mana klon dibuat. Selama volume asli ada, volume klon hanya dianggap sebagai pemilik halaman yang dibuat atau dimodifikasi oleh klon. Dengan demikian, `VolumeBytesUsed` metrik untuk volume klon mulai kecil dan hanya tumbuh saat data menyimpang antara cluster asli dan klon. Untuk halaman yang identik antara volume sumber dan klon, biaya penyimpanan hanya berlaku untuk cluster asli. Untuk informasi lebih lanjut tentang `VolumeBytesUsed` metrik, lihatMetrik tingkat klaster untuk Amazon Aurora.

Saat Anda menghapus volume cluster sumber yang memiliki satu atau lebih klon yang terkait dengannya, data dalam volume klon klon tidak berubah. Aurora menyimpan halaman yang sebelumnya dimiliki oleh volume cluster sumber. Aurora mendistribusikan kembali tagihan penyimpanan untuk halaman yang dimiliki oleh cluster yang dihapus. Misalnya, misalkan cluster asli memiliki dua klon dan kemudian cluster asli dihapus. Setengah dari halaman data yang dimiliki oleh

cluster asli sekarang akan dimiliki oleh satu klon. Setengah halaman lainnya akan dimiliki oleh klon lainnya.

Jika Anda menghapus cluster asli, maka saat Anda membuat atau menghapus lebih banyak klon, Aurora terus mendistribusikan kembali kepemilikan halaman data di antara semua klon yang berbagi halaman yang sama. Dengan demikian, Anda mungkin menemukan bahwa nilai `VolumeBytesUsed` metrik berubah untuk volume cluster klon. Nilai metrik dapat menurun karena lebih banyak klon dibuat dan kepemilikan halaman tersebar di lebih banyak cluster. Nilai metrik juga dapat meningkat karena klon dihapus dan kepemilikan halaman ditetapkan ke sejumlah kecil cluster. Untuk informasi tentang bagaimana operasi penulisan memengaruhi halaman data pada volume klon, lihatMemahami copy-on-write protokol.

Ketika cluster asli dan klon dimiliki oleh AWS akun yang sama, semua biaya penyimpanan untuk cluster tersebut berlaku untuk akun yang sama AWS . Jika beberapa cluster adalah klon lintas akun, menghapus cluster asli dapat mengakibatkan biaya penyimpanan tambahan ke AWS akun yang memiliki klon lintas akun.

Misalnya, misalkan volume cluster memiliki 1000 halaman data yang digunakan sebelum Anda membuat klon apa pun. Saat Anda mengkloning cluster itu, awalnya volume klon memiliki nol halaman yang digunakan. Jika klon membuat modifikasi ke 100 halaman data, hanya 100 halaman yang disimpan pada volume klon dan ditandai sebagai yang digunakan. 900 halaman lain yang tidak berubah dari volume induk dibagikan oleh kedua cluster. Dalam hal ini, cluster induk memiliki biaya penyimpanan untuk 1000 halaman dan volume klon untuk 100 halaman.

Jika Anda menghapus volume sumber, biaya penyimpanan untuk klon termasuk 100 halaman yang diubah, ditambah 900 halaman bersama dari volume asli, dengan total 1000 halaman.

## Membuat klon Amazon Aurora

Anda dapat membuat klon di AWS akun yang sama dengan cluster Aurora DB sumber. Untuk melakukannya, Anda dapat menggunakan AWS Management Console atau AWS CLI dan prosedur berikut.

Untuk mengizinkan AWS akun lain membuat klon atau berbagi klon dengan AWS akun lain, gunakan prosedur di. Kloning lintas akun dengan AWS RAM dan Amazon Aurora

Konsol

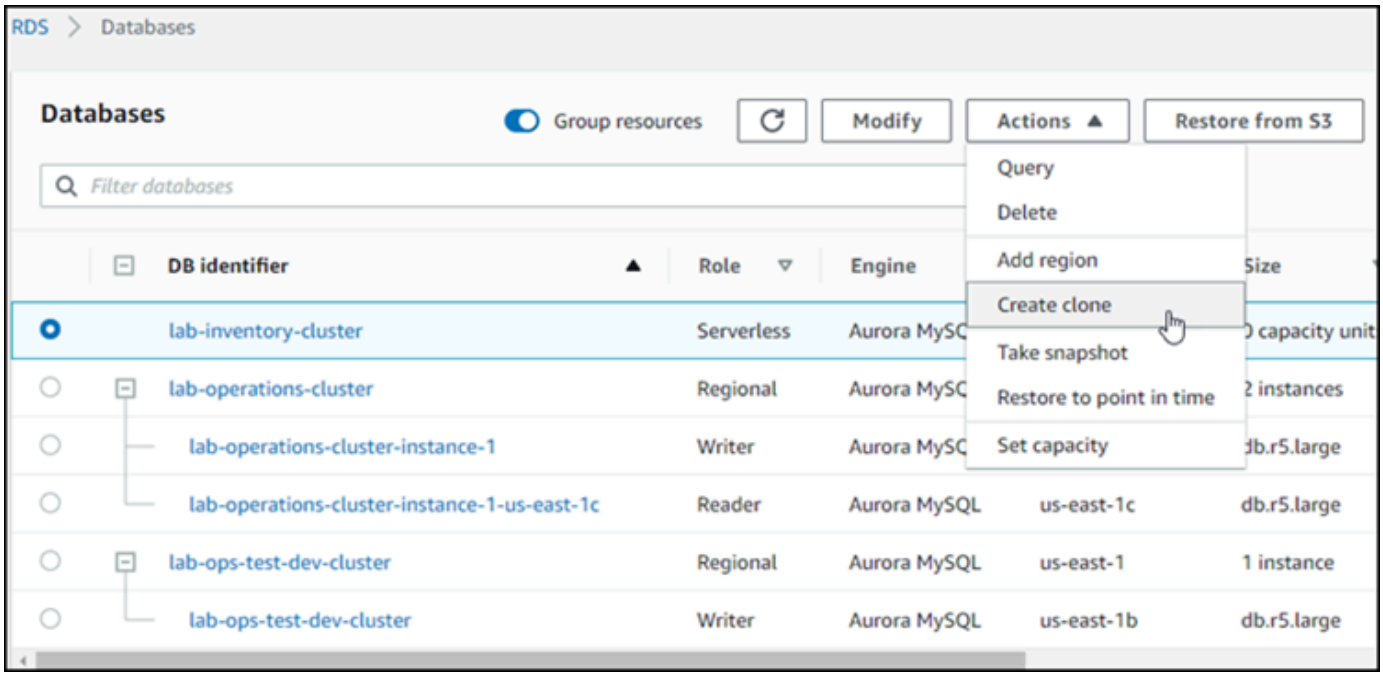
Prosedur berikut menjelaskan cara mengkloning klaster DB Aurora menggunakan AWS Management Console.

Membuat klon menggunakan AWS Management Console hasil dalam cluster Aurora DB dengan satu instance Aurora DB.

Instruksi ini berlaku untuk cluster DB yang dimiliki oleh AWS akun yang sama yang membuat klon. Jika cluster DB dimiliki oleh AWS akun yang berbeda, lihat Kloning lintas akun dengan AWS RAM dan Amazon Aurora saja.

Untuk membuat tiruan dari cluster DB yang dimiliki oleh AWS akun Anda menggunakan AWS Management Console

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB Aurora Anda dari daftar, dan untuk Tindakan pilih Buat klon.




   Halaman Buat klon terbuka, yang memungkinkan Anda mengonfigurasi Pengaturan, Konektivitas, dan opsi lain untuk klon klaster DB Aurora.
4. Untuk Pengidentifikasi instans DB, masukkan nama yang ingin Anda berikan ke klaster DB Aurora Anda.
5. Untuk Aurora Serverless v1 Cluster DB, pilih Provisioned atau Serverless untuk tipe Kapasitas.

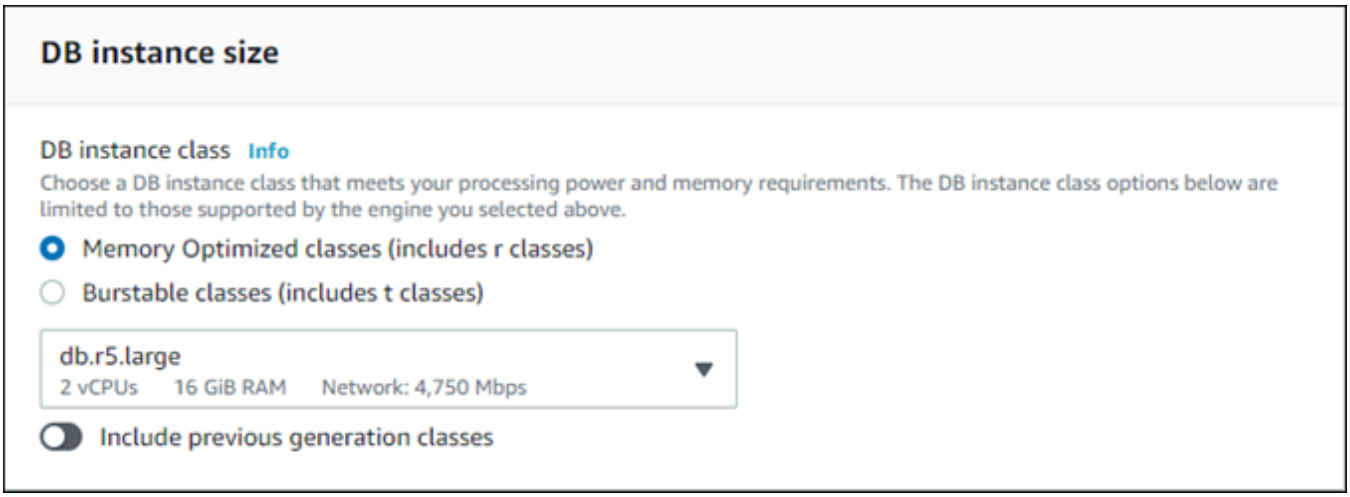
   Anda dapat memilih Tanpa Server hanya jika sumber cluster Aurora DB adalah Aurora Serverless v1 Cluster DB atau merupakan cluster Aurora DB yang disediakan yang dienkripsi.

6. Untuk Aurora Serverless v2 atau cluster DB yang disediakan, pilih salah satu Aurora I/O-Optimized atau Aurora Standarduntuk konfigurasi penyimpanan Cluster.

   Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

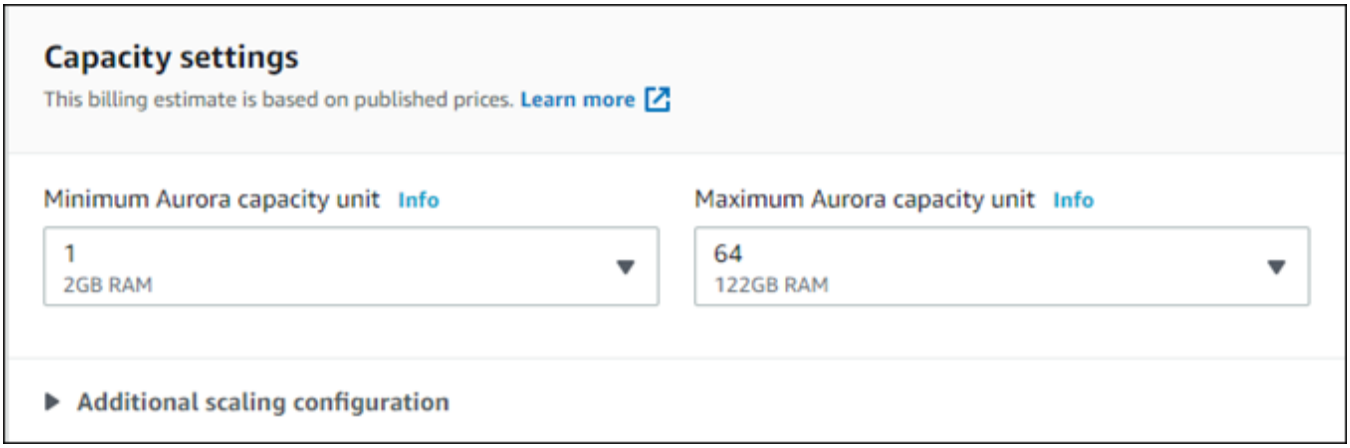
7. Pilih ukuran instans DB atau kapasitas klaster DB:

   - Untuk klon yang disediakan, pilih kelas instans DB.




   Anda dapat menerima pengaturan yang disediakan, atau Anda dapat menggunakan kelas instans DB yang berbeda untuk klon Anda.

   - Untuk Aurora Serverless v1 atau Aurora Serverless v2 klon, pilih pengaturan Kapasitas.




   Anda dapat menerima pengaturan yang disediakan, atau Anda dapat mengubahnya untuk klon Anda.

8. Pilih pengaturan lain sesuai kebutuhan untuk klon Anda. Untuk mempelajari selengkapnya tentang klaster DB Aurora dan pengaturan instans, lihat Membuat klaster DB Amazon Aurora.

9. Pilih Buat klon.

Ketika dibuat, klon akan tercantum dengan klaster DB Aurora Anda lainnya di bagian Basis data pada konsol dan menampilkan statusnya saat ini. Klon Anda siap digunakan ketika statusnya Tersedia.

AWS CLI

Menggunakan AWS CLI untuk mengkloning cluster Aurora DB Anda melibatkan langkah-langkah terpisah untuk membuat klaster klon dan menambahkan satu atau lebih instance DB ke dalamnya.

`restore-db-cluster-to-point-in-time` AWS CLI Perintah yang Anda gunakan menghasilkan cluster Aurora DB dengan data penyimpanan yang sama dengan cluster asli, tetapi tidak ada instance Aurora DB. Anda membuat instance DB secara terpisah setelah klon tersedia. Anda dapat memilih jumlah instans DB dan kelas instans mereka untuk memberikan kapasitas komputasi klon lebih atau kurang dari cluster asli. Langkah-langkah dalam prosesnya adalah sebagai berikut:

1. Buat klon dengan menggunakan perintah restore-db-cluster-to- point-in-time CLI.
2. Buat instance DB penulis untuk klon dengan menggunakan perintah create-db-instanceCLI.
3. (Opsional) Jalankan perintah create-db-instanceCLI tambahan untuk menambahkan satu atau beberapa instance pembaca ke klon klon. Menggunakan instance pembaca membantu meningkatkan ketersediaan tinggi dan membaca aspek skalabilitas klon. Anda dapat melewati langkah ini jika Anda hanya bermaksud menggunakan klon untuk pengembangan dan pengujian.

Topik

- Membuat klon
- Memeriksa status dan mendapatkan detail klon
- Membuat instans DB Aurora untuk klon Anda
- Parameter yang digunakan untuk kloning

Membuat klon

Gunakan perintah `restore-db-cluster-to-point-in-time` CLI untuk membuat klon klon awal.

Untuk membuat klon dari cluster Aurora DB sumber

- Gunakan perintah CLI `restore-db-cluster-to-point-in-time`. Tentukan nilai untuk parameter berikut. Dalam kasus tipikal ini, klon menggunakan mode mesin yang sama dengan cluster asli, baik yang disediakan atau Aurora Serverless v1.

- `--db-cluster-identifier` – Pilih nama yang bermakna untuk klon anda. Anda memberi nama klon saat Anda menggunakan perintah restore-db-cluster-to- point-in-time CLI. Anda kemudian meneruskan nama klon dalam perintah create-db-instanceCLI.
- `--restore-type` – Gunakan `copy-on-write` untuk membuat klon dari klaster DB sumber. Tanpa parameter ini, `restore-db-cluster-to-point-in-time` akan memulihkan klaster DB Aurora dan bukan membuat klon.
- `--source-db-cluster-identifier` – Gunakan nama klaster DB Aurora sumber yang ingin Anda kloning.
- `--use-latest-restorable-time`— Nilai ini menunjuk ke data volume terbaru yang dapat dipulihkan untuk cluster DB sumber. Gunakan untuk membuat klon.

Contoh berikut membuat klon bernama `my-clone` dari klaster bernama `my-source-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

Perintah tersebut menghasilkan objek JSON yang berisi detail klon. Periksa untuk memastikan bahwa klaster DB yang Anda kloning tersedia sebelum mencoba membuat instans DB untuk klon Anda. Untuk informasi selengkapnya, lihat Memeriksa status dan mendapatkan detail klon.

Misalnya, anggap Anda memiliki klaster bernama `tpch100g` yang ingin Anda kloning. Contoh Linux berikut membuat kloning kloning bernama`tpch100g-clone`, Aurora Serverless v2 instance writer bernama`tpch100g-clone-instance`, dan instance pembaca yang disediakan dinamai `tpch100g-clone-instance-2` untuk cluster baru.

Anda tidak perlu menyediakan beberapa parameter, seperti `--master-username` dan `--master-user-password`. Aurora secara otomatis menentukannya dari klaster asli. Anda perlu menentukan mesin DB yang akan digunakan. Dengan demikian, uji coba klaster baru untuk menentukan nilai yang tepat yang akan digunakan untuk parameter `--engine`.

Contoh ini juga mencakup `--serverless-v2-scaling-configuration` opsi saat membuat klaster klon. Dengan begitu, Anda dapat menambahkan Aurora Serverless v2 instance ke klon meskipun cluster asli tidak menggunakan Aurora Serverless v2.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.serverless \
  --engine aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance-2 \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r6g.2xlarge \
  --engine aurora-mysql
```

Untuk membuat klon dengan mode mesin yang berbeda dari sumber Aurora DB cluster

- Prosedur ini hanya berlaku untuk versi mesin lama yang mendukung Aurora Serverless v1. Misalkan Anda memiliki Aurora Serverless v1 cluster dan Anda ingin membuat klon yang merupakan cluster yang disediakan. Dalam hal ini, gunakan perintah `restore-db-cluster-to-point-in-time` CLI dan tentukan nilai nilai parameter yang sama seperti pada contoh sebelumnya, ditambah parameter tambahan ini:

- `--engine-mode`— Gunakan parameter ini hanya untuk membuat klon yang memiliki mode mesin yang berbeda dari cluster Aurora DB sumber. Parameter ini hanya berlaku untuk versi mesin lama yang mendukung Aurora Serverless v1. Pilih nilai yang akan dilewati `--engine-mode` sebagai berikut:
  - Gunakan `--engine-mode provisioned` untuk membuat klon cluster Aurora DB yang disediakan dari Aurora Serverless klaster DB.

    > **Note**
    >
    > Jika Anda berniat untuk menggunakan Aurora Serverless v2 dengan cluster yang dikloning dari Aurora Serverless v1, Anda masih menentukan mode mesin untuk klon sebagai`provisioned`. Kemudian Anda melakukan peningkatan tambahan dan langkah migrasi sesudahnya.

  - Gunakan `--engine-mode serverless` untuk membuat Aurora Serverless v1 klon dari cluster Aurora DB yang disediakan. Saat Anda menentukan mode `serverless` mesin, Anda juga dapat memilih`--scaling-configuration`.
- `--scaling-configuration`— (Opsional) Gunakan dengan `--engine-mode serverless` untuk mengkonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v1 klon. Jika Anda tidak menggunakan parameter ini, Aurora membuat Aurora Serverless v1 kloning menggunakan default Aurora Serverless v1 nilai kapasitas untuk mesin DB.

Contoh berikut membuat klon disediakan bernama`my-clone`, dari Aurora Serverless v1 Cluster DB bernama`my-source-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode provisioned \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode provisioned ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

Perintah ini menghasilkan objek JSON yang berisi detail klon yang Anda butuhkan untuk membuat instans DB. Anda tidak dapat melakukannya sampai klon (klaster DB Aurora kosong) memiliki status Tersedia.

> **Note**
>
> Perintah restore-db-cluster-to- point-in-time AWS CLI hanya mengembalikan cluster DB, bukan instance DB untuk cluster DB itu. Anda menjalankan create-db-instanceperintah untuk membuat instance DB untuk cluster DB yang dipulihkan. Dengan perintah itu, Anda menentukan pengidentifikasi cluster DB yang dipulihkan sebagai `--db-cluster-identifier` parameter. Anda dapat membuat instans DB hanya setelah perintah `restore-db-cluster-to-point-in-time` selesai dan klaster DB tersedia.
> Misalkan Anda memulai dengan Aurora Serverless v1 cluster dan bermaksud untuk memigrasikannya ke Aurora Serverless v2 klaster. Anda membuat klon yang disediakan dari Aurora Serverless v1 cluster sebagai langkah awal dalam migrasi. Untuk prosedur lengkap, termasuk upgrade versi yang diperlukan, lihatUpgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2.

Memeriksa status dan mendapatkan detail klon

Anda dapat menggunakan perintah berikut untuk memeriksa status klon klon yang baru dibuat.

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

Atau Anda dapat memperoleh status dan nilai-nilai lain yang Anda butuhkan untuk membuat instance DB untuk klon Anda dengan menggunakan AWS CLI kueri berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
```

```
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

Untuk Windows:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

Kueri ini menghasilkan output seperti yang berikut ini:

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

Membuat instans DB Aurora untuk klon Anda

Gunakan perintah create-db-instanceCLI untuk membuat instance DB untuk Anda Aurora Serverless v2 atau klon yang disediakan. Anda tidak membuat instance DB untuk Aurora Serverless v1 klon.

Instans DB mewarisi `--master-username` dan `--master-user-password` properti dari cluster DB sumber.

Contoh berikut membuat instance DB untuk klon yang disediakan.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.r6g.2xlarge \
    --engine aurora-mysql
```

Untuk Windows:

```
aws rds create-db-instance ^
```

```
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r6g.2xlarge ^
    --engine aurora-mysql
```

Contoh berikut menciptakan sebuah Aurora Serverless v2 Instans DB, untuk klon yang menggunakan versi mesin yang mendukung Aurora Serverless v2.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.serverless \
  --engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.serverless ^
    --engine aurora-mysql
```

Parameter yang digunakan untuk kloning

Tabel berikut merangkum berbagai parameter yang digunakan dengan `restore-db-cluster-to-point-in-time` untuk mengkloning klaster DB Aurora.

| Parameter | Deskripsi |
| --- | --- |
| `--source-db-cluster-identifier` | Pilih parameter ini untuk menggunakan nama klaster DB Aurora sumber yang ingin Anda kloning. |
| `--db-cluster-identifier` | Pilih nama yang berarti untuk klon Anda saat Anda membuatnya dengan `restore-db-cluster-to-point-in-time` perintah. Kemudian, Anda meneruskan nama ini ke perintah `create-db-instance` . |

| Parameter | Deskripsi |
| --- | --- |
| `--restore-type` | Tentukan `copy-on-write` sebagai `--restore-type` untuk membuat klon klaster DB sumber dan bukan memulihkan klaster DB Aurora sumber. |
| `--use-latest-restorable-time` | Nilai ini menunjuk ke data volume terbaru yang dapat dipulihkan untuk cluster DB sumber. Gunakan untuk membuat klon. |
| `--serverless-v2-scaling-configuration` | (Versi terbaru yang mendukung Aurora Serverless v2) Gunakan parameter ini untuk mengkonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v2 klon. Jika Anda tidak menentukan parameter ini, Anda tidak dapat membuat apa pun Aurora Serverless v2 instance di klon klon sampai Anda memodifikasi cluster untuk menambahkan atribut ini. |
| `--engine-mode` | (Versi lama yang mendukung Aurora Serverless v1 hanya) Gunakan parameter ini untuk membuat klon yang memiliki tipe berbeda dari cluster Aurora DB sumber, dengan salah satu nilai berikut:<br>• Gunakan `provisioned` untuk membuat klon yang disediakan dari Aurora Serverless v1 klaster DB.<br>• Gunakan `serverless` untuk membuat Aurora Serverless v1 kloning dari yang disediakan atau Aurora Serverless v2 klaster DB.<br>Saat Anda menentukan mode `serverless` mesin, Anda juga dapat memilih`--scaling-configuration` . |
| `--scaling-configuration` | (Versi lama yang mendukung Aurora Serverless v1 hanya) Gunakan parameter ini untuk mengkonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v1 klon. Jika Anda tidak menentukan parameter ini, Aurora membuat klon menggunakan nilai kapasitas default untuk mesin DB. |

Untuk informasi tentang kloning lintas VPC dan lintas akun, lihat bagian berikut.

Topik

- VPCKloning silang dengan Amazon Aurora
- Kloning lintas akun dengan AWS RAM dan Amazon Aurora


# VPCKloning silang dengan Amazon Aurora

Misalkan Anda ingin memaksakan kontrol akses jaringan yang berbeda pada cluster asli dan klon. Misalnya, Anda dapat menggunakan kloning untuk membuat salinan klaster Aurora produksi yang VPC berbeda untuk pengembangan dan pengujian. Atau Anda dapat membuat klon sebagai bagian dari migrasi dari subnet publik ke subnet pribadi, untuk meningkatkan keamanan database Anda.

Bagian berikut menunjukkan cara mengatur konfigurasi jaringan untuk klon sehingga cluster asli dan klon dapat mengakses node penyimpanan Aurora yang sama, bahkan dari subnet yang berbeda atau berbeda. VPCs Memverifikasi sumber daya jaringan terlebih dahulu dapat menghindari kesalahan selama kloning yang mungkin sulit didiagnosis.

Jika Anda tidak terbiasa dengan bagaimana Aurora berinteraksi denganVPCs, subnet, dan grup subnet DB, lihat dulu. Amazon VPC dan Aurora Anda dapat mengerjakan tutorial di bagian itu untuk membuat sumber daya semacam ini di AWS konsol, dan memahami bagaimana mereka cocok bersama.

Karena langkah-langkahnya melibatkan peralihan antara EC2 layanan RDS dan layanan, contoh menggunakan AWS CLI perintah untuk membantu Anda memahami cara mengotomatiskan operasi tersebut dan menyimpan output.

Topik


- Sebelum Anda mulai
- Mengumpulkan informasi tentang lingkungan jaringan
- Membuat sumber daya jaringan untuk klon
- Membuat klon Aurora dengan pengaturan jaringan baru
- Memindahkan cluster dari subnet publik ke subnet pribadi
- End-to-end contoh membuat VPC klon silang


## Sebelum Anda mulai

Sebelum Anda mulai menyiapkan VPC klon silang, pastikan untuk memiliki sumber daya berikut:

- Cluster Aurora DB untuk digunakan sebagai sumber kloning. Jika ini adalah pertama kalinya Anda membuat cluster Aurora DB, lihat tutorial Memulai dengan Amazon Aurora untuk menyiapkan cluster menggunakan mesin database My SQL atau PostgreSQL.
- SedetikVPC, jika Anda berniat membuat VPC klon silang. Jika Anda tidak memiliki VPC untuk digunakan untuk klon, lihat Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 atauTutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda).

## Mengumpulkan informasi tentang lingkungan jaringan

Dengan VPC kloning silang, lingkungan jaringan dapat berbeda secara substansional antara cluster asli dan klonnya. Sebelum Anda membuat klon, kumpulkan dan catat informasi tentang, subnetVPC, grup subnet DB, dan AZs digunakan dalam cluster asli. Dengan begitu, Anda dapat meminimalkan kemungkinan masalah. Jika masalah jaringan terjadi, Anda tidak perlu mengganggu aktivitas pemecahan masalah apa pun untuk mencari informasi diagnostik. Bagian berikut menunjukkan CLI contoh untuk mengumpulkan jenis informasi ini. Anda dapat menyimpan detail dalam format mana pun yang nyaman untuk dikonsultasikan saat membuat klon dan melakukan pemecahan masalah apa pun.

- Langkah 1: Periksa Availability Zones dari cluster asli
- Langkah 2: Periksa grup subnet DB dari cluster asli
- Langkah 3: Periksa subnet dari cluster asli
- Langkah 4: Periksa Availability Zones dari instans DB di cluster asli
- Langkah 5: Periksa yang dapat VPCs Anda gunakan untuk klon

### Langkah 1: Periksa Availability Zones dari cluster asli

Sebelum Anda membuat klon, verifikasi mana yang AZs digunakan cluster asli untuk penyimpanannya. Seperti dijelaskan dalamPenyimpanan Amazon Aurora, penyimpanan untuk setiap cluster Aurora dikaitkan dengan tepat tiga. AZs Karena Klaster DB Amazon Aurora mengambil keuntungan dari pemisahan komputasi dan penyimpanan, aturan ini benar terlepas dari berapa banyak instance dalam cluster.

Misalnya, jalankan CLI perintah seperti berikut ini, ganti nama cluster Anda sendiri. *my_cluster* Contoh berikut menghasilkan daftar yang diurutkan menurut abjad dengan nama AZ.

```
aws rds describe-db-clusters \
  --db-cluster-identifier my_cluster \
```

```
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' \
  --output text
```

Contoh berikut menunjukkan output sampel dari perintah sebelumnya. `describe-db-clusters` Ini menunjukkan bahwa penyimpanan untuk cluster Aurora selalu menggunakan tiga. AZs

```
us-east-1c
us-east-1d
us-east-1e
```

Untuk membuat klon di lingkungan jaringan yang tidak memiliki semua sumber daya untuk terhubung ke iniAZs, Anda harus membuat subnet yang terkait dengan setidaknya dua subnet tersebutAZs, dan kemudian membuat grup subnet DB yang berisi dua atau tiga subnet tersebut. Contoh berikut menunjukkan caranya.

Langkah 2: Periksa grup subnet DB dari cluster asli

Jika Anda ingin menggunakan jumlah subnet yang sama untuk klon seperti pada cluster asli, Anda bisa mendapatkan jumlah subnet dari grup subnet DB dari cluster asli. Grup subnet Aurora DB berisi setidaknya dua subnet, masing-masing terkait dengan AZ yang berbeda. Catat subnet mana AZs yang terkait dengannya.

Contoh berikut menunjukkan bagaimana menemukan kelompok subnet DB dari cluster asli, dan kemudian bekerja mundur ke yang sesuai. AZs Gantikan nama cluster Anda *`my_cluster`* dengan perintah pertama. Gantikan nama grup subnet DB *`my_subnet`* dengan perintah kedua.

```
aws rds describe-db-clusters --db-cluster-identifier my_cluster \
  --query '*[].DBSubnetGroup' --output text

aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetAvailabilityZone.Name]' --output text
```

Output sampel mungkin terlihat mirip dengan berikut ini, untuk cluster dengan grup subnet DB yang berisi dua subnet. Dalam hal ini, `two-subnets` adalah nama yang ditentukan saat membuat grup subnet DB.

```
two-subnets

us-east-1d
```

```
us-east-1c
```

Untuk cluster di mana grup subnet DB berisi tiga subnet, output mungkin terlihat mirip dengan berikut ini.

```
three-subnets

us-east-1f
us-east-1d
us-east-1c
```

Langkah 3: Periksa subnet dari cluster asli

Jika Anda memerlukan detail lebih lanjut tentang subnet di cluster asli, jalankan AWS CLI perintah yang mirip dengan berikut ini. Anda dapat memeriksa atribut subnet seperti rentang alamat IP, pemilik, dan sebagainya. Dengan begitu, Anda dapat menentukan apakah akan menggunakan subnet yang berbeda dalam hal yang samaVPC, atau membuat subnet dengan karakteristik serupa di tempat yang berbeda. VPC

Temukan subnet IDs dari semua subnet yang tersedia di Anda. VPC

```
aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc \
  --query '*[].[SubnetId]' --output text
```

Temukan subnet yang tepat digunakan dalam grup subnet DB Anda.

```
aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetIdentifier]' --output text
```

Kemudian tentukan subnet yang ingin Anda selidiki dalam daftar, seperti pada perintah berikut. Ganti nama subnet Anda untuk *my_subnet_1* dan sebagainya.

```
aws ec2 describe-subnets \
  --subnet-ids '["my_subnet_1","my_subnet2","my_subnet3"]'
```

Contoh berikut menunjukkan sebagian output dari `describe-subnets` perintah tersebut. Output menunjukkan beberapa atribut penting yang dapat Anda lihat untuk setiap subnet, seperti AZ terkait dan bagian darinya. VPC

```
{
    'Subnets': [
        {
            'AvailabilityZone': 'us-east-1d',
            'AvailableIpAddressCount': 54,
            'CidrBlock': '10.0.0.64/26',
            'State': 'available',
            'SubnetId': 'subnet-000a0bca00e0b0000',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
        },
        {
            'AvailabilityZone': 'us-east-1c',
            'AvailableIpAddressCount': 55,
            'CidrBlock': '10.0.0.0/26',
            'State': 'available',
            'SubnetId': 'subnet-4b4dbfe4d4a4fd4c4',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
```

Langkah 4: Periksa Availability Zones dari instans DB di cluster asli

Anda dapat menggunakan prosedur ini untuk memahami yang AZs digunakan untuk instance DB di cluster asli. Dengan begitu, Anda dapat mengatur hal yang sama persis AZs untuk instance DB di klon. Anda juga dapat menggunakan lebih banyak atau lebih sedikit instans DB di klon tergantung pada apakah klon digunakan untuk produksi, pengembangan dan pengujian, dan sebagainya.

Untuk setiap instance di cluster asli, jalankan perintah seperti berikut ini. Pastikan bahwa instance telah selesai dibuat dan berada dalam `Available` status terlebih dahulu. Gantikan pengidentifikasi instance untuk*my_instance*.

```
aws rds describe-db-instances --db-instance-identifier my_instance \
  --query '*[].AvailabilityZone' --output text
```

Contoh berikut menunjukkan output dari menjalankan perintah sebelumnya. `describe-db-instances` Cluster Aurora memiliki empat instance database. Oleh karena itu, kami menjalankan perintah empat kali, mengganti pengidentifikasi instans DB yang berbeda setiap kali. Output menunjukkan bagaimana instans DB tersebut tersebar di maksimal tigaAZs.

```
us-east-1a
us-east-1c
```

```
us-east-1d
us-east-1a
```

Setelah klon dibuat dan Anda menambahkan instance DB ke dalamnya, Anda dapat menentukan nama AZ yang sama ini dalam perintah. `create-db-instance` Anda dapat melakukannya untuk menyiapkan instance DB di cluster baru yang dikonfigurasi untuk persis sama AZs seperti di cluster asli.

Langkah 5: Periksa yang dapat VPCs Anda gunakan untuk klon

Jika Anda berniat untuk membuat klon yang berbeda VPC dari aslinya, Anda bisa mendapatkan daftar yang VPC IDs tersedia untuk akun Anda. Anda juga dapat melakukan langkah ini jika Anda perlu membuat subnet tambahan yang VPC sama dengan cluster asli. Ketika Anda menjalankan perintah untuk membuat subnet, Anda menentukan VPC ID sebagai parameter.

Untuk membuat daftar semua VPCs untuk akun Anda, jalankan CLI perintah berikut:

```
aws ec2 describe-vpcs --query '*[].[VpcId]' --output text
```

Contoh berikut menunjukkan output sampel dari perintah sebelumnya. `describe-vpcs` Output menunjukkan bahwa ada empat VPCs di AWS akun saat ini yang dapat digunakan sebagai sumber atau tujuan untuk VPC kloning silang.

```
vpc-fd111111
vpc-2222e2cd2a222f22e
vpc-33333333a33333d33
vpc-4ae4d4de4a4444dad
```

Anda dapat menggunakan yang VPC sama dengan tujuan untuk klon, atau yang berbedaVPC. Jika cluster asli dan klonnya samaVPC, Anda dapat menggunakan kembali grup subnet DB yang sama untuk klon. Anda juga dapat membuat grup subnet DB yang berbeda. Misalnya, grup subnet DB baru mungkin menggunakan subnet pribadi, sedangkan grup subnet DB cluster asli mungkin menggunakan subnet publik. Jika Anda membuat klon dalam yang berbedaVPC, pastikan bahwa ada cukup subnet di baru VPC dan bahwa subnet dikaitkan dengan hak AZs dari cluster asli.

## Membuat sumber daya jaringan untuk klon

Jika saat mengumpulkan informasi jaringan Anda menemukan bahwa sumber daya jaringan tambahan diperlukan untuk klon, Anda dapat membuat sumber daya tersebut sebelum mencoba

mengatur klon. Misalnya, Anda mungkin perlu membuat lebih banyak subnet, subnet yang terkait dengan spesifikAZs, atau grup subnet DB baru.

- Langkah 1: Buat subnet untuk klon
- Langkah 2: Buat grup subnet DB untuk klon

### Langkah 1: Buat subnet untuk klon

Jika Anda perlu membuat subnet baru untuk klon, jalankan perintah yang mirip dengan berikut ini. Anda mungkin perlu melakukan ini saat membuat klon di tempat yang berbedaVPC, atau saat membuat beberapa perubahan jaringan lain seperti menggunakan subnet pribadi alih-alih subnet publik.

AWS secara otomatis menghasilkan ID subnet. Gantikan nama klon VPC untuk*my_vpc*. Pilih rentang alamat untuk `--cidr-block` opsi untuk mengizinkan setidaknya 16 alamat IP dalam rentang tersebut. Anda dapat menyertakan properti lain yang ingin Anda tentukan. Jalankan perintah `aws ec2 create-subnet help` untuk melihat semua pilihan.

```
aws ec2 create-subnet --vpc-id my_vpc \
  --availability-zone AZ_name --cidr-block IP_range
```

Contoh berikut menunjukkan beberapa atribut penting dari subnet yang baru dibuat.

```
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1b',
        'AvailableIpAddressCount': 59,
        'CidrBlock': '10.0.0.64/26',
        'State': 'available',
        'SubnetId': 'subnet-44b4a44f4e44db444',
        'VpcId': 'vpc-555fc5df555e555dc',
        ...
        }
}
```

### Langkah 2: Buat grup subnet DB untuk klon

Jika Anda membuat klon dalam subnet yang berbedaVPC, atau kumpulan subnet yang berbeda dalam hal yang samaVPC, maka Anda membuat grup subnet DB baru dan menentukannya saat membuat klon.

Pastikan Anda mengetahui semua detail berikut. Anda dapat menemukan semua ini dari output dari contoh sebelumnya.

1. VPCdari cluster asli. Untuk petunjuk, silakan lihat Langkah 3: Periksa subnet dari cluster asli.
2. VPCdari klon, jika Anda membuatnya berbedaVPC. Untuk petunjuk, silakan lihat Langkah 5: Periksa yang dapat VPCs Anda gunakan untuk klon.
3. Tiga AZs terkait dengan penyimpanan Aurora untuk cluster asli. Untuk petunjuk, silakan lihat Langkah 1: Periksa Availability Zones dari cluster asli.
4. Dua atau tiga AZs terkait dengan grup subnet DB untuk cluster asli. Untuk petunjuk, silakan lihat Langkah 2: Periksa grup subnet DB dari cluster asli.
5. Subnet IDs dan terkait AZs dari semua subnet dalam yang ingin VPC Anda gunakan untuk klon. Gunakan `describe-subnets` perintah yang sama seperti diLangkah 3: Periksa subnet dari cluster asli, menggantikan VPC ID tujuanVPC.

Periksa berapa banyak AZs keduanya terkait dengan penyimpanan cluster asli, dan terkait dengan subnet di tujuanVPC. Agar berhasil membuat klon, harus ada dua atau tiga AZs kesamaan. Jika Anda memiliki kurang dari dua AZs kesamaan, kembalilah keLangkah 1: Buat subnet untuk klon. Buat satu, dua, atau tiga subnet baru yang AZs terkait dengan penyimpanan cluster asli.

Pilih subnet di tujuan VPC yang terkait dengan penyimpanan Aurora di cluster awalnya. AZs Idealnya, pilih tigaAZs. Melakukannya memberi Anda fleksibilitas paling besar untuk menyebarkan instans DB klon di beberapa AZs untuk ketersediaan sumber daya komputasi yang tinggi.

Jalankan perintah yang mirip dengan berikut ini untuk membuat grup subnet DB baru. Gantikan subnet Anda dalam daftar. IDs Jika Anda menentukan subnet IDs menggunakan variabel lingkungan, berhati-hatilah untuk mengutip daftar `--subnet-ids` parameter dengan cara yang mempertahankan tanda kutip ganda di sekitar. IDs

```
aws rds create-db-subnet-group --db-subnet-group-name my_subnet_group \
  --subnet-ids '["my_subnet_1","my_subnet_2","my_subnet3"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets for clone'
```

Contoh berikut menunjukkan output sebagian dari `create-db-subnet-group` perintah.

```
{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'my_subnet_group',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets for clone',
```

```
        'VpcId': 'vpc-555fc5df555e555dc',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'my_subnet_1',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1c'
                },
                'SubnetStatus': 'Active'
            },
            {
                'SubnetIdentifier': 'my_subnet_2',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1d'
                },
                'SubnetStatus': 'Active'
            }
            ...
        ],
        'SupportedNetworkTypes': [
            'IPV4'
        ]
    }
}
```

Pada titik ini, Anda belum benar-benar membuat klon. Anda telah membuat semua sumber daya yang relevan VPC dan subnet sehingga Anda dapat menentukan parameter yang sesuai dengan `create-db-instance` perintah `restore-db-cluster-to-point-in-time` dan saat membuat klon.

## Membuat klon Aurora dengan pengaturan jaringan baru

Setelah Anda memastikan bahwa konfigurasi yang tepat dariVPCs, subnetAZs, dan grup subnet tersedia untuk digunakan cluster baru, Anda dapat melakukan operasi kloning yang sebenarnya. CLIContoh berikut menyoroti opsi seperti`--db-subnet-group-name`,`--availability-zone`, dan `--vpc-security-group-ids` yang Anda tentukan pada perintah untuk mengatur klon dan instance DB-nya.

- Langkah 1: Tentukan grup subnet DB untuk klon
- Langkah 2: Tentukan pengaturan jaringan untuk instance di klon
- Langkah 3: Membangun konektivitas dari sistem klien ke klon

Langkah 1: Tentukan grup subnet DB untuk klon

Saat Anda membuat klon, Anda dapat mengonfigurasi semua pengaturan kananVPC, subnet, dan AZ dengan menentukan grup subnet DB. Gunakan perintah dalam contoh sebelumnya untuk memverifikasi semua hubungan dan pemetaan yang masuk ke grup subnet DB.

Misalnya, perintah berikut menunjukkan kloning cluster asli ke klon. Pada contoh pertama, cluster sumber dikaitkan dengan dua subnet dan klon dikaitkan dengan tiga subnet. Contoh kedua menunjukkan kasus sebaliknya, kloning dari cluster dengan tiga subnet ke cluster dengan dua subnet.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-3-subnets \
  --db-cluster-identifier cluster-cloned-to-2-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name two-subnets
```

Jika Anda bermaksud menggunakan instance Aurora Serverless v2 di klon, sertakan `--serverless-v2-scaling-configuration` opsi saat Anda membuat klon, seperti yang ditunjukkan. Melakukannya memungkinkan Anda menggunakan `db.serverless` kelas saat membuat instance DB di klon. Anda juga dapat memodifikasi klon nanti untuk menambahkan atribut konfigurasi penskalaan ini. Nomor kapasitas dalam contoh ini memungkinkan setiap instance v2 Tanpa Server di cluster untuk menskalakan antara 2 dan 32 Unit Kapasitas Aurora (). ACUs Untuk informasi tentang fitur Aurora Serverless v2 dan cara memilih rentang kapasitas, lihat. Menggunakan Aurora Serverless v2

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-2-subnets \
  --db-cluster-identifier cluster-cloned-to-3-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name three-subnets \
  --serverless-v2-scaling-configuration 'MinCapacity=2,MaxCapacity=32'
```

Terlepas dari jumlah subnet yang digunakan oleh instans DB, penyimpanan Aurora untuk cluster sumber dan klon dikaitkan dengan tiga. AZs Contoh berikut mencantumkan yang AZs terkait dengan cluster asli dan klon, untuk kedua `restore-db-cluster-to-point-in-time` perintah dalam contoh sebelumnya.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-with-3-subnets \
```

```
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-with-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d
```

Karena setidaknya dua AZs tumpang tindih antara setiap pasangan cluster asli dan klon, kedua cluster dapat mengakses penyimpanan Aurora dasar yang sama.

Langkah 2: Tentukan pengaturan jaringan untuk instance di klon

Saat Anda membuat instance DB di klon, secara default mereka mewarisi grup subnet DB dari cluster itu sendiri. Dengan begitu, Aurora secara otomatis menetapkan setiap instance ke subnet tertentu, dan membuatnya di AZ yang terkait dengan subnet. Pilihan ini nyaman, terutama untuk sistem pengembangan dan pengujian, karena Anda tidak perlu melacak subnet IDs atau AZs sambil menambahkan instance baru ke klon.

Sebagai alternatif, Anda dapat menentukan AZ saat Anda membuat instance Aurora DB untuk klon. AZ yang Anda tentukan harus dari himpunan AZs yang terkait dengan klon. Jika grup subnet DB yang Anda gunakan untuk klon hanya berisi dua subnet, maka Anda hanya dapat memilih dari yang AZs terkait dengan dua subnet tersebut. Pilihan ini menawarkan fleksibilitas dan ketahanan

untuk sistem yang sangat tersedia, karena Anda dapat memastikan bahwa instance penulis dan instance pembaca siaga berbeda. AZs Atau jika Anda menambahkan pembaca tambahan ke cluster, Anda dapat memastikan bahwa mereka tersebar di tigaAZs. Dengan begitu, bahkan dalam kasus kegagalan AZ yang jarang terjadi, Anda masih memiliki instance penulis dan instance pembaca lain di dua lainnyaAZs.

Contoh berikut menambahkan instance DB yang disediakan ke klaster Aurora SQL Postgre kloning yang menggunakan grup subnet DB kustom.

```
aws rds create-db-instance --db-cluster-identifier my_aurora_postgresql_clone \
  --db-instance-identifier my_postgres_instance \
  --db-subnet-group-name my_new_subnet \
  --engine aurora-postgresql \
  --db-instance-class db.t4g.medium
```

Contoh berikut menunjukkan sebagian output dari perintah tersebut.

```
{
  'DBInstanceIdentifier': 'my_postgres_instance',
  'DBClusterIdentifier': 'my_aurora_postgresql_clone',
  'DBInstanceClass': 'db.t4g.medium',
  'DBInstanceStatus': 'creating'
  ...
}
```

Contoh berikut menambahkan instance Aurora Serverless v2 DB ke SQL klon Aurora My yang menggunakan grup subnet DB kustom. Untuk dapat menggunakan instance v2 Tanpa Server, pastikan untuk menentukan `--serverless-v2-scaling-configuration` opsi untuk `restore-db-cluster-to-point-in-time` perintah, seperti yang ditunjukkan pada contoh sebelumnya.

```
aws rds create-db-instance --db-cluster-identifier my_aurora_mysql_clone \
  --db-instance-identifier my_mysql_instance \
  --db-subnet-group-name my_other_new_subnet \
  --engine aurora-mysql \
  --db-instance-class db.serverless
```

Contoh berikut menunjukkan sebagian output dari perintah tersebut.

```
{
  'DBInstanceIdentifier': 'my_mysql_instance',
```

```
  'DBClusterIdentifier': 'my_aurora_mysql_clone',
  'DBInstanceClass': 'db.serverless',
  'DBInstanceStatus': 'creating'
  ...
}
```

Langkah 3: Membangun konektivitas dari sistem klien ke klon

Jika Anda sudah terhubung ke cluster Aurora dari sistem klien, Anda mungkin ingin mengizinkan jenis konektivitas yang sama ke klon baru. Misalnya, Anda mungkin tersambung ke cluster asli dari instans atau instance Amazon Cloud9. EC2 Untuk mengizinkan koneksi dari sistem klien yang sama, atau yang baru yang Anda buat di tujuanVPC, siapkan grup subnet DB dan grup VPC keamanan yang setara seperti di VPC file. Kemudian tentukan grup subnet dan grup keamanan saat Anda membuat klon.

Contoh berikut menyiapkan klon Aurora Serverless v2. Konfigurasi itu didasarkan pada kombinasi `--engine-mode provisioned` dan `--serverless-v2-scaling-configuration` saat membuat cluster DB, dan kemudian `--db-instance-class db.serverless` saat membuat setiap instans DB di cluster. Mode `provisioned` mesin adalah default, sehingga Anda dapat menghilangkan opsi itu jika Anda mau.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier serverless-sql-postgres\
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-subnet-group-name 'default-vpc-1234' \
  --vpc-security-group-ids 'sg-4567' \
  --serverless-v2-scaling-configuration 'MinCapacity=0.5,MaxCapacity=16' \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

Kemudian, saat membuat instance DB di klon, tentukan opsi yang sama`--db-subnet-group-name`. Secara opsional, Anda dapat menyertakan `--availability-zone` opsi dan menentukan salah satu yang AZs terkait dengan subnet dalam grup subnet itu. AZ itu juga harus menjadi salah satu yang AZs terkait dengan cluster asli.

```
aws rds create-db-instance \
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-instance-identifier serverless-sql-postgres-clone-instance \
  --db-instance-class db.serverless \
  --db-subnet-group-name 'default-vpc-987zyx654' \
```

```
  --availability-zone 'us-east-1c' \
  --engine aurora-postgresql
```

## Memindahkan cluster dari subnet publik ke subnet pribadi

Anda dapat menggunakan kloning untuk memigrasikan cluster antara subnet publik dan pribadi. Anda dapat melakukan ini saat menambahkan lapisan keamanan tambahan ke aplikasi Anda sebelum menerapkannya ke produksi. Untuk contoh ini, Anda seharusnya sudah menyiapkan subnet pribadi dan NAT gateway sebelum memulai proses kloning dengan Aurora.

Untuk langkah-langkah yang melibatkan Aurora, Anda dapat mengikuti langkah-langkah umum yang sama seperti pada contoh sebelumnya untuk dan. Mengumpulkan informasi tentang lingkungan jaringan Membuat klon Aurora dengan pengaturan jaringan baru Perbedaan utama adalah bahwa bahkan jika Anda memiliki subnet publik yang memetakan ke semua AZs dari cluster asli, sekarang Anda harus memverifikasi bahwa Anda memiliki cukup subnet pribadi untuk cluster Aurora, dan bahwa subnet tersebut terkait dengan semua yang sama yang AZs digunakan untuk penyimpanan Aurora di cluster asli. Mirip dengan kasus penggunaan kloning lainnya, Anda dapat membuat grup subnet DB untuk klon dengan tiga atau dua subnet pribadi yang terkait dengan yang diperlukan. AZs Namun, jika Anda menggunakan dua subnet pribadi di grup subnet DB, Anda harus memiliki subnet pribadi ketiga yang terkait dengan AZ ketiga yang digunakan untuk penyimpanan Aurora di cluster asli.

Anda dapat berkonsultasi dengan daftar periksa ini untuk memverifikasi bahwa semua persyaratan ada untuk melakukan operasi kloning jenis ini.

- Catat tiga AZs yang terkait dengan cluster asli. Untuk petunjuk, silakan lihat Langkah 1: Periksa Availability Zones dari cluster asli.
- Rekam tiga atau dua AZs yang terkait dengan subnet publik di grup subnet DB untuk cluster asli. Untuk petunjuk, silakan lihat Langkah 3: Periksa subnet dari cluster asli.
- Buat subnet pribadi yang memetakan ke ketiga AZs yang terkait dengan cluster asli. Juga lakukan pengaturan jaringan lainnya, seperti membuat NAT gateway, untuk dapat berkomunikasi dengan subnet pribadi. Untuk petunjuk, lihat Membuat subnet di Panduan Pengguna Amazon Virtual Private Cloud.
- Buat grup subnet DB baru yang berisi tiga atau dua subnet pribadi yang terkait dengan AZs dari titik pertama. Untuk petunjuk, silakan lihat Langkah 2: Buat grup subnet DB untuk klon.

Ketika semua prasyarat sudah ada, Anda dapat menjeda aktivitas database pada cluster asli saat Anda membuat klon dan mengalihkan aplikasi Anda untuk menggunakannya. Setelah klon dibuat

dan Anda memverifikasi bahwa Anda dapat terhubung ke sana, menjalankan kode aplikasi Anda, dan sebagainya, Anda dapat menghentikan penggunaan cluster asli.

## End-to-end contoh membuat VPC klon silang

Membuat klon yang berbeda VPC dari aslinya menggunakan langkah-langkah umum yang sama seperti pada contoh sebelumnya. Karena VPC ID adalah properti dari subnet, Anda tidak benar-benar menentukan VPC ID sebagai parameter saat menjalankan salah satu RDS CLI perintah. Perbedaan utamanya adalah Anda lebih mungkin perlu membuat subnet baru, subnet baru yang dipetakan ke spesifikAZs, grup VPC keamanan, dan grup subnet DB baru. Itu terutama benar jika ini adalah cluster Aurora pertama yang Anda buat di dalamnya. VPC

Anda dapat berkonsultasi dengan daftar periksa ini untuk memverifikasi bahwa semua persyaratan ada untuk melakukan operasi kloning jenis ini.

- Catat tiga AZs yang terkait dengan cluster asli. Untuk petunjuk, silakan lihat Langkah 1: Periksa Availability Zones dari cluster asli.
- Rekam tiga atau dua AZs yang terkait dengan subnet dalam grup subnet DB untuk cluster asli. Untuk petunjuk, silakan lihat Langkah 2: Periksa grup subnet DB dari cluster asli.
- Buat subnet yang memetakan ke ketiga AZs yang terkait dengan cluster asli. Untuk petunjuk, silakan lihat Langkah 1: Buat subnet untuk klon.
- Lakukan pengaturan jaringan lainnya, seperti menyiapkan grup VPC keamanan, untuk sistem klien, server aplikasi, dan sebagainya untuk dapat berkomunikasi dengan instans DB di klon. Untuk petunjuk, silakan lihat Mengontrol akses dengan grup keamanan.
- Buat grup subnet DB baru yang berisi tiga atau dua subnet yang terkait dengan AZs dari titik pertama. Untuk petunjuk, silakan lihat Langkah 2: Buat grup subnet DB untuk klon.

Ketika semua prasyarat sudah ada, Anda dapat menjeda aktivitas database pada cluster asli saat Anda membuat klon dan mengalihkan aplikasi Anda untuk menggunakannya. Setelah klon dibuat dan Anda memverifikasi bahwa Anda dapat terhubung ke sana, menjalankan kode aplikasi Anda, dan sebagainya, Anda dapat mempertimbangkan apakah akan menjaga kedua asli dan klon berjalan, atau menghentikan penggunaan cluster asli.

Contoh Linux berikut menunjukkan urutan AWS CLI operasi untuk mengkloning cluster Aurora DB dari VPC satu ke yang lain. Beberapa bidang yang tidak relevan dengan contoh tidak ditampilkan dalam output perintah.

Pertama, kami memeriksa IDs sumber dan tujuanVPCs. Nama deskriptif yang Anda tetapkan VPC saat Anda membuatnya direpresentasikan sebagai tag dalam metadata. VPC

```
$ aws ec2 describe-vpcs --query '*[].[VpcId,Tags]'
[
    [
        'vpc-0f0c0fc0000b0ffb0',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-source'
            }
        ]
    ],
    [
        'vpc-9e99d9f99a999bd99',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-dest'
            }
        ]
    ]
]
```

Cluster asli sudah ada di sumbernyaVPC. Untuk mengatur klon menggunakan set yang sama AZs untuk penyimpanan Aurora, kami memeriksa AZs yang digunakan oleh cluster asli.

```
$ aws rds describe-db-clusters --db-cluster-identifier original-cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

Kami memastikan ada subnet yang sesuai dengan yang AZs digunakan oleh cluster asli:`us-east-1c`,`us-east-1d`, dan`us-east-1f`.

```
$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
  --availability-zone us-east-1c --cidr-block 10.0.0.128/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1c',
        'SubnetId': 'subnet-3333a33be3ef3e333',
        'VpcId': 'vpc-9e99d9f99a999bd99',
```

```
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1d --cidr-block 10.0.0.160/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1d',
        'SubnetId': 'subnet-4eeb444cd44b4d444',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1f --cidr-block 10.0.0.224/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1f',
        'SubnetId': 'subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}
```

Contoh ini menegaskan bahwa ada subnet yang memetakan ke yang diperlukan AZs di tujuanVPC.

```
aws ec2 describe-subnets --query 'sort_by(*[] | [?VpcId == `vpc-9e99d9f99a999bd99`] |
[].{SubnetId:SubnetId,VpcId:VpcId,AvailabilityZone:AvailabilityZone},
 &AvailabilityZone)' --output table

-------------------------------------------------------------------------
|                           DescribeSubnets                             |
+------------------+---------------------------+-------------------------+
| AvailabilityZone |         SubnetId          |          VpcId          |
+------------------+---------------------------+-------------------------+
|  us-east-1a      |  subnet-000ff0e00000c0aea  |  vpc-9e99d9f99a999bd99  |
|  us-east-1b      |  subnet-1111d111111ca11b1  |  vpc-9e99d9f99a999bd99  |
|  us-east-1c      | subnet-3333a33be3ef3e333   |  vpc-9e99d9f99a999bd99  |
|  us-east-1d      | subnet-4eeb444cd44b4d444   |  vpc-9e99d9f99a999bd99  |
|  us-east-1f      | subnet-66eea6666fb66d66c   |  vpc-9e99d9f99a999bd99  |
+------------------+---------------------------+-------------------------+
```

Sebelum membuat cluster Aurora DB diVPC, Anda harus memiliki grup subnet DB dengan subnet yang memetakan ke yang digunakan AZs untuk penyimpanan Aurora. Saat Anda membuat cluster

biasa, Anda dapat menggunakan set tigaAZs. Saat Anda mengkloning cluster yang ada, grup subnet harus cocok setidaknya dua dari tiga AZs yang digunakannya untuk penyimpanan Aurora.

```
$ aws rds create-db-subnet-group \
  --db-subnet-group-name subnet-group-in-other-vpc \
  --subnet-ids
 '["subnet-3333a33be3ef3e333","subnet-4eeb444cd44b4d444","subnet-66eea6666fb66d66c"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c'

{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'subnet-group-in-other-vpc',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'subnet-4eeb444cd44b4d444',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1d' }
            },
            {
                'SubnetIdentifier': 'subnet-3333a33be3ef3e333',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1c' }
            },
            {
                'SubnetIdentifier': 'subnet-66eea6666fb66d66c',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1f' }
            }
        ]
    }
}
```

Sekarang subnet dan grup subnet DB sudah ada. Contoh berikut menunjukkan `restore-db-cluster-to-point-in-time` yang mengkloning cluster. `--db-subnet-group-name`Opsi ini mengaitkan klon dengan kumpulan subnet yang benar yang memetakan ke set yang benar AZs dari cluster asli.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier original-cluster \
  --db-cluster-identifier clone-in-other-vpc \
```

```
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name subnet-group-in-other-vpc

{
  'DBClusterIdentifier': 'clone-in-other-vpc',
  'DBSubnetGroup': 'subnet-group-in-other-vpc',
  'Engine': 'aurora-postgresql',
  'EngineVersion': '15.4',
  'Status': 'creating',
  'Endpoint': 'clone-in-other-vpc.cluster-c0abcdef.us-east-1.rds.amazonaws.com'
}
```

Contoh berikut menegaskan bahwa penyimpanan Aurora di klon menggunakan set yang sama AZs seperti di cluster asli.

```
$ aws rds describe-db-clusters --db-cluster-identifier clone-in-other-vpc \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

Pada titik ini, Anda dapat membuat instance DB untuk klon. Pastikan bahwa grup VPC keamanan yang terkait dengan setiap instance memungkinkan koneksi dari rentang alamat IP yang Anda gunakan untuk EC2 instance, server aplikasi, dan sebagainya yang ada di tujuanVPC.

## Kloning lintas akun dengan AWS RAM dan Amazon Aurora

Dengan menggunakan AWS Resource Access Manager (AWS RAM) dengan Amazon Aurora, Anda dapat berbagi klaster Aurora DB dan klon milik akun Anda dengan AWS akun atau organisasi lain. AWS Kloning lintas akun lebih cepat daripada membuat dan memulihkan snapshot basis data. Anda dapat membuat klon dari salah satu klaster DB Aurora Anda lalu membagikan klon tersebut. Atau Anda dapat membagikan cluster Aurora DB Anda dengan AWS akun lain dan membiarkan pemegang akun membuat klon. Pendekatan yang Anda pilih tergantung pada kasus penggunaan Anda.

Misalnya, Anda mungkin perlu berbagi klon basis data keuangan Anda secara rutin dengan tim audit internal organisasi Anda. Dalam hal ini, tim audit Anda memiliki akun AWS untuk aplikasi yang digunakannya. Anda dapat memberikan izin kepada AWS akun tim audit untuk mengakses klaster Aurora DB Anda dan mengkloningnya sesuai kebutuhan.

Di sisi lain, jika vendor luar mengaudit data keuangan Anda, Anda mungkin lebih memilih untuk membuat klon sendiri. Anda kemudian memberi vendor luar tersebut akses ke klon saja.

Anda juga dapat menggunakan kloning lintas akun untuk mendukung banyak kasus penggunaan yang sama untuk kloning dalam AWS akun yang sama, seperti pengembangan dan pengujian. Misalnya, organisasi Anda mungkin menggunakan AWS akun yang berbeda untuk produksi, pengembangan, pengujian, dan sebagainya. Untuk informasi selengkapnya, lihat Gambaran umum kloning Aurora.

Dengan demikian, Anda mungkin ingin berbagi klon dengan AWS akun lain atau mengizinkan akun lain untuk membuat klon dari cluster Aurora DB Anda. AWS Dalam kedua kasus, mulailah dengan menggunakan AWS RAM untuk membuat objek berbagi. Untuk informasi lengkap tentang berbagi AWS sumber daya antar AWS akun, lihat Panduan AWS RAM Pengguna.

Membuat klon lintas akun memerlukan tindakan dari AWS akun yang memiliki cluster asli, dan AWS akun yang membuat klon. Pertama, pemilik klaster asli memodifikasi klaster untuk mengizinkan satu atau beberapa akun lain mengkloningnya. Jika salah satu akun berada di AWS organisasi yang berbeda AWS , buat undangan berbagi. Akun lain tersebut harus menerima undangan sebelum melanjutkan. Kemudian, setiap akun terotorisasi dapat mengkloning klaster. Selama proses ini, klaster diidentifikasi berdasarkan Amazon Resource Name (ARN) yang unik.

Seperti halnya kloning dalam AWS akun yang sama, ruang penyimpanan tambahan hanya digunakan jika perubahan dilakukan pada data oleh sumber atau klon. Biaya untuk penyimpanan kemudian diterapkan pada waktu itu. Jika klaster sumber dihapus, biaya penyimpanan didistribusikan secara merata di antara klon yang tersisa.

Topik

- Batasan kloning lintas akun
- Mengizinkan AWS akun lain untuk mengkloning klaster Anda
- Kloning cluster yang dimiliki oleh akun lain AWS

## Batasan kloning lintas akun

Kloning lintas akun Aurora memiliki batasan sebagai berikut:

- Anda tidak dapat mengkloning Aurora Serverless v1 cluster di seluruh AWS akun.

- Anda tidak dapat melihat atau menerima undangan ke sumber daya bersama dengan. AWS Management Console Gunakan API Amazon RDS, atau AWS RAM konsol untuk melihat dan menerima undangan ke sumber daya bersama. AWS CLI
- Anda hanya dapat membuat satu klon baru dari sumber daya yang dibagikan dengan Anda Akun AWS. Ini berlaku apakah sumber daya bersama adalah cluster Aurora DB asli atau klon yang dibuat sebelumnya.
- Anda hanya dapat membuat satu klon baru dari klon yang telah dibagikan dengan akun Anda AWS .
- Anda tidak dapat berbagi sumber daya (klon atau klaster Aurora DB) yang telah dibagikan dengan akun Anda. AWS
- Anda dapat membuat maksimum 15 klon lintas akun dari setiap klaster DB Aurora.
- Masing-masing dari 15 klon lintas akun harus dimiliki oleh akun yang berbeda AWS . Artinya, Anda hanya dapat membuat satu klon lintas akun dari sebuah cluster dalam akun apa pun AWS .
- Setelah Anda mengkloning klaster, klaster asli dan klonnya dianggap sama untuk tujuan memberlakukan batasan pada klon lintas akun. Anda tidak dapat membuat klon lintas akun dari klaster asli dan kloning kloning dalam akun yang sama. AWS Jumlah total klon lintas akun untuk klaster asli dan klonnya tidak boleh melebihi 15.
- Anda tidak dapat berbagi cluster Aurora DB dengan AWS akun lain kecuali klaster dalam keadaan. `ACTIVE`
- Anda tidak dapat mengganti nama cluster Aurora DB yang telah dibagikan dengan AWS akun lain.
- Anda tidak dapat membuat klon lintas akun dari klaster yang dienkripsikan dengan kunci RDS default.
- Anda tidak dapat membuat klon yang tidak terenkripsi dalam satu akun AWS dari kluster Aurora DB terenkripsi yang telah dibagikan oleh akun lain. AWS Pemilik klaster harus memberikan izin untuk mengakses AWS KMS key klaster sumber. Namun, Anda dapat menggunakan kunci yang berbeda saat Anda membuat klon.

## Mengizinkan AWS akun lain untuk mengkloning klaster Anda

Untuk mengizinkan AWS akun lain mengkloning klaster yang Anda miliki, gunakan AWS RAM untuk mengatur izin berbagi. Melakukannya juga mengirimkan undangan ke masing-masing akun lain yang ada di AWS organisasi yang berbeda.

Untuk prosedur berbagi sumber daya yang Anda miliki di AWS RAM konsol, lihat Berbagi sumber daya yang Anda miliki di Panduan AWS RAM Pengguna.

Topik


- Memberikan izin ke AWS akun lain untuk mengkloning klaster Anda
- Memeriksa apakah klaster yang Anda miliki dibagikan dengan AWS akun lain


## Memberikan izin ke AWS akun lain untuk mengkloning klaster Anda

Jika klaster yang Anda bagikan dienkripsi, Anda juga berbagi AWS KMS key untuk klaster tersebut. Anda dapat mengizinkan pengguna atau peran AWS Identity and Access Management (IAM) dalam satu AWS akun untuk menggunakan kunci KMS di akun yang berbeda.

Untuk melakukan ini, pertama-tama Anda menambahkan akun eksternal (pengguna root) ke kebijakan kunci kunci KMS melalui AWS KMS. Anda tidak menambahkan pengguna atau peran individual ke kebijakan kunci, hanya akun eksternal yang memilikinya. Anda hanya dapat berbagi kunci KMS yang Anda buat, bukan kunci layanan RDS default. Untuk informasi tentang kontrol akses untuk kunci KMS, lihat Autentikasi dan kontrol akses untuk AWS KMS.

Konsol

Untuk memberikan izin untuk mengkloning klaster Anda

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB yang ingin Anda bagikan untuk melihat halaman Detail, lalu pilih tab Konektivitas & keamanan.
4. Di bagian Share DB cluster with other AWS accounts, masukkan ID akun numerik untuk AWS akun yang ingin Anda izinkan untuk mengkloning cluster ini. Untuk akun IDs di organisasi yang sama, Anda dapat mulai mengetik di kotak dan kemudian memilih dari menu.

> **Important**
>
> Dalam beberapa kasus, Anda mungkin ingin akun yang tidak berada dalam organisasi AWS yang sama dengan akun Anda untuk mengkloning klaster. Dalam kasus ini, demi alasan keamanan, konsol tidak melaporkan siapa yang memiliki ID akun tersebut atau apakah akun tersebut ada.

Hati-hati memasukkan nomor akun yang tidak berada di AWS organisasi yang sama dengan AWS akun Anda. Segera verifikasi bahwa Anda berbagi kepada akun yang dimaksud.

5. Di halaman konfirmasi, verifikasi bahwa ID akun yang Anda tentukan sudah benar. Masukkan `share` di kotak konfirmasi untuk mengonfirmasi.

   Pada halaman Detail, muncul entri yang menunjukkan ID AWS akun yang ditentukan di bawah Akun tempat cluster DB ini dibagikan. Kolom Status awalnya menunjukkan status Tertunda.

6. Hubungi pemilik AWS akun lain, atau masuk ke akun itu jika Anda memiliki keduanya. Minta pemilik akun lain untuk menerima undangan berbagi dan mengkloning klaster DB, sebagaimana dijelaskan sebagai berikut.

AWS CLI

Untuk memberikan izin untuk mengkloning klaster Anda

1. Kumpulkan informasi untuk parameter yang diperlukan. Anda memerlukan ARN untuk cluster Anda dan ID numerik untuk akun lainnya. AWS
2. Jalankan AWS RAM perintah CLI. `create-resource-share`

   Untuk Linux, macOS, atau Unix:

```
aws ram create-resource-share --name descriptive_name \
  --region region \
  --resource-arns cluster_arn \
  --principals other_account_ids
```

   Untuk Windows:

```
aws ram create-resource-share --name descriptive_name ^
  --region region ^
  --resource-arns cluster_arn ^
  --principals other_account_ids
```

   Untuk memasukkan beberapa akun IDs untuk `--principals` parameter, pisahkan IDs satu sama lain dengan spasi. Untuk menentukan apakah akun yang diizinkan IDs dapat berada di

luar AWS organisasi Anda, sertakan `--allow-external-principals` atau `--no-allow-external-principals` parameter untuk`create-resource-share`.

AWS RAM API

Untuk memberikan izin untuk mengkloning klaster Anda

1. Kumpulkan informasi untuk parameter yang diperlukan. Anda memerlukan ARN untuk cluster Anda dan ID numerik untuk akun lainnya. AWS
2. Panggil operasi AWS RAM API CreateResourceShare, dan tentukan nilai berikut:
   - Tentukan ID akun untuk satu atau lebih AWS akun sebagai `principals` parameter.
   - Tentukan ARN untuk satu atau beberapa klaster DB Aurora sebagai parameter `resourceArns`.
   - Tentukan apakah akun yang diizinkan IDs dapat berada di luar AWS organisasi Anda dengan menyertakan nilai Boolean untuk `allowExternalPrincipals` parameter tersebut.

Membuat ulang klaster yang menggunakan kunci RDS default

Jika klaster terenkripsi yang ingin Anda bagikan menggunakan kunci RDS default, pastikan untuk membuat ulang klaster tersebut. Untuk melakukannya, buat snapshot manual dari klaster DB Anda, gunakan AWS KMS key, lalu pulihkan klaster tersebut ke klaster baru. Kemudian, bagikan klaster baru ini. Untuk melakukan proses ini, lakukan langkah-langkah berikut.

Untuk membuat ulang klaster terenkripsi yang menggunakan kunci RDS default

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Snapshot dari panel navigasi.
3. Pilih snapshot Anda.
4. Untuk Tindakan, pilih Salin Snapshot, lalu pilih Aktifkan enkripsi.
5. Untuk AWS KMS key, pilih kunci enkripsi baru yang ingin Anda gunakan.
6. Pulihkan snapshot yang telah disalin. Untuk melakukannya, ikuti prosedur dalam Memulihkan dari snapshot klaster DB. Instans DB baru akan menggunakan kunci enkripsi baru Anda.
7. (Opsional) Hapus klaster DB lama jika Anda tidak lagi membutuhkannya. Untuk melakukannya, ikuti prosedur dalam Menghapus snapshot klaster DB. Sebelum Anda melakukannya, konfirmasi

bahwa klaster baru Anda memiliki semua data yang diperlukan dan bahwa aplikasi Anda dapat mengaksesnya dengan sukses.

Memeriksa apakah klaster yang Anda miliki dibagikan dengan AWS akun lain

Anda dapat memeriksa apakah pengguna lain memiliki izin untuk berbagi klaster. Tindakan tersebut dapat membantu Anda memahami apakah klaster hampir mencapai batas jumlah maksimum klon lintas akun.

Untuk prosedur berbagi sumber daya menggunakan AWS RAM konsol, lihat Berbagi sumber daya yang Anda miliki di Panduan AWS RAM Pengguna.

AWS CLI

Untuk mengetahui apakah klaster yang Anda miliki dibagikan dengan AWS akun lain

- Panggil perintah AWS RAM CLI `list-principals`, menggunakan ID akun Anda sebagai pemilik sumber daya dan ARN cluster Anda sebagai ARN sumber daya. Anda dapat melihat semua pembagian dengan perintah berikut. Hasilnya menunjukkan AWS akun mana yang diizinkan untuk mengkloning cluster.

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

AWS RAM API

Untuk mengetahui apakah klaster yang Anda miliki dibagikan dengan AWS akun lain

- Panggil operasi AWS RAM API ListPrincipals. Gunakan ID akun Anda sebagai pemilik sumber daya dan ARN klaster sebagai ARN sumber daya.

## Kloning cluster yang dimiliki oleh akun lain AWS

Untuk mengkloning cluster yang dimiliki oleh AWS akun lain, gunakan AWS RAM untuk mendapatkan izin untuk membuat klon. Setelah Anda mendapatkan izin yang diperlukan, gunakan prosedur standar untuk mengkloning klaster Aurora.

Anda juga dapat memeriksa apakah cluster yang Anda miliki adalah tiruan dari cluster yang dimiliki oleh AWS akun yang berbeda.

Untuk prosedur untuk bekerja dengan sumber daya yang dimiliki oleh orang lain di AWS RAM konsol, lihat Mengakses sumber daya yang dibagikan dengan Anda di Panduan AWS RAM Pengguna.

Topik

- Melihat undangan untuk mengkloning cluster yang dimiliki oleh akun lain AWS
- Menerima undangan untuk berbagi cluster yang dimiliki oleh akun lain AWS
- Kloning cluster Aurora yang dimiliki oleh akun lain AWS
- Memeriksa apakah klaster DB merupakan klon lintas akun

### Melihat undangan untuk mengkloning cluster yang dimiliki oleh akun lain AWS

Untuk bekerja dengan undangan untuk mengkloning cluster yang dimiliki oleh AWS akun di AWS organisasi lain, gunakan, AWS RAM konsol AWS CLI, atau API. AWS RAM Saat ini, Anda tidak dapat melakukan prosedur ini menggunakan konsol Amazon RDS.

Untuk prosedur untuk bekerja dengan undangan di AWS RAM konsol, lihat Mengakses sumber daya yang dibagikan dengan Anda di Panduan Pengguna.AWS RAM

AWS CLI

Untuk melihat undangan untuk mengkloning cluster yang dimiliki oleh akun lain AWS

1. Jalankan AWS RAM perintah CLI. `get-resource-share-invitations`

   ```
   aws ram get-resource-share-invitations --region region_name
   ```

   Hasil dari perintah sebelumnya menunjukkan semua undangan untuk klaster klon, termasuk semua yang telah Anda terima atau ditolak.
2. (Opsional) Filter daftar sehingga Anda hanya melihat undangan yang memerlukan tindakan dari Anda. Untuk melakukannya, tambahkan parameter `--query 'resourceShareInvitations[?status==`PENDING`]'`.

AWS RAM API

Untuk melihat undangan untuk mengkloning cluster yang dimiliki oleh akun lain AWS

1. Panggil operasi AWS RAM API `GetResourceShareInvitations`. Operasi ini akan menampilkan semua undangan tersebut, termasuk semua yang telah Anda terima atau tolak.
2. (Opsional) Temukan hanya undangan yang memerlukan tindakan dari Anda dengan memeriksa `status` dengan nilai `PENDING` pada bidang `resourceShareAssociations` yang dihasilkan.

Menerima undangan untuk berbagi cluster yang dimiliki oleh akun lain AWS

Anda dapat menerima undangan untuk berbagi cluster yang dimiliki oleh AWS akun lain yang berada di organisasi yang berbeda. AWS Untuk bekerja dengan undangan ini, gunakan, AWS RAM dan RDS APIs, atau konsol. AWS CLI AWS RAM Saat ini, Anda tidak dapat melakukan prosedur ini menggunakan konsol RDS.

Untuk prosedur untuk bekerja dengan undangan di AWS RAM konsol, lihat Mengakses sumber daya yang dibagikan dengan Anda di Panduan Pengguna.AWS RAM

AWS CLI

Untuk menerima undangan untuk berbagi klaster dari AWS akun lain

1. Temukan ARN undangan dengan menjalankan perintah AWS RAM CLI `get-resource-share-invitations`, seperti yang ditunjukkan sebelumnya.
2. Terima undangan dengan memanggil perintah AWS RAM CLI `accept-resource-share-invitation`, seperti yang ditunjukkan berikut.

   Untuk Linux, macOS, atau Unix:

   ```
   aws ram accept-resource-share-invitation \
     --resource-share-invitation-arn invitation_arn \
     --region region
   ```

   Untuk Windows:

   ```
   aws ram accept-resource-share-invitation ^
     --resource-share-invitation-arn invitation_arn ^
     --region region
   ```

AWS RAM dan RDS API

Untuk menerima undangan untuk membagikan klaster orang lain

1. Temukan ARN undangan dengan memanggil operasi AWS RAM API `GetResourceShareInvitations`, seperti yang ditunjukkan sebelumnya.
2. Lewatkan ARN tersebut sebagai parameter `resourceShareInvitationArn` untuk operasi API RDS AcceptResourceShareInvitation.

Kloning cluster Aurora yang dimiliki oleh akun lain AWS

Setelah Anda menerima undangan dari AWS akun yang memiliki cluster DB, seperti yang ditunjukkan sebelumnya, Anda dapat mengkloning cluster.

Konsol

Untuk mengkloning cluster Aurora yang dimiliki oleh akun lain AWS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

   Di bagian atas daftar basis data, Anda akan melihat satu atau beberapa item dengan nilai Peran yang menampilkan `Shared from account #account_id`. Untuk alasan keamanan, Anda hanya dapat melihat informasi terbatas tentang klaster asli. Properti yang dapat Anda lihat antara lain adalah mesin dan versi basis data, yang harus sama dalam kloning klaster Anda.
3. Pilih klaster yang akan dikloning.
4. Untuk Tindakan, pilih Buat klon.
5. Ikuti prosedur dalam Konsol untuk menyelesaikan penyiapan klaster yang dikloning.
6. Sesuai kebutuhan, aktifkan enkripsi untuk klaster yang dikloning. Jika klaster tersebut dienkripsi, Anda harus mengaktifkan enkripsi untuk klaster yang dikloning. Akun AWS yang berbagi klaster dengan Anda juga harus berbagi kunci KMS yang digunakan untuk mengenkripsi klaster. Anda dapat menggunakan kunci KMS yang sama untuk mengenkripsi klon, atau kunci KMS Anda sendiri. Anda tidak dapat membuat klon lintas akun untuk klaster yang dienkripsi dengan kunci KMS default.

   Akun yang memiliki kunci enkripsi harus memberikan izin penggunaan kunci tersebut ke akun tujuan melalui kebijakan kunci. Proses ini serupa dengan cara berbagi snapshot

terenkripsi, dengan menggunakan kebijakan kunci yang memberikan izin ke akun tujuan untuk menggunakan kunci.

AWS CLI

Untuk mengkloning cluster Aurora yang dimiliki oleh akun lain AWS

1. Terima undangan dari AWS akun yang memiliki cluster DB, seperti yang ditunjukkan sebelumnya.
2. Kloning klaster dengan menentukan ARN lengkap klaster sumber dalam parameter `source-db-cluster-identifier` pada perintah CLI RDS `restore-db-cluster-to-point-in-time`, sebagaimana ditunjukkan berikut ini.

   Jika ARN yang diteruskan sebagai `source-db-cluster-identifier` belum dibagikan, kesalahan yang sama akan ditampilkan seolah-olah klaster tersebut tidak ada.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds restore-db-cluster-to-point-in-time \
     --source-db-cluster-identifier=arn:aws:rds:arn_details \
     --db-cluster-identifier=new_cluster_id \
     --restore-type=copy-on-write \
     --use-latest-restorable-time
   ```

   Untuk Windows:

   ```
   aws rds restore-db-cluster-to-point-in-time ^
     --source-db-cluster-identifier=arn:aws:rds:arn_details ^
     --db-cluster-identifier=new_cluster_id ^
     --restore-type=copy-on-write ^
     --use-latest-restorable-time
   ```

3. Jika klaster yang Anda kloning terenkripsi, enkripsi klaster yang dikloning dengan menyertakan parameter `kms-key-id`. Nilai `kms-key-id` ini dapat sama dengan yang digunakan untuk mengenkripsi klaster DB asli, atau kunci KMS Anda sendiri. Akun Anda harus memiliki izin untuk menggunakan kunci enkripsi tersebut.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds restore-db-cluster-to-point-in-time \
   ```

```
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time \
  --kms-key-id=arn:aws:kms:arn_details
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time ^
  --kms-key-id=arn:aws:kms:arn_details
```

Akun yang memiliki kunci enkripsi harus memberikan izin penggunaan kunci tersebut ke akun tujuan melalui kebijakan kunci. Proses ini serupa dengan cara berbagi snapshot terenkripsi, dengan menggunakan kebijakan kunci yang memberikan izin ke akun tujuan untuk menggunakan kunci. Contoh kebijakan kunci adalah sebagai berikut.

JSON

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
```

```
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

> **Note**
>
> Perintah restore-db-cluster-to- point-in-time AWS CLI hanya mengembalikan cluster DB, bukan instance DB untuk cluster DB itu. Untuk membuat instance DB untuk cluster DB yang dipulihkan, panggil perintah. create-db-instance Tentukan pengidentifikasi klaster DB yang dipulihkan di `--db-cluster-identifier`.
> Anda dapat membuat instans DB hanya setelah perintah `restore-db-cluster-to-point-in-time` selesai dan klaster DB tersedia.

API RDS

Untuk mengkloning cluster Aurora yang dimiliki oleh akun lain AWS

1. Terima undangan dari AWS akun yang memiliki cluster DB, seperti yang ditunjukkan sebelumnya.
2. Kloning klaster dengan menentukan ARN lengkah klaster sumber dalam parameter `SourceDBClusterIdentifier` pada operasi API RDS `RestoreDBClusterToPointInTime`.

Jika ARN yang diteruskan sebagai `SourceDBClusterIdentifier` belum dibagikan, kesalahan yang sama akan ditampilkan seolah-olah klaster tersebut tidak ada.

3. Jika klaster yang Anda kloning terenkripsi, sertakan parameter `KmsKeyId` untuk mengenkripsi klaster yang dikloning. Nilai `kms-key-id` ini dapat sama dengan yang digunakan untuk mengenkripsi klaster DB asli, atau kunci KMS Anda sendiri. Akun Anda harus memiliki izin untuk menggunakan kunci enkripsi tersebut.

   Saat mengkloning volume, akun tujuan harus memiliki izin untuk menggunakan kunci enkripsi yang digunakan untuk mengenkripsi klaster sumber. Aurora mengenkripsi klaster baru yang dikloning dengan kunci enkripsi yang ditentukan dalam `KmsKeyId`.

   Akun yang memiliki kunci enkripsi harus memberikan izin penggunaan kunci tersebut ke akun tujuan melalui kebijakan kunci. Proses ini serupa dengan cara berbagi snapshot terenkripsi, dengan menggunakan kebijakan kunci yang memberikan izin ke akun tujuan untuk menggunakan kunci. Contoh kebijakan kunci adalah sebagai berikut.

   JSON

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
```

```
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

> **Note**
>
> Operasi Restore DBCluster ToPointInTime RDS API hanya mengembalikan cluster DB, bukan instance DB untuk cluster DB tersebut. Untuk membuat instans DB untuk cluster DB yang dipulihkan, jalankan operasi Create DBInstance RDS API. Tentukan pengidentifikasi klaster DB yang dipulihkan di `DBClusterIdentifier`. Anda dapat membuat instans DB hanya setelah operasi `RestoreDBClusterToPointInTime` selesai dan klaster DB tersedia.

Memeriksa apakah klaster DB merupakan klon lintas akun

Objek `DBClusters` mengidentifikasi apakah setiap klaster merupakan klon lintas akun. Anda dapat melihat klaster yang izin kloning Anda miliki dengan menggunakan opsi `include-shared` saat Anda menjalankan perintah CLI RDS `describe-db-clusters`. Namun, Anda tidak dapat melihat sebagian besar detail konfigurasi untuk klaster tersebut.

AWS CLI

Untuk memeriksa apakah klaster DB merupakan klon lintas akun

- Panggil perintah CLI RDS `describe-db-clusters`.

Contoh berikut menunjukkan bagaimana klaster DB klon lintas akun yang sebenarnya atau potensial muncul di output `describe-db-clusters`. Untuk klaster yang ada yang dimiliki oleh AWS akun Anda, `CrossAccountClone` bidang menunjukkan apakah klaster adalah tiruan dari klaster DB yang dimiliki oleh akun lain AWS .

Dalam beberapa kasus, entri mungkin memiliki nomor AWS akun yang berbeda dari milik Anda di `DBClusterArn` lapangan. Dalam hal ini, entri itu mewakili cluster yang dimiliki oleh AWS akun yang berbeda dan Anda dapat mengkloning. Entri tersebut memiliki beberapa bidang selain `DBClusterArn`. Saat membuat klaster yang diklon, tentukan nilai `StorageEncrypted`, `Engine`, dan `EngineVersion` yang sama seperti dalam klaster asli.

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0
  ]
}
```

API RDS

Untuk memeriksa apakah klaster DB merupakan klon lintas akun

- Panggil operasi API RDS Jelaskan DBClusters.

  Untuk klaster yang ada yang dimiliki oleh AWS akun Anda, `CrossAccountClone` bidang menunjukkan apakah klaster adalah tiruan dari klaster DB yang dimiliki oleh akun lain AWS . Entri dengan nomor AWS akun yang berbeda di `DBClusterArn` bidang mewakili cluster yang dapat Anda kloning dan yang dimiliki oleh akun lain. AWS Entri tersebut memiliki beberapa bidang selain `DBClusterArn`. Saat membuat klaster yang diklon, tentukan nilai `StorageEncrypted`, `Engine`, dan `EngineVersion` yang sama seperti dalam klaster asli.

  Contoh berikut menunjukkan nilai yang dihasilkan yang menunjukkan klaster yang dikloning yang sebenarnya dan potensial.

```
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0"
      }
  ]
}
```

# Mengintegrasikan Aurora dengan layanan lain AWS

Integrasikan Amazon Aurora dengan AWS layanan lain sehingga Anda dapat memperluas cluster Aurora DB Anda untuk menggunakan kemampuan tambahan di Cloud. AWS

Topik


- Mengintegrasikan AWS layanan dengan Amazon Aurora My SQL
- Mengintegrasikan AWS layanan dengan Amazon Aurora Postgre SQL


## Mengintegrasikan AWS layanan dengan Amazon Aurora My SQL

Amazon Aurora My SQL terintegrasi dengan AWS layanan lain sehingga Anda dapat memperluas cluster Aurora My SQL DB Anda untuk menggunakan kemampuan tambahan di Cloud. AWS Cluster Aurora My SQL DB Anda dapat menggunakan AWS layanan untuk melakukan hal berikut:

- Secara sinkron atau asinkron memanggil AWS Lambda fungsi menggunakan fungsi asli atau. `lambda_sync` `lambda_async` Atau, secara asinkron menginvokasi fungsi AWS Lambda menggunakan prosedur `mysql.lambda_async`.
- Muat data dari teks atau XML file yang disimpan dalam bucket Amazon S3 ke dalam cluster DB Anda menggunakan perintah `LOAD DATA FROM S3` or`LOAD XML FROM S3`.
- Menyimpan data ke file teks yang disimpan dalam bucket Amazon S3 dari klaster DB Anda menggunakan perintah `SELECT INTO OUTFILE S3`.
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Application Auto Scaling. Untuk informasi selengkapnya, lihat Auto Scaling Amazon Aurora dengan Replika Aurora.

Untuk informasi selengkapnya tentang mengintegrasikan Aurora SQL My dengan layanan AWS lain, lihat. Mengintegrasikan Amazon Aurora SQL My dengan layanan lain AWS

## Mengintegrasikan AWS layanan dengan Amazon Aurora Postgre SQL

Amazon Aurora Postgre SQL terintegrasi dengan AWS layanan lain sehingga Anda dapat memperluas cluster Aurora Postgre SQL DB Anda untuk menggunakan kemampuan tambahan di Cloud. AWS Cluster Aurora Postgre SQL DB Anda dapat menggunakan AWS layanan untuk melakukan hal berikut:

- Mengumpulkan, melihat, dan menilai performa dengan cepat pada beban kerja basis data relasional Anda menggunakan Wawasan Performa.
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Aurora Auto Scaling. Untuk informasi selengkapnya, lihat Auto Scaling Amazon Aurora dengan Replika Aurora.

Untuk informasi lebih lanjut tentang mengintegrasikan Aurora SQL Postgre dengan AWS layanan lain, lihat. Mengintegrasikan Amazon Aurora SQL Postgre dengan layanan lain AWS

# Memelihara klaster DB Amazon Aurora

Amazon RDS melakukan pemeliharaan secara berkala pada sumber daya Amazon RDS. Topik berikut menjelaskan tindakan pemeliharaan ini dan cara menerapkannya.

## Ikhtisar pembaruan pemeliharaan cluster DB

Pemeliharaan sering kali melibatkan pembaruan ke sumber daya berikut di klaster DB:

- Perangkat keras yang mendasarinya
- Sistem operasi yang mendasarinya (OS)
- Versi mesin basis data

Pembaruan pada sistem operasi paling sering terjadi untuk masalah keamanan. Kami menyarankan Anda melakukannya sesegera mungkin. Untuk informasi selengkapnya tentang pembaruan sistem operasi, lihat .

Topik

- Sumber daya offline selama pembaruan pemeliharaan
- Modifikasi instans DB dan cluster DB yang ditangguhkan
- Konsistensi akhir untuk API DescribePendingMaintenanceActions

### Sumber daya offline selama pembaruan pemeliharaan

Beberapa item pemeliharaan mengharuskan Amazon RDS membuat klaster DB Anda offline selama waktu yang singkat. Item pemeliharaan yang mengharuskan sumber daya untuk offline mencakup patching sistem operasi atau basis data yang diperlukan. Patching yang diperlukan secara otomatis dijadwalkan hanya untuk patch yang terkait dengan keamanan dan keandalan instans. Patching tersebut jarang terjadi, biasanya sekali setiap beberapa bulan. Ini jarang membutuhkan lebih dari periode pemeliharaan Anda.

### Modifikasi instans DB dan cluster DB yang ditangguhkan

Kluster DB yang ditangguhkan dan modifikasi instance yang telah Anda pilih untuk tidak segera diterapkan diterapkan selama jendela pemeliharaan. Misalnya, Anda dapat memilih untuk mengubah kelas atau klaster instans DB atau grup parameter DB selama periode pemeliharaan. Modifikasi seperti yang Anda tentukan menggunakan pengaturan boot ulang tertunda tidak muncul dalam

daftar Pemeliharaan tertunda. Untuk informasi tentang cara mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Untuk melihat modifikasi yang tertunda untuk jendela pemeliharaan berikutnya, gunakan describe-db-clusters AWS CLI perintah dan periksa `PendingModifiedValues` bidangnya.

### Konsistensi akhir untuk API DescribePendingMaintenanceActions

Amazon RDS `DescribePendingMaintenanceActions` API mengikuti model konsistensi akhirnya. Ini berarti bahwa hasil dari `DescribePendingMaintenanceActions` perintah mungkin tidak langsung terlihat oleh semua perintah RDS berikutnya. Ingatlah hal ini saat Anda menggunakan `DescribePendingMaintenanceActions` segera setelah menggunakan perintah API sebelumnya.

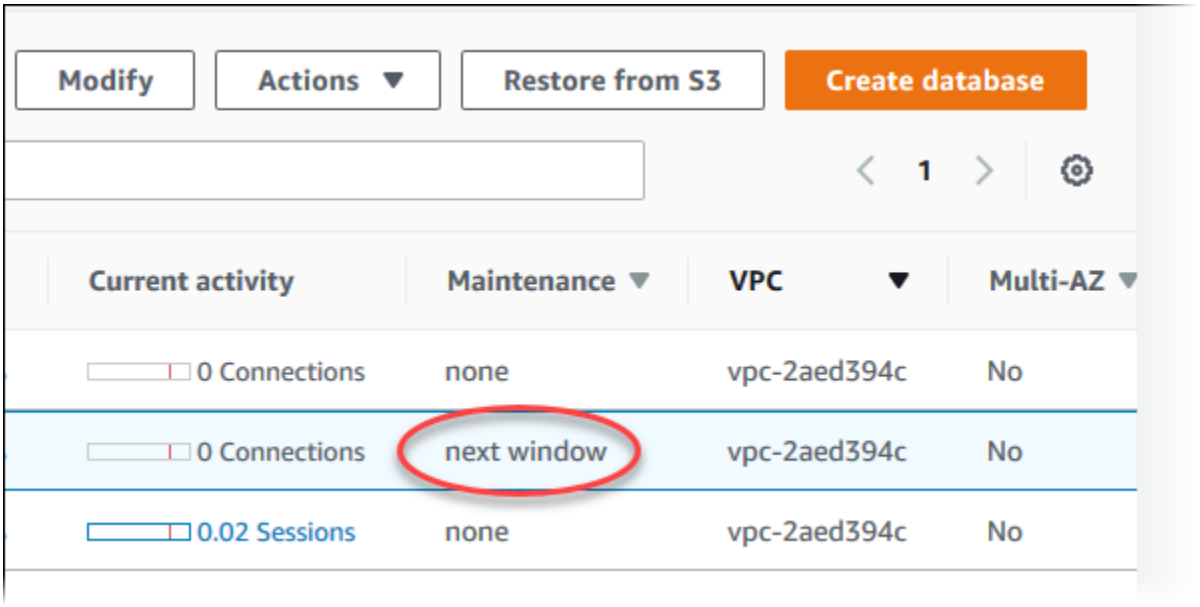
Konsistensi akhirnya dapat memengaruhi cara Anda mengelola pembaruan pemeliharaan. Misalnya, jika Anda menjalankan `ApplyPendingMaintenanceActions` perintah untuk memperbarui versi mesin database untuk cluster DB, pada akhirnya akan terlihat`DescribePendingMaintenanceActions`. Dalam skenario ini, `DescribePendingMaintenanceActions` mungkin menunjukkan bahwa tindakan pemeliharaan tidak diterapkan meskipun itu.

Untuk mengelola konsistensi akhirnya, Anda dapat melakukan hal berikut:

- Konfirmasikan status cluster DB Anda sebelum Anda menjalankan perintah untuk memodifikasinya. Jalankan `DescribePendingMaintenanceActions` perintah yang sesuai menggunakan algoritma backoff eksponensial untuk memastikan bahwa Anda memberikan cukup waktu untuk perintah sebelumnya untuk menyebar melalui sistem. Untuk melakukan ini, jalankan `DescribePendingMaintenanceActions` perintah berulang kali, dimulai dengan beberapa detik waktu tunggu, dan meningkat secara bertahap hingga lima menit waktu tunggu.
- Tambahkan waktu tunggu antara perintah berikutnya, bahkan jika `DescribePendingMaintenanceActions` perintah mengembalikan respons yang akurat. Terapkan algoritma backoff eksponensial dimulai dengan beberapa detik waktu tunggu, dan tingkatkan secara bertahap hingga sekitar lima menit waktu tunggu.

## Melihat pembaruan pemeliharaan yang tertunda

Lihat apakah pembaruan pemeliharaan tersedia untuk cluster DB Anda dengan menggunakan konsol RDS, API AWS CLI, atau RDS. Jika pembaruan tersedia, ini ditunjukkan di kolom Pemeliharaan untuk cluster DB di konsol Amazon RDS, seperti yang ditunjukkan pada gambar ini.




Jika pembaruan pemeliharaan tidak tersedia untuk klaster DB, nilai kolomnya adalah tidak ada.

Jika pembaruan pemeliharaan tersedia untuk klaster DB, kemungkinan nilai kolomnya adalah sebagai berikut:

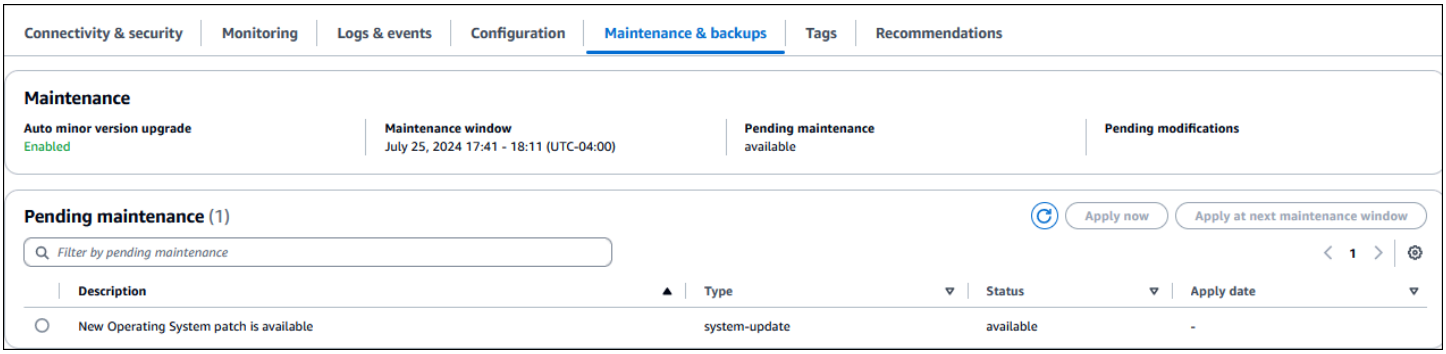
- diperlukan – Tindakan pemeliharaan akan diterapkan ke sumber daya dan tidak dapat ditunda tanpa batas waktu.
- tersedia – Tindakan pemeliharaan tersedia, tetapi tidak akan diterapkan ke sumber daya secara otomatis. Anda dapat menerapkannya secara manual.
- periode berikutnya – Tindakan pemeliharaan akan diterapkan ke sumber daya pada periode pemeliharaan berikutnya.
- Sedang berlangsung — Tindakan pemeliharaan sedang diterapkan pada sumber daya.

Jika pembaruan tersedia, Anda dapat melakukan salah satu hal berikut:

- Jika nilai pemeliharaan adalah jendela berikutnya, tunda tindakan pemeliharaan dengan memilih Defer upgrade dari Actions. Anda tidak dapat menunda tindakan pemeliharaan yang sudah dimulai.
- Terapkan tindakan pemeliharaan segera.
- Terapkan tindakan pemeliharaan selama jendela pemeliharaan Anda berikutnya.
- Tidak melakukan tindakan apa pun.

Untuk mengambil tindakan dengan menggunakan AWS Management Console

1. Pilih instans atau cluster DB untuk menampilkan detailnya.
2. Pilih Maintenance & backup. Tindakan pemeliharaan yang tertunda muncul.
3. Pilih tindakan yang akan diambil, lalu pilih kapan harus menerapkannya.




Periode pemeliharaan menentukan kapan operasi yang tertunda dimulai, tetapi tidak membatasi total waktu eksekusi operasi ini. Operasi pemeliharaan tidak dijamin selesai sebelum periode pemeliharaan berakhir, dan dapat berlanjut melebihi waktu akhir yang ditentukan. Untuk informasi selengkapnya, lihat Jendela pemeliharaan Amazon RDS.

Anda juga dapat melihat apakah pembaruan pemeliharaan tersedia untuk cluster DB Anda dengan menjalankan describe-pending-maintenance-actions AWS CLI perintah.

Untuk informasi tentang menerapkan pembaruan pemeliharaan, lihatMenerapkan pembaruan ke cluster DB.

## Tindakan pemeliharaan untuk Amazon Aurora

Tindakan pemeliharaan berikut berlaku untuk cluster Aurora DB:

- `os-upgrade`— Perbarui sistem operasi semua instans DB di cluster DB, menggunakan peningkatan bergulir. Untuk informasi selengkapnya, lihat the section called “Pembaruan sistem operasi”.
- `system-update`— Menambal mesin DB untuk Aurora PostgreSQL.

Tindakan pemeliharaan berikut berlaku untuk instans Aurora DB:

- `ca-certificate-rotation`— Perbarui sertifikat Otoritas Sertifikat Amazon RDS untuk instans DB.

- `hardware-maintenance`— Lakukan pemeliharaan pada perangkat keras yang mendasarinya untuk instans DB.
- `system-update`— Perbarui sistem operasi untuk instans DB.

## Memilih frekuensi pembaruan pemeliharaan Aurora MySQL

Anda dapat mengendalikan apakah peningkatan Aurora MySQL sering atau jarang terjadi untuk setiap klaster DB. Pilihan terbaiknya bergantung pada penggunaan Aurora MySQL Anda dan prioritas untuk aplikasi yang berjalan di Aurora. Untuk informasi tentang rilis stabilitas jangka panjang (LTS) Aurora MySQL yang membutuhkan peningkatan yang jarang, lihat Aurora Dukungan SQL jangka panjang saya () rilis LTS.

Anda dapat memilih untuk jarang meningkatkan klaster Aurora MySQL jika sebagian atau semua kondisi berikut berlaku:

- Siklus pengujian untuk aplikasi Anda membutuhkan waktu yang lama untuk setiap pembaruan ke mesin basis data Aurora MySQL.
- Anda memiliki banyak klaster DB atau banyak aplikasi yang semuanya berjalan di versi Aurora MySQL yang sama. Anda lebih suka meningkatkan semua klaster DB dan aplikasi yang terkait secara bersamaan.
- Anda menggunakan Aurora MySQL dan RDS for MySQL. Anda lebih suka untuk tetap menggunakan klaster Aurora MySQL dan instans DB RDS for MySQL yang kompatibel dengan tingkat MySQL yang sama.
- Aplikasi Aurora MySQL Anda sedang dalam produksi atau penting untuk bisnis. Anda tidak dapat memberikan waktu henti untuk peningkatan di luar peristiwa langka untuk patch penting.
- Aplikasi Aurora MySQL Anda tidak dibatasi oleh masalah performa atau celah fitur yang ditangani di versi Aurora MySQL selanjutnya.

Jika faktor-faktor sebelumnya terjadi pada situasi Anda, Anda dapat membatasi jumlah peningkatan paksa untuk klaster DB Aurora MySQL. Anda dapat melakukannya dengan memilih versi Aurora MySQL spesifik yang dikenal sebagai versi "Dukungan Jangka Panjang" (LTS) saat Anda membuat atau meningkatkan klaster DB tersebut. Tindakan ini dapat meminimalkan jumlah siklus peningkatan, siklus pengujian, dan pemadaman terkait peningkatan untuk klaster DB tersebut.

Anda dapat memilih untuk jarang meningkatkan klaster Aurora MySQL jika sebagian atau semua kondisi berikut berlaku:

- Siklus pengujian untuk aplikasi Anda itu mudah dan singkat.
- Aplikasi Anda masih dalam tahap pengembangan.
- Lingkungan basis data Anda menggunakan berbagai versi Aurora MySQL, atau versi Aurora MySQL dan RDS for MySQL. Setiap klaster Aurora MySQL memiliki siklus peningkatan-nya sendiri.
- Anda menunggu peningkatan performa atau peningkatan fitur spesifik sebelum Anda meningkatkan penggunaan Aurora MySQL.

Jika faktor-faktor sebelumnya terjadi pada situasi Anda, Anda dapat mengaktifkan Aurora untuk lebih sering menerapkan peningkatan penting. Untuk melakukannya, tingkatkan klaster DB Aurora MySQL ke Aurora MySQL versi yang lebih baru dari versi LTS. Dengan begitu, peningkatan performa, perbaikan bug, dan fitur-fitur terbaru akan tersedia bagi Anda dengan lebih cepat.

## Jendela pemeliharaan Amazon RDS

Jendela pemeliharaan adalah interval waktu mingguan di mana setiap perubahan sistem diterapkan. Setiap cluster DB memiliki jendela pemeliharaan mingguan. Jendela pemeliharaan adalah kesempatan untuk mengontrol kapan modifikasi dan penambalan perangkat lunak terjadi. Untuk informasi selengkapnya tentang menyesuaikan jendela pemeliharaan, lihat .

RDS menggunakan beberapa sumber daya di klaster DB Anda saat pemeliharaan diterapkan. Anda mungkin mendapati efek minimal pada performa. Untuk instans DB, dalam situasi yang jarang terjadi, failover Multi-AZ mungkin diperlukan untuk menyelesaikan pembaruan pemeliharaan.

Jika peristiwa pemeliharaan dijadwalkan selama satu minggu, ini akan dimulai selama periode pemeliharaan 30 menit yang Anda identifikasi. Sebagian besar peristiwa pemeliharaan juga selesai selama periode pemeliharaan 30 menit, meskipun peristiwa pemeliharaan yang lebih besar bisa memakan waktu lebih dari 30 menit. Jendela pemeliharaan dijeda ketika cluster DB dihentikan.

Periode pemeliharaan 30 menit dipilih secara acak dari blok waktu 8 jam per wilayah. Jika Anda tidak menentukan periode pemeliharaan saat membuat klaster DB, RDS akan menetapkan periode pemeliharaan 30 menit pada hari yang dipilih secara acak dalam seminggu.

Tabel berikut menunjukkan blok waktu untuk masing-masing Wilayah AWS dari mana jendela pemeliharaan default ditetapkan.

| Nama Wilayah | Wilayah | Blok Waktu |
|---|---|---|
| US East (Northern Virginia) | us-east-1 | 03.00–11.00 UTC |
| AS Timur (Ohio) | us-east-2 | 03.00–11.00 UTC |
| AS Barat (California Utara) | us-west-1 | 06.00–14.00 UTC |
| AS Barat (Oregon) | us-west-2 | 06.00–14.00 UTC |
| Afrika (Cape Town) | af-south-1 | 03.00–11.00 UTC |
| Asia Pasifik (Hong Kong) | ap-east-1 | 06.00–14.00 UTC |
| Asia Pasifik (Hyderabad) | ap-south-2 | 06.30–14.30 UTC |
| Asia Pasifik (Jakarta) | ap-southeast-3 | 08.00–16.00 UTC |
| Asia Pasifik (Malaysia) | ap-southeast-5 | 09:00 — 17:00 UTC |
| Asia Pasifik (Melbourne) | ap-southeast-4 | 11.00–19.00 UTC |
| Asia Pasifik (Mumbai) | ap-south-1 | 06.00–14.00 UTC |
| Asia Pasifik (Osaka) | ap-northeast-3 | 22.00–23.59 UTC |
| Asia Pasifik (Seoul) | ap-northeast-2 | 13.00–21.00 UTC |
| Asia Pasifik (Singapura) | ap-southeast-1 | 14.00–22.00 UTC |
| Asia Pasifik (Sydney) | ap-southeast-2 | 12.00–20.00 UTC |
| Asia Pasifik (Tokyo) | ap-northeast-1 | 13.00–21.00 UTC |

| Nama Wilayah | Wilayah | Blok Waktu |
|---|---|---|
| Kanada (Pusat) | ca-central-1 | 03.00–11.00 UTC |
| Kanada Barat (Calgary) | ca-west-1 | 18:00 — 02:00 UTC |
| Tiongkok (Beijing) | cn-north-1 | 06.00–14.00 UTC |
| Tiongkok (Ningxia) | cn-northwest-1 | 06.00–14.00 UTC |
| Eropa (Frankfurt) | eu-central-1 | 21.00–05.00 UTC |
| Eropa (Irlandia) | eu-west-1 | 22.00–06.00 UTC |
| Eropa (London) | eu-west-2 | 22.00–06.00 UTC |
| Eropa (Milan) | eu-south-1 | 02.00–10.00 UTC |
| Eropa (Paris) | eu-west-3 | 23.59–07.29 UTC |
| Eropa (Spanyol) | eu-south-2 | 02.00–10.00 UTC |
| Eropa (Stockholm) | eu-north-1 | 23.00–07.00 UTC |
| Eropa (Zürich) | eu-central-2 | 02.00–10.00 UTC |
| Israel (Tel Aviv) | il-central-1 | 03.00–11.00 UTC |
| Timur Tengah (Bahrain) | me-south-1 | 06.00–14.00 UTC |
| Timur Tengah (UEA) | me-central-1 | 05.00–13.00 UTC |
| Amerika Selatan (Sao Paulo) | sa-east-1 | 00.00–08.00 UTC |
| AWS GovCloud (AS-Timur) | us-gov-east-1 | 17.00–01.00 UTC |

| Nama Wilayah | Wilayah | Blok Waktu |
| --- | --- | --- |
| AWS GovCloud (AS-Barat) | us-gov-west-1 | 06.00–14.00 UTC |

Topik

- Menyesuaikan periode pemeliharaan klaster DB yang dinginkan

## Menyesuaikan periode pemeliharaan klaster DB yang dinginkan

Periode pemeliharaan klaster DB Aurora harus berada dalam waktu penggunaan terendah, sehingga kemungkinan memerlukan perubahan dari waktu ke waktu. klaster DB Anda tidak tersedia selama waktu ini hanya jika pembaruan yang diterapkan memerlukan pemadaman. Pemadaman adalah untuk waktu minimal yang diperlukan untuk melakukan pembaruan yang diperlukan.

> **Note**
>
> Untuk peningkatan ke mesin basis data, Amazon Aurora mengelola periode pemeliharaan yang dinginkan untuk klaster DB dan bukan instans individu.

Konsol

Untuk menyesuaikan periode pemeliharaan klaster DB yang dinginkan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB yang periode pemeliharaannya ingin diubah.
4. Pilih Ubah.
5. Di bagian Pemeliharaan, perbarui periode pemeliharaan.
6. Pilih Lanjutkan.

   Di halaman konfirmasi, tinjau perubahan Anda.
7. Untuk menerapkan perubahan pada periode pemeliharaan secara langsung, pilih Segera di bagian Penjadwalan perubahan.

8. Pilih Ubah klaster untuk menyimpan perubahan Anda.

   Atau, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan.

### AWS CLI

Untuk menyesuaikan jendela pemeliharaan cluster DB yang disukai, gunakan AWS CLI `modify-db-cluster`perintah dengan parameter berikut:

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

### Example

Contoh kode berikut mengatur periode pemeliharaan ke Selasa mulai pukul 04.00-04.30 UTC.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

Untuk Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

### RDS API

Untuk menyesuaikan periode pemeliharaan klaster DB yang diinginkan, gunakan operasi Amazon RDS API `ModifyDBCluster` dengan parameter berikut:

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

# Menerapkan pembaruan ke cluster DB

Dengan Amazon RDS, Anda dapat memilih waktu untuk menerapkan operasi pemeliharaan. Anda dapat memutuskan kapan Amazon RDS menerapkan pembaruan dengan menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Untuk mengelola pembaruan untuk instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB yang memiliki pembaruan yang diperlukan.
4. Untuk Tindakan, pilih salah satu opsi berikut:
   - Patch sekarang
   - Patch di jendela berikutnya

   > **Note**
   >
   > Jika Anda memilih Patch di jendela berikutnya dan kemudian ingin menunda pembaruan, Anda dapat memilih Tangguhkan peningkatan. Anda tidak dapat menunda tindakan pemeliharaan jika sudah dimulai.
   > Untuk membatalkan tindakan pemeliharaan, ubah instans DB dan nonaktifkan Peningkatan versi minor otomatis.

AWS CLI

Untuk menerapkan pembaruan yang tertunda ke cluster DB, gunakan apply-pending-maintenance-action AWS CLI perintah.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
```

```
    --apply-action system-update \
    --opt-in-type immediate
```

Untuk Windows:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> Untuk menunda tindakan pemeliharaan, tentukan `undo-opt-in` untuk `--opt-in-type`. Anda tidak dapat menentukan `undo-opt-in` untuk `--opt-in-type` jika tindakan pemeliharaan sudah dimulai.
>
> Untuk membatalkan tindakan pemeliharaan, jalankan modify-db-instance AWS CLI perintah dan tentukan`--no-auto-minor-version-upgrade`.

Untuk mengembalikan daftar sumber daya yang memiliki setidaknya satu pembaruan yang tertunda, gunakan describe-pending-maintenance-actions AWS CLI perintah.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Untuk Windows:

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Anda juga dapat mengembalikan daftar sumber daya untuk cluster DB dengan menentukan `--filters` parameter `describe-pending-maintenance-actions` AWS CLI perintah. Format untuk perintah `--filters` adalah `Name=`*`filter-name`*`,Value=`*`resource-id`*`,...`.

Berikut adalah nilai yang diterima untuk parameter `Name` dari filter:

- `db-instance-id`— Menerima daftar pengidentifikasi instans DB atau Nama Sumber Daya Amazon ()ARNs. Daftar yang dikembalikan hanya mencakup tindakan pemeliharaan yang tertunda untuk instans DB yang diidentifikasi oleh pengidentifikasi ini atau ARNs.
- `db-cluster-id`— Menerima daftar pengidentifikasi cluster DB atau ARNs untuk Amazon Aurora. Daftar yang dikembalikan hanya mencakup tindakan pemeliharaan yang tertunda untuk klaster DB yang diidentifikasi oleh pengidentifikasi ini atau ARNs.

Misalnya, contoh berikut menampilkan tindakan pemeliharaan yang tertunda untuk klaster DB `sample-cluster1` dan `sample-cluster2`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Untuk Windows:

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

Untuk menerapkan pembaruan ke klaster DB, panggil operasi `ApplyPendingMaintenanceAction` Amazon RDS API.

Untuk menampilkan daftar sumber daya yang memiliki setidaknya satu pembaruan tertunda, panggil operasi `DescribePendingMaintenanceActions` Amazon RDS API.

## Peningkatan versi minor otomatis untuk klaster DB Aurora

Pengaturan Peningkatan versi minor otomatis menentukan apakah Aurora secara otomatis menerapkan peningkatan ke klaster DB Anda. Peningkatan ini mencakup versi minor baru yang berisi fitur tambahan dan patch yang berisi perbaikan bug.

Upgrade versi minor otomatis memperbarui database Anda secara berkala ke versi mesin database terbaru. Namun, upgrade mungkin tidak selalu menyertakan versi mesin database terbaru. Jika

Anda perlu menyimpan database Anda pada versi tertentu pada waktu tertentu, kami sarankan Anda meningkatkan secara manual ke versi database yang Anda butuhkan sesuai dengan jadwal yang Anda butuhkan. Dalam kasus masalah keamanan kritis atau ketika versi mencapai end-of-support tanggalnya, Aurora mungkin menerapkan pemutakhiran versi minor meskipun Anda belum mengaktifkan opsi pemutakhiran versi minor Otomatis. Untuk informasi selengkapnya, lihat dokumentasi pemutakhiran untuk mesin database spesifik Anda.

Lihat Memutakhirkan versi minor atau level patch cluster Aurora SQL My DB dan Melakukan upgrade versi minor.

> **Note**
>
> Aurora Global Database tidak mendukung peningkatan versi minor otomatis.

Pengaturan ini diaktifkan secara default. Untuk setiap klaster DB baru, pilih nilai yang sesuai untuk pengaturan ini. Nilai ini didasarkan pada kepentingannya, masa pakai yang diharapkan, dan jumlah pengujian verifikasi yang Anda lakukan setelah setiap peningkatan.

Untuk petunjuk tentang cara mengaktifkan atau menonaktifkan pengaturan Peningkatan versi minor otomatis, lihat referensi berikut:

- Mengaktifkan peningkatan versi minor otomatis untuk klaster DB Aurora
- Mengaktifkan peningkatan versi minor otomatis untuk instans DB individu dalam klaster DB Aurora

> **Important**
>
> Untuk klaster DB baru dan yang sudah ada, sebaiknya Anda menerapkan pengaturan ini ke klaster DB, bukan ke instans DB di klaster satu per satu. Jika ada instans DB di klaster Anda yang menonaktifkan pengaturan ini, klaster DB tersebut tidak akan ditingkatkan secara otomatis.

Tabel berikut menunjukkan cara kerja pengaturan Peningkatan versi minor otomatis saat diterapkan di tingkat klaster dan instans.

| Tindakan | Pengaturan klaster | Pengaturan instans | Klaster ditingkatkan ditingkatkan secara otomatis? |
|---|---|---|---|
| Anda mengaturnya ke True di klaster DB. | True | True untuk semua instans baru dan yang sudah ada | Ya |
| Anda mengaturnya ke False di klaster DB. | False | False untuk semua instans baru dan yang sudah ada | Tidak |
| Sebelumnya diatur ke True di klaster DB.<br><br>Anda mengaturnya ke False setidaknya di satu instans DB. | Perubahan ke False | False untuk satu atau beberapa instans | Tidak |
| Itu sebelum ditetapkan ke False pada klaster DB.<br><br>Anda mengaturnya ke True setidaknya di satu instans DB, tetapi tidak semua instans. | False | True untuk satu atau beberapa instans, tetapi tidak semua instans | Tidak |
| Itu sebelum ditetapkan ke False pada klaster DB.<br><br>Anda mengaturnya ke True di semua instans DB. | Perubahan ke True | True untuk semua instans | Ya |

Peningkatan versi minor otomatis dikomunikasikan terlebih dahulu melalui peristiwa klaster DB Amazon RDS dengan kategori `maintenance` dan ID `RDS-EVENT-0156`. Untuk informasi selengkapnya, lihat Kategori acara Amazon RDS dan pesan acara untuk Aurora.

Peningkatan otomatis terjadi selama periode pemeliharaan. Jika instans DB individu di klaster DB memiliki periode pemeliharaan yang berbeda dengan periode pemeliharaan klaster, periode pemeliharaan klaster akan diutamakan.

Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora PostgreSQL, lihat Pembaruan mesin database untuk Amazon Aurora Postgre SQL.

Untuk informasi selengkapnya tentang pengaturan Peningkatan versi minor otomatis untuk Aurora MySQL, lihat Mengaktifkan upgrade otomatis antara Aurora minor versi Saya SQL. Untuk informasi umum tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin database untuk Amazon Aurora My SQL.

Mengaktifkan peningkatan versi minor otomatis untuk klaster DB Aurora

Ikuti prosedur umum dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

Konsol

Di halaman Mengubah klaster DB, di bagian Pemeliharaan, pilih kotak centang Aktifkan peningkatan versi minor otomatis.

AWS CLI

Panggil modify-db-cluster AWS CLI perintahnya. Tentukan nama klaster DB untuk opsi `--db-cluster-identifier` dan `true` untuk opsi `--auto-minor-version-upgrade`. Secara opsional, tentukan opsi `--apply-immediately` untuk segera mengaktifkan pengaturan ini untuk klaster DB Anda.

API RDS

Panggil operasi Modify DBCluster API dan tentukan nama cluster DB Anda `true` untuk `DBClusterIdentifier` parameter dan `AutoMinorVersionUpgrade` parameter. Atau, atur parameter `ApplyImmediately` untuk `true` untuk segera mengaktifkan pengaturan ini untuk klaster DB Anda.

Mengaktifkan peningkatan versi minor otomatis untuk instans DB individu dalam klaster DB Aurora

Ikuti prosedur umum dalam Memodifikasi instans DB dalam klaster DB.

Konsol

Di halaman Mengubah instans DB, di bagian Pemeliharaan, pilih kotak centang Aktifkan peningkatan versi minor otomatis.

AWS CLI

Panggil modify-db-instance AWS CLI perintahnya. Tentukan nama instans DB Anda untuk opsi `--db-instance-identifier` dan `true` untuk opsi `--auto-minor-version-upgrade`. Secara opsional, tentukan opsi `--apply-immediately` untuk segera mengaktifkan pengaturan ini untuk instans DB Anda. Jalankan perintah `modify-db-instance` terpisah untuk setiap instans DB di klaster.

API RDS

Panggil operasi Modify DBInstance API dan tentukan nama cluster DB Anda `true` untuk `DBInstanceIdentifier` parameter dan `AutoMinorVersionUpgrade` parameter. Atau, atur parameter `ApplyImmediately` ke `true` untuk segera mengaktifkan pengaturan ini untuk instans DB Anda. Panggil operasi `ModifyDBInstance` terpisah untuk setiap instans DB dalam klaster.

Anda dapat menggunakan perintah CLI seperti berikut ini untuk memeriksa status pengaturan `AutoMinorVersionUpgrade` untuk semua instans DB di klaster Aurora MySQL.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Perintah ini menghasilkan output yang serupa dengan berikut:

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-reader-instance1",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": false
  },
  {
```

```
      "DBInstanceIdentifier": "db-writer-instance2",
      "DBClusterIdentifier": "my-db-cluster-80",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

Dalam contoh ini, opsi Aktifkan peningkatan versi minor otomatis dinonaktifkan untuk klaster DB `my-db-cluster-57`, karena dinonaktifkan untuk salah satu instans DB dalam klaster.

## Pembaruan sistem operasi untuk cluster Aurora DB

Instans DB dalam klaster DB Aurora MySQL dan Aurora PostgreSQL terkadang memerlukan pembaruan sistem operasi. Amazon RDS meningkatkan sistem operasi ke versi yang lebih baru untuk meningkatkan performa basis data dan postur keamanan pelanggan secara keseluruhan. Pembaruan biasanya memerlukan waktu sekitar 10 menit. Pembaruan sistem operasi tidak akan mengubah versi mesin DB atau kelas instans DB dari instans DB.

Ada dua jenis pembaruan sistem operasi, dibedakan dengan deskripsi untuk tindakan pemeliharaan yang tertunda:

- Peningkatan distribusi sistem operasi - Digunakan untuk migrasi ke versi utama Amazon Linux terbaru yang didukung. Deskripsinya adalah`New Operating System upgrade is available`.
- Patch sistem operasi - Digunakan untuk menerapkan berbagai perbaikan keamanan dan terkadang untuk meningkatkan performa basis data. Deskripsinya adalah`New Operating System patch is available`.

Pembaruan sistem operasi bisa opsional atau wajib:

- Pembaruan opsional dapat diterapkan kapan saja. Meskipun pembaruan ini bersifat opsional, sebaiknya Anda menerapkannya secara berkala agar armada RDS Anda tetap diperbarui. RDS tidak menerapkan pembaruan ini secara otomatis.

  Untuk menerima pemberitahuan saat patch sistem operasi opsional yang baru tersedia, Anda dapat berlangganan RDS-EVENT-0230 dalam kategori peristiwa patching keamanan. Untuk informasi tentang berlangganan peristiwa RDS, lihat Berlangganan notifikasi RDS acara Amazon.

> **Note**
>
> `RDS-EVENT-0230` tidak berlaku untuk peningkatan distribusi sistem operasi.

- Pembaruan wajib diperlukan, dan kami mengirim pemberitahuan sebelum pembaruan wajib. Pemberitahuan mungkin berisi tanggal jatuh tempo. Rencanakan untuk menjadwalkan pembaruan sebelum tanggal jatuh tempo ini. Setelah tanggal jatuh tempo yang ditentukan, Amazon RDS secara otomatis meningkatkan sistem operasi untuk instans DB Anda ke versi terbaru selama salah satu periode pemeliharaan yang ditetapkan.

  Peningkatan distribusi sistem operasi bersifat wajib.

> **Note**
>
> Tetap mengikuti semua pembaruan opsional dan wajib mungkin diperlukan untuk memenuhi berbagai kewajiban kepatuhan. Sebaiknya Anda menerapkan semua pembaruan yang disediakan oleh RDS secara rutin selama periode pemeliharaan Anda.

Untuk cluster Aurora DB, Anda dapat menggunakan opsi pemeliharaan tingkat cluster untuk melakukan pembaruan sistem operasi (OS). Temukan opsi untuk melakukan pembaruan tingkat kluster di tab Pemeliharaan & pencadangan saat Anda memilih nama cluster DB Anda di konsol, atau gunakan perintah di. `os-upgrade` AWS CLI Metode ini mempertahankan ketersediaan baca dengan peningkatan bergulir yang secara otomatis menerapkan pembaruan ke beberapa instans DB pembaca sekaligus. Untuk mencegah beberapa kegagalan dan mengurangi waktu henti yang tidak perlu, Aurora memutakhirkan instans DB penulis terakhir.

Pembaruan OS tingkat cluster terjadi selama jendela pemeliharaan yang Anda tentukan untuk cluster. Ini memastikan pembaruan terkoordinasi di seluruh cluster.

Untuk kompatibilitas mundur, Aurora juga mempertahankan opsi pemeliharaan tingkat instans. Namun, kami menyarankan Anda menggunakan pembaruan tingkat cluster sebagai gantinya. Jika Anda harus menggunakan pembaruan tingkat instans, perbarui instans DB pembaca di cluster DB terlebih dahulu, lalu perbarui instans DB penulis. Jika Anda memperbarui instance pembaca dan penulis secara bersamaan, Anda meningkatkan kemungkinan downtime terkait kegagalan. Temukan opsi untuk melakukan pembaruan tingkat instans di tab Pemeliharaan & pencadangan saat Anda memilih nama instans DB Anda di konsol, atau gunakan perintah di. `system-update` AWS CLI

Pembaruan OS tingkat instans terjadi selama jendela pemeliharaan yang Anda tentukan untuk setiap instans masing-masing. Misalnya, jika cluster dan dua instance pembaca memiliki waktu jendela pemeliharaan yang berbeda, pembaruan OS di tingkat cluster sejajar dengan jendela pemeliharaan cluster.

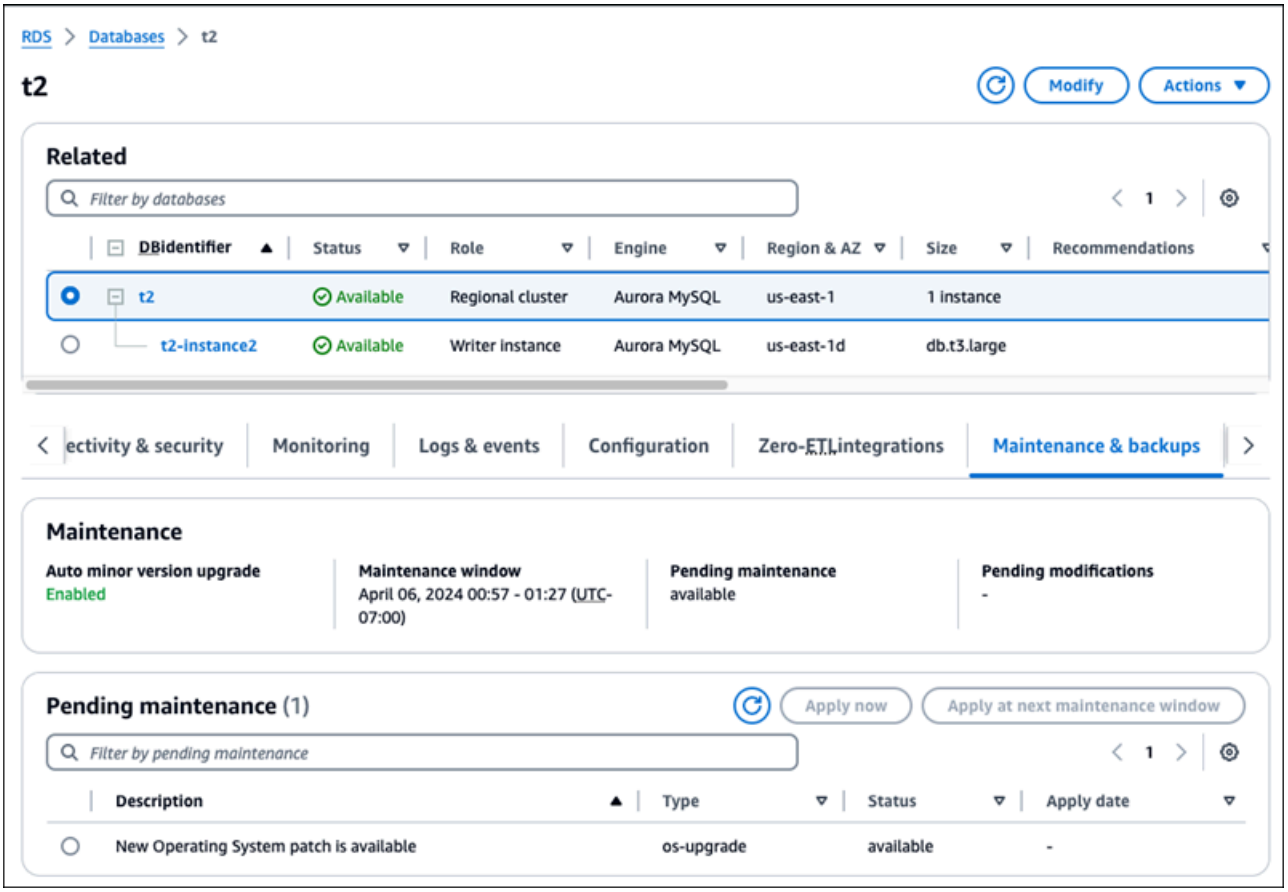
Anda dapat menggunakan AWS Management Console atau AWS CLI untuk mendapatkan informasi tentang jenis upgrade sistem operasi.

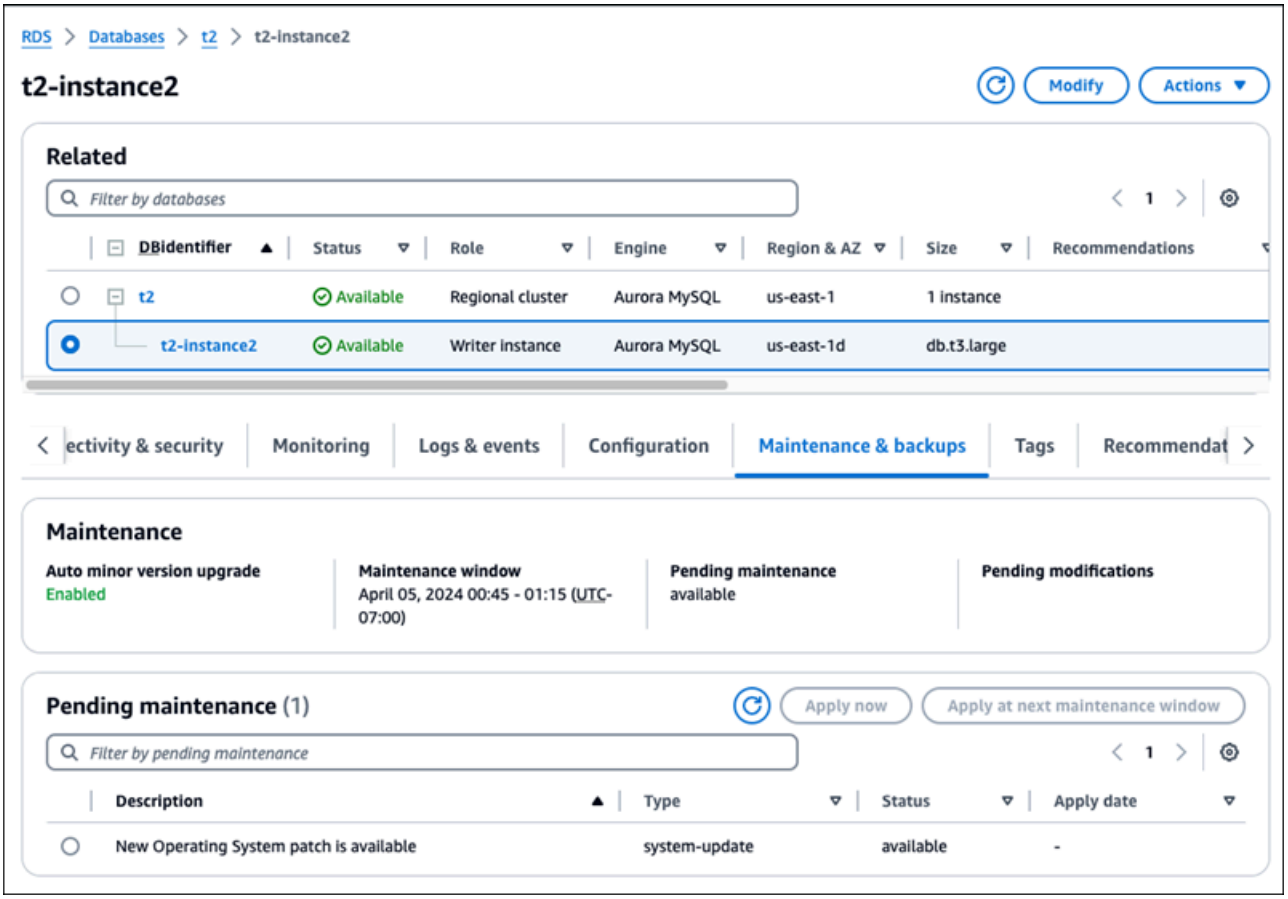
Konsol

Untuk mendapatkan informasi pembaruan menggunakan AWS Management Console

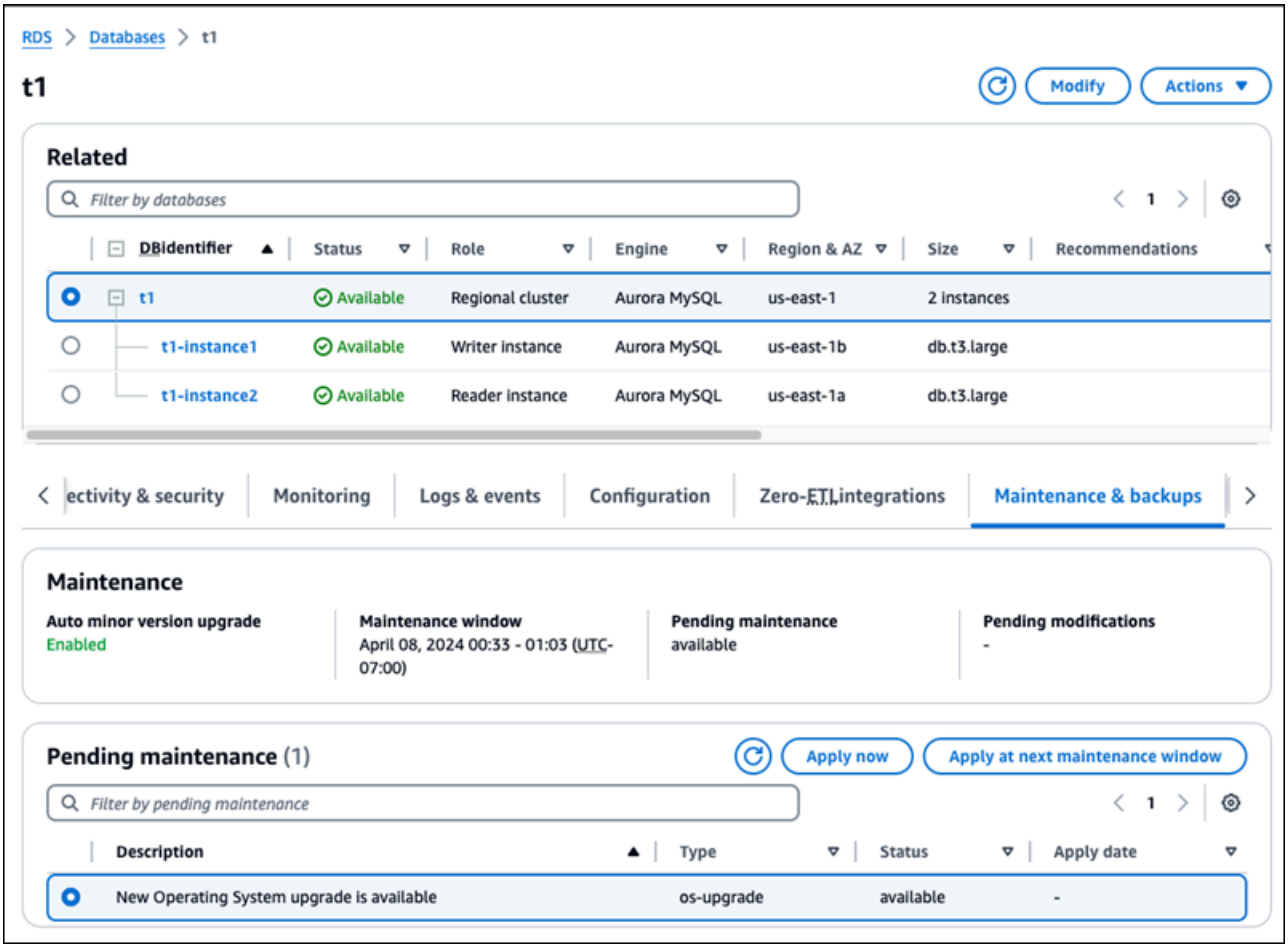
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih instans DB.
3. Pilih Maintenance & backup.
4. Di bagian Pending maintenance, temukan pembaruan sistem operasi, dan periksa nilai Description.

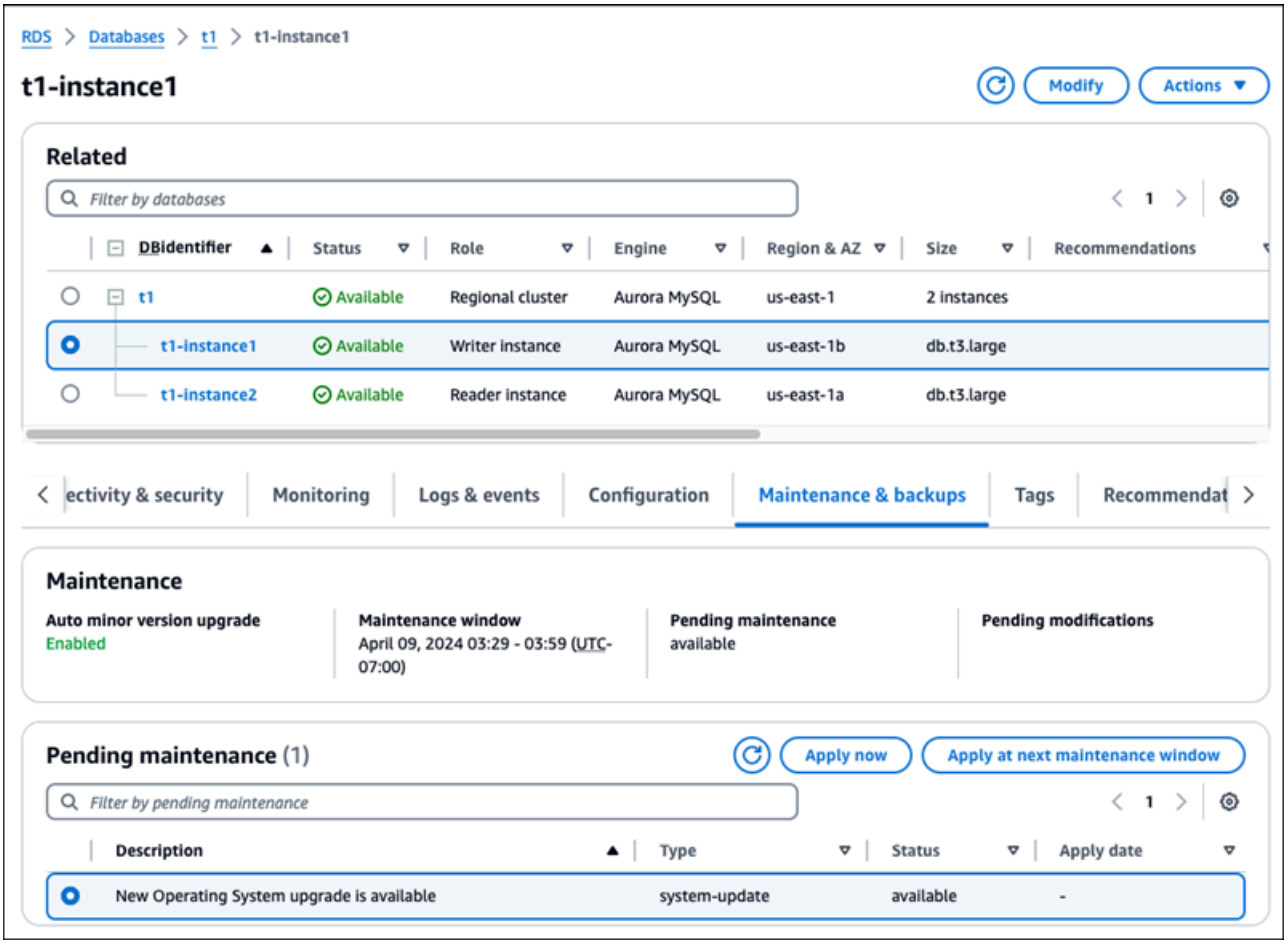
Gambar berikut menunjukkan cluster DB dengan instance DB penulis yang memiliki patch sistem operasi yang tersedia.

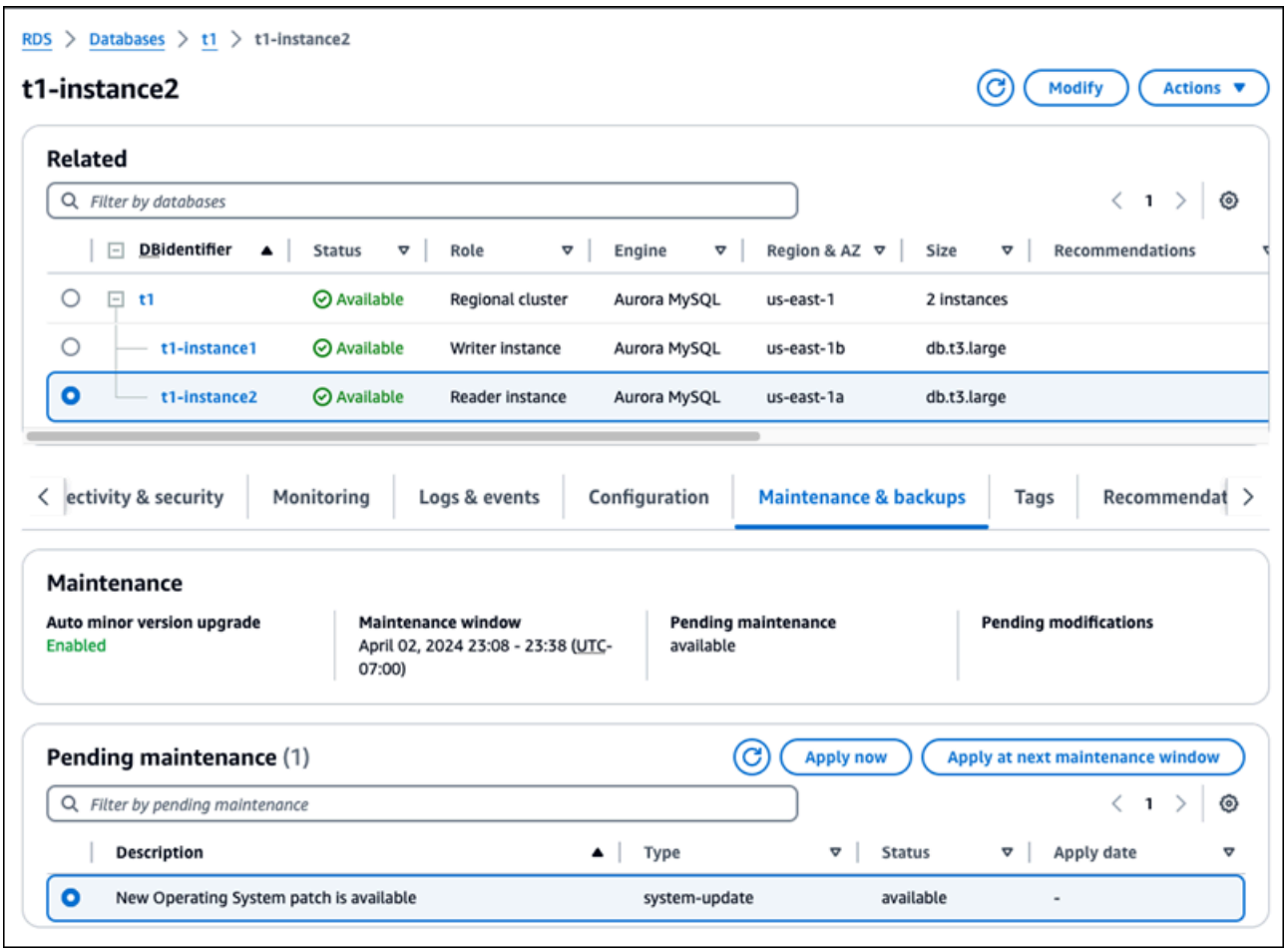

RDS > Databases > t2
t2
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t2
Available
Regional cluster
Aurora MySQL
us-east-1
1 instance
t2-instance2
Available
Writer instance
Aurora MySQL
us-east-1d
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Zero-ETL integrations
Maintenance & backups
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 06, 2024 00:57 - 01:27 (UTC-07:00)
Pending maintenance
available
Pending modifications
-
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System patch is available
os-upgrade
available
-




Gambar berikut menunjukkan cluster DB dengan instance DB penulis dan instance DB pembaca. Contoh penulis memiliki upgrade sistem operasi wajib yang tersedia. Instance pembaca memiliki patch sistem operasi yang tersedia.


RDS > Databases > t1
t1
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t1
Available
Regional cluster
Aurora MySQL
us-east-1
2 instances
t1-instance1
Available
Writer instance
Aurora MySQL
us-east-1b
db.t3.large
t1-instance2
Available
Reader instance
Aurora MySQL
us-east-1a
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Zero-ETL integrations
Maintenance & backups
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 08, 2024 00:33 - 01:03 (UTC-07:00)
Pending maintenance
available
Pending modifications
-
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System upgrade is available
os-upgrade
available
-


RDS > Databases > t1 > t1-instance1
t1-instance1
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t1
Available
Regional cluster
Aurora MySQL
us-east-1
2 instances
t1-instance1
Available
Writer instance
Aurora MySQL
us-east-1b
db.t3.large
t1-instance2
Available
Reader instance
Aurora MySQL
us-east-1a
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Maintenance & backups
Tags
Recommendat
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 09, 2024 03:29 - 03:59 (UTC-07:00)
Pending maintenance
available
Pending modifications
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System upgrade is available
system-update
available
-




AWS CLI

Untuk mendapatkan informasi pembaruan dari AWS CLI, gunakan describe-pending-maintenance-actionsperintah.

```
aws rds describe-pending-maintenance-actions
```

Output berikut menunjukkan upgrade distribusi sistem operasi untuk cluster DB dan instance DB.

```
{
  "PendingMaintenanceActions": [
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:cluster:t3",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "os-upgrade",
          "Description": "New Operating System upgrade is available"
        }
```

```
      ]
    },
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:t3-instance1",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "system-update",
          "Description": "New Operating System upgrade is available"
        }
      ]
    },
  ]
}
```

Output berikut menunjukkan patch sistem operasi untuk instance DB.

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System patch is available"
    }
  ]
}
```

## Ketersediaan pembaruan sistem operasi

Pembaruan sistem operasi khusus untuk versi mesin DB dan kelas instans DB. Oleh karena itu, instans DB menerima atau memerlukan pembaruan di waktu yang berbeda. Ketika pembaruan sistem operasi tersedia untuk instans DB Anda berdasarkan versi mesin dan kelas instansnya, pembaruan akan muncul di konsol. Hal ini juga dapat dilihat dengan menjalankan describe-pending-maintenance-actions AWS CLI perintah atau dengan memanggil operasi DescribePendingMaintenanceActionsRDS API. Jika pembaruan tersedia untuk instans Anda, Anda dapat memperbarui sistem operasi dengan mengikuti petunjuk di Menerapkan pembaruan ke cluster DB.

# Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora

Anda mungkin perlu mem-boot ulang klaster DB atau beberapa instans dalam klaster Anda, biasanya karena alasan pemeliharaan. Sebagai contoh, misalkan Anda mengubah parameter dalam grup parameter atau menghubungkan grup parameter lainnya dengan klaster Anda. Dalam kasus ini, Anda harus mem-boot ulang klaster agar perubahan diterapkan. Sama halnya, Anda mungkin mem-boot ulang satu atau beberapa instans DB pembaca dalam klaster. Anda dapat mengatur operasi boot ulang untuk setiap instans guna meminimalkan waktu henti untuk seluruh klaster.

Waktu yang diperlukan untuk mem-boot ulang setiap instans DB dalam klaster Anda bergantung pada aktivitas basis data pada saat boot ulang. Hal ini juga bergantung pada proses pemulihan mesin DB khusus Anda. Jika memungkinkan, kurangi aktivitas basis data pada instans tertentu sebelum memulai proses boot ulang. Opsi ini dapat mengurangi waktu yang diperlukan untuk memulai ulang basis data.

Anda hanya dapat mem-boot ulang setiap instans DB dalam klaster Anda jika status instans tersebut tersedia. Instans DB mungkin tidak tersedia karena beberapa alasan. Alasan ini meliputi klaster dalam status dihentikan, modifikasi sedang diterapkan pada instans, dan tindakan periode pemeliharaan seperti peningkatan versi.

Boot ulang instans DB akan memulai ulang proses mesin basis data. Mem-boot ulang instans DB akan menyebabkan pemadaman sementara, dan selama pemadaman sementara, status instans DB diatur ke mem-boot ulang.

> **Note**
>
> Jika instans DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait, akan AWS Management Console menampilkan grup parameter DB dengan status pending-reboot. Status grup parameter boot ulang tertunda tidak menyebabkan boot ulang otomatis selama periode pemeliharaan berikutnya. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, boot ulang instans DB secara manual. Untuk informasi selengkapnya tentang grup parameter, lihat .

Topik

- Mem-boot ulang instans DB dalam klaster Aurora
- Mem-boot ulang klaster Aurora dengan ketersediaan baca

- Mem-boot ulang klaster Aurora tanpa ketersediaan baca
- Memeriksa waktu aktif untuk klaster dan instans Aurora
- Contoh operasi boot ulang Aurora

## Mem-boot ulang instans DB dalam klaster Aurora

Prosedur ini adalah operasi terpenting yang Anda lakukan saat mem-boot ulang dengan Aurora. Banyak prosedur pemeliharaan yang mewajibkan Anda mem-boot ulang satu atau beberapa instans DB Aurora dalam urutan tertentu.

Konsol

Untuk mem-boot ulang instans DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih instans DB yang ingin di-boot ulang.
3. Untuk Tindakan, pilih Boot ulang.

   Halaman Boot ulang Instans DB akan muncul.
4. Pilih Boot ulang untuk mem-boot ulang instans DB.

   Atau pilih Batalkan.

AWS CLI

Untuk me-reboot instance DB dengan menggunakan AWS CLI, panggil `reboot-db-instance`perintah.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

Untuk Windows:

```
aws rds reboot-db-instance ^
```

```
    --db-instance-identifier mydbinstance
```

RDS API

Untuk me-reboot instans DB dengan menggunakan Amazon RDSAPI, panggil `RebootDBInstance`operasi.

## Mem-boot ulang klaster Aurora dengan ketersediaan baca

Dengan fitur ketersediaan baca, Anda dapat me-reboot instance penulis cluster Aurora Anda tanpa me-reboot instance pembaca di cluster DB primer atau sekunder. Tindakan ini dapat membantu menjaga ketersediaan tinggi klaster untuk operasi baca saat Anda mem-boot ulang instans penulis. Anda dapat mem-boot ulang instans pembaca di lain waktu pada jadwal yang dikehendaki. Misalnya, dalam cluster produksi Anda dapat me-reboot instance pembaca satu per satu, dimulai hanya setelah reboot instance utama selesai. Untuk setiap instans DB yang di-boot ulang, ikuti prosedur di Mem-boot ulang instans DB dalam klaster Aurora.

Fitur ketersediaan baca untuk cluster DB primer tersedia di Aurora SQL My versi 2.10 dan lebih tinggi. Ketersediaan baca untuk cluster DB sekunder tersedia di Aurora SQL My versi 3.06 dan lebih tinggi.

Untuk Aurora Postgre fitur SQL ini tersedia secara default dalam versi berikut:

- Versi 15.2 dan versi 15 yang lebih tinggi
- Versi 14.7 dan versi 14 yang lebih tinggi
- Versi 13.10 dan versi 13 yang lebih tinggi
- Versi 12.14 dan versi 12 yang lebih tinggi

Untuk informasi lebih lanjut tentang fitur ketersediaan baca di Aurora SQL Postgre, lihat. Meningkatkan ketersediaan baca Aurora Replica

Sebelum fitur ini, me-reboot instance utama menyebabkan reboot untuk setiap instance pembaca secara bersamaan. Jika klaster Aurora Anda menjalankan versi yang lebih lama, gunakan prosedur boot ulang di Mem-boot ulang klaster Aurora tanpa ketersediaan baca.

> **Note**
>
> Perubahan perilaku reboot di cluster Aurora DB dengan ketersediaan baca berbeda untuk database global Aurora di Aurora versi Saya lebih rendah dari 3.06. SQL Jika Anda mem-

boot ulang instans penulis untuk klaster primer dalam basis data global Aurora, instans pembaca dalam klaster primer tetap tersedia. Namun, instans DB di setiap klaster sekunder akan di-boot ulang secara bersamaan.
Versi terbatas dari fitur ketersediaan baca yang ditingkatkan didukung oleh database global Aurora untuk Aurora Postgre SQL versi 12.16, 13.12, 14.9, 15.4, dan lebih tinggi.

Anda sering mem-boot ulang klaster setelah menerapkan perubahan pada grup parameter klaster. Anda menerapkan perubahan parameter dengan mengikuti prosedur di . Misalkan Anda mem-boot ulang instans penulis DB dalam klaster Aurora untuk menerapkan perubahan pada parameter klaster. Beberapa atau semua instans DB pembaca dapat terus menggunakan pengaturan parameter lama. Namun, pengaturan parameter yang berbeda tidak memengaruhi integritas data klaster. Setiap parameter klaster yang memengaruhi susunan file data hanya digunakan oleh instans DB penulis.

Misalnya, di SQL klaster Aurora My, Anda dapat memperbarui parameter cluster seperti `binlog_format` dan `innodb_purge_threads` pada instance penulis sebelum instance pembaca. Hanya instans penulis yang menulis log biner dan membersihkan data undo. Untuk parameter yang mengubah cara kueri menafsirkan SQL pernyataan atau keluaran kueri, Anda mungkin perlu berhati-hati untuk segera me-reboot instance pembaca. Dengan demikian, Anda dapat menghindari perilaku aplikasi tak terduga selama kueri. Sebagai contoh, misalkan Anda mengubah parameter `lower_case_table_names` dan mem-boot ulang instans penulis. Dalam kasus ini, instans pembaca mungkin tidak dapat mengakses tabel yang baru dibuat hingga semua instans di-boot ulang.

Untuk daftar semua parameter SQL cluster Aurora My, lihat. Parameter tingkat klaster

Untuk daftar semua parameter SQL cluster Aurora Postgre, lihat. Parameter tingkat klaster Aurora PostgreSQL

**Tip**

Aurora My SQL mungkin masih me-reboot beberapa instance pembaca bersama dengan instance penulis jika cluster Anda memproses beban kerja dengan throughput tinggi. Pengurangan jumlah boot ulang juga berlaku selama operasi failover. Aurora My SQL hanya me-restart instans DB penulis dan target failover selama failover. Instans DB pembaca lain dalam klaster ini tetap tersedia untuk terus memproses kueri melalui koneksi ke titik akhir pembaca. Oleh karena itu, Anda dapat meningkatkan ketersediaan selama failover dengan menyediakan lebih dari satu instans DB pembaca dalam sebuah klaster.

# Mem-boot ulang klaster Aurora tanpa ketersediaan baca

Dengan fitur ketersediaan baca, Anda dapat mem-boot ulang seluruh klaster DB Aurora dengan mem-boot ulang instans DB dalam klaster tersebut. Untuk melakukannya, ikuti prosedurnya di Mem-boot ulang instans DB dalam klaster Aurora.

Mem-boot ulang instans DB penulis juga akan memulai boot ulang untuk setiap instans DB pembaca dalam klaster. Dengan begitu, setiap perubahan parameter tingkat klaster diterapkan ke semua instans DB secara bersamaan. Namun, jika semua instans DB di-boot ulang, klaster tersebut akan mengalami pemadaman singkat. Instans DB pembaca tetap tidak tersedia hingga instans DB penulis menyelesaikan boot ulang dan menjadi tersedia.

Perilaku reboot ini berlaku untuk semua cluster DB yang dibuat di Aurora SQL My versi 2.09 dan yang lebih rendah.

Untuk Aurora Postgre perilaku SQL ini berlaku untuk versi berikut:

- 14.6 dan versi 14 yang lebih rendah
- 13.9 dan versi 13 yang lebih rendah
- 12.13 dan versi 12 yang lebih rendah
- Semua Postgre 11 versi SQL

Di RDS konsol, instance DB penulis memiliki nilai Writer di bawah kolom Peran pada halaman Database. Dalam RDSCLI, output dari `describe-db-clusters` perintah termasuk bagian`DBClusterMembers`. Elemen `DBClusterMembers` yang mewakili instans DB penulis memiliki nilai `true` untuk kolom `IsClusterWriter`.

> **Important**
>
> Dengan fitur ketersediaan baca, perilaku reboot berbeda di Aurora My dan SQL Aurora PostgreSQL: instance DB pembaca biasanya tetap tersedia saat Anda me-reboot instance penulis. Kemudian, Anda dapat mem-boot ulang instans pembaca pada waktu yang tepat. Anda dapat mem-boot ulang instans pembaca pada jadwal yang berurutan jika menginginkan beberapa instans pembaca selalu tersedia. Untuk informasi selengkapnya, lihat Mem-boot ulang klaster Aurora dengan ketersediaan baca.

# Memeriksa waktu aktif untuk klaster dan instans Aurora

Anda dapat memeriksa dan memantau jangka waktu sejak boot ulang terakhir untuk setiap instans DB di klaster Aurora Anda. CloudWatch Metrik Amazon `EngineUptime` melaporkan jumlah detik sejak terakhir kali instans DB dimulai. Anda dapat memeriksa metrik ini pada titik waktu tertentu guna mengetahui waktu aktif instans DB. Anda juga dapat memantau metrik ini dari waktu ke waktu untuk mendeteksi kapan instans di-boot ulang.

Anda juga dapat memeriksa metrik `EngineUptime` pada tingkat klaster. Dimensi `Minimum` dan `Maximum` melaporkan nilai waktu aktif terkecil dan terbesar untuk semua instans DB dalam klaster. Untuk memeriksa waktu terbaru saat setiap instans pembaca dalam klaster di-boot ulang, atau dimulai ulang karena alasan lain, pantau metrik tingkat klaster menggunakan dimensi `Minimum`. Untuk memeriksa instans dalam klaster telah berjalan paling lama tanpa boot ulang, pantau metrik tingkat klaster menggunakan dimensi `Maximum`. Sebagai contoh, sebaiknya Anda mengonfirmasi bahwa semua instans DB dalam klaster di-boot ulang setelah perubahan konfigurasi.

> **Tip**
>
> Untuk pemantauan jangka panjang, sebaiknya Anda memantau metrik `EngineUptime` untuk setiap instans, bukan di tingkat klaster. Metrik `EngineUptime` tingkat klaster diatur ke nol saat instans DB baru ditambahkan ke klaster. Perubahan klaster tersebut bisa terjadi sebagai bagian dari operasi pemeliharaan dan penskalaan seperti yang dilakukan oleh Penskalaan Otomatis.

CLIContoh berikut menunjukkan cara memeriksa `EngineUptime` metrik untuk contoh penulis dan pembaca dalam sebuah cluster. Contoh tersebut menggunakan sebuah klaster bernama `tpch100g`. Klaster ini memiliki instans DB penulis `instance-1234`. Klaster ini juga memiliki dua instans DB pembaca, `instance-7448` dan `instance-6305`.

Pertama, perintah `reboot-db-instance` mem-boot ulang salah satu instans pembaca. Perintah `wait` menunggu hingga instans selesai di-boot ulang.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
```

```
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

CloudWatch `get-metric-statistics`Perintah memeriksa `EngineUptime` metrik selama lima menit terakhir dengan interval satu menit. Waktu aktif untuk instans `instance-6305` diatur ulang ke nol dan mulai menghitung ke atas lagi. AWS CLI Contoh untuk Linux ini menggunakan substitusi `$()` variabel untuk memasukkan stempel waktu yang sesuai ke dalam perintah. CLI Contoh ini juga menggunakan perintah `sort` Linux untuk mengurutkan output pada saat metrik dikumpulkan. Nilai stempel waktu tersebut adalah kolom ketiga di setiap baris output.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

Waktu aktif minimum untuk klaster diatur ulang ke nol karena salah satu instans dalam klaster di-boot ulang. Waktu aktif maksimum untuk klaster tidak diatur ulang karena setidaknya salah satu instans DB dalam klaster tetap tersedia.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
```

```
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

Kemudian, perintah `reboot-db-instance` lainnya mem-boot ulang instans penulis pada klaster. Perintah `wait` lainnya dijeda hingga instans penulis selesai di-boot ulang.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

Saat ini, metrik `EngineUptime` untuk instans penulis menunjukkan bahwa instans `instance-1234` baru-baru ini di-boot ulang. Instans pembaca `instance-6305` juga di-boot ulang secara otomatis bersama dengan instans penulis. Cluster ini menjalankan Aurora My SQL 2.09, yang tidak membuat instance pembaca tetap berjalan saat instance penulis reboot.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
```

```
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

## Contoh operasi boot ulang Aurora

SQLContoh Aurora My berikut menunjukkan kombinasi operasi reboot yang berbeda untuk instance DB pembaca dan penulis dalam cluster Aurora DB. Setelah setiap reboot, SQL kueri mendemonstrasikan waktu aktif untuk instance di cluster.

Topik

- Menemukan instans penulis dan pembaca untuk klaster Aurora
- Mem-boot ulang satu instans pembaca
- Mem-boot ulang instans penulis
- Mem-boot ulang penulis dan pembaca secara independen
- Menerapkan perubahan parameter cluster ke cluster Aurora My SQL version 2.10

### Menemukan instans penulis dan pembaca untuk klaster Aurora

Dalam SQL cluster Aurora My dengan beberapa instance DB, penting untuk mengetahui mana yang menjadi penulis dan mana yang pembaca. Instans penulis dan pembaca juga dapat beralih peran ketika terjadi operasi failover. Maka, sangat disarankan untuk memeriksa hal berikut sebelum melakukan operasi apa pun yang membutuhkan instans penulis atau pembaca. Dalam kasus ini, nilai `False` untuk `IsClusterWriter` mengidentifikasi instans pembaca, `instance-6305` dan `instance-7448`. Nilai `True` mengidentifikasi instans penulis, `instance-1234`.

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

Sebelum kita memulai contoh boot ulang, instans penulis memiliki waktu aktif sekitar satu minggu. SQLKueri dalam contoh ini menunjukkan cara SQL spesifik Saya untuk memeriksa waktu aktif. Anda dapat menggunakan teknik ini dalam aplikasi basis data. Untuk teknik lain yang menggunakan AWS

CLI dan bekerja untuk kedua mesin Aurora, lihat. Memeriksa waktu aktif untuk klaster dan instans Aurora

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

## Mem-boot ulang satu instans pembaca

Contoh ini mem-boot ulang salah satu instans DB pembaca. Mungkin instans ini mengalami kelebihan beban oleh kueri besar atau banyak koneksi bersamaan. Atau mungkin tertinggal di belakang instans penulis karena masalah jaringan. Setelah memulai operasi boot ulang, contoh tersebut menggunakan perintah `wait` untuk menjeda hingga instans menjadi tersedia. Pada saat itu, instans memiliki waktu aktif beberapa menit.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:35:02.000000 | 00h 03m |
```

```
+----------------------------+---------+
```

Boot ulang instans pembaca tidak memengaruhi waktu aktif instans penulis. Instans ini tetap memiliki waktu aktif sekitar satu minggu.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

## Mem-boot ulang instans penulis

Contoh ini mem-boot ulang instans penulis. Cluster ini menjalankan Aurora My SQL versi 2.09. Karena SQL versi Aurora My lebih rendah dari 2.10, me-reboot instance writer juga me-reboot instance pembaca apa pun di cluster.

Perintah `wait` dijeda hingga proses boot ulang selesai. Sekarang, waktu aktif untuk instans tersebut diatur ulang ke nol. Ada kemungkinan bahwa operasi boot ulang mungkin memakan waktu yang sangat berbeda untuk instans DB penulis dan pembaca. Instans DB penulis dan pembaca melakukan berbagai jenis operasi pembersihan bergantung pada perannya.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
```

```
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

Setelah instans DB penulis di-boot ulang, waktu aktif kedua instans DB pembaca juga akan diatur ulang. Mem-boot ulang instans penulis juga menyebabkan instans pembaca di-boot ulang. Perilaku ini berlaku untuk cluster Aurora Postgre dan Aurora SQL My SQL cluster sebelum versi 2.10.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+

$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

### Mem-boot ulang penulis dan pembaca secara independen

Contoh berikutnya menunjukkan cluster yang menjalankan Aurora My SQL versi 2.10. Dalam SQL versi Aurora My ini dan yang lebih tinggi, Anda dapat me-reboot instance penulis tanpa menyebabkan reboot untuk semua instance pembaca. Dengan demikian, aplikasi padat kueri Anda tidak mengalami pemadaman saat Anda mem-boot ulang instans penulis. Anda dapat mem-boot ulang instans pembaca di lain waktu. Anda juga dapat melakukan beberapa boot ulang saat lalu lintas kueri rendah. Anda juga dapat mem-boot ulang beberapa instans pembaca satu per satu. Dengan demikian, setidaknya satu instans pembaca selalu tersedia untuk lalu lintas kueri aplikasi Anda.

Contoh berikut menggunakan cluster bernama`cluster-2393`, menjalankan Aurora versi SayaSQL. `5.7.mysql_aurora.2.10.0` Klaster ini memiliki instans penulis bernama `instance-9404` dan tiga instans pembaca bernama `instance-6772`, `instance-2470`, dan `instance-5138`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:        cluster-2393
Instance:       instance-5138        False
Instance:       instance-2470        False
Instance:       instance-6772        False
Instance:       instance-9404        True
```

Memeriksa nilai `uptime` dari setiap instans basis data melalui perintah `mysql` menunjukkan bahwa setiap instans memiliki waktu aktif yang kurang lebih sama. Contoh berikut adalah uptime untuk `instance-5138`.

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

Dengan menggunakan CloudWatch, kita bisa mendapatkan informasi uptime yang sesuai tanpa benar-benar masuk ke instance. Dengan demikian, administrator dapat memantau basis data, tetapi tidak dapat melihat atau mengubah data tabel. Dalam hal ini, kita menentukan jangka waktu selama lima menit, dan memeriksa nilai waktu aktif setiap menit. Peningkatan nilai waktu aktif menunjukkan bahwa instans tidak dimulai ulang selama periode tersebut.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
```

```
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

Sekarang kita mem-boot ulang salah satu instans pembaca, `instance-5138`. Kita tunggu instans tersedia lagi setelah boot ulang. Sekarang, pemantauan waktu aktif selama jangka waktu lima menit menunjukkan bahwa waktu aktif diatur ulang ke nol selama waktu tersebut. Nilai waktu aktif terbaru diukur lima detik setelah boot ulang selesai.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

Selanjutnya, kita lakukan boot ulang untuk instans penulis, `instance-9404`. Kita bandingkan nilai waktu aktif untuk instans penulis dan salah satu instans pembaca. Dengan demikian, kita dapat melihat bahwa mem-boot ulang penulis tidak akan mem-boot ulang pembaca. Dalam versi sebelum

Aurora My SQL 2.10, nilai uptime untuk semua pembaca akan diatur ulang pada saat yang sama dengan penulis.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

Untuk memastikan bahwa semua instans pembaca memiliki semua perubahan pada parameter konfigurasi yang sama dengan instans penulis, boot ulang seluruh instans pembaca setelah penulis. Contoh ini mem-boot ulang semua pembaca, lalu menunggu hingga semua pembaca tersedia sebelum melanjutkan.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
```

```
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

Sekarang kita dapat melihat bahwa instans DB penulis memiliki waktu aktif tertinggi. Nilai waktu aktif instans ini meningkat dengan stabil selama periode pemantauan. Semua instans DB pembaca di-boot ulang setelah pembaca. Kita dapat melihat titik dalam periode pemantauan saat setiap pembaca di-boot ulang dan waktu aktif-nya diatur ulang ke nol.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
```

```
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

## Menerapkan perubahan parameter cluster ke cluster Aurora My SQL version 2.10

Contoh berikut menunjukkan cara menerapkan perubahan parameter ke semua instance DB di cluster Aurora My 2.10 Anda. SQL Dengan SQL versi Aurora My ini, Anda me-reboot instance penulis dan semua instance pembaca secara independen.

Contoh menggunakan parameter SQL konfigurasi Saya `lower_case_table_names` untuk ilustrasi. Jika pengaturan parameter ini berbeda antara instans DB penulis dan pembaca, kueri mungkin tidak dapat mengakses tabel yang dinyatakan dengan nama dalam huruf besar atau campuran huruf besar dan kecil. Atau, jika dua nama tabel hanya berbeda dari segi huruf besar dan huruf kecil, kueri mungkin mengakses tabel yang salah.

Contoh ini menunjukkan cara menentukan instans penulis dan pembaca dalam klaster dengan memeriksa atribut `IsClusterWriter` pada setiap instans. Klaster ini bernama `cluster-2393`. Klaster ini memiliki satu instans penulis bernama `instance-9404`. Instans pembaca dalam klaster tersebut bernama `instance-5138` dan `instance-2470`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
cluster-2393
instance-5138        False
instance-2470        False
```

```
instance-9404        True
```

Untuk menunjukkan efek dari perubahan parameter `lower_case_table_names`, kita mengatur dua grup parameter klaster DB. Parameter dari grup parameter `lower-case-table-names-0` ini diatur ke 0. Grup parameter dari grup parameter `lower-case-table-names-1` ini diatur ke 1.

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-1 \
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
```

```
}
```

Nilai default `lower_case_table_names` adalah 0. Dengan pengaturan parameter ini, tabel `foo` berbeda dengan tabel `F00`. Contoh ini memverifikasi bahwa parameter tersebut masih memiliki pengaturan default. Kemudian, contoh tersebut membuat tiga tabel yang hanya berbeda dari segi huruf besar dan huruf kecil dalam namanya.

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Mixed-case table name Foo |
+--------------------------+

mysql> select * from FOO;
```

```
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

Selanjutnya, kita mengaitkan grup parameter DB dengan klaster untuk mengatur parameter `lower_case_table_names` ke 1. Perubahan ini hanya diterapkan setelah setiap instans DB di-boot ulang.

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

Boot ulang pertama yang kita lakukan adalah untuk instans DB penulis. Kemudian, kita tunggu hingga instans tersedia lagi. Pada saat itu, kita terhubung ke titik akhir penulis dan memverifikasi bahwa nilai parameter instans penulis berubah. Perintah `SHOW TABLES` menegaskan bahwa basis data berisi tiga tabel yang berbeda. Namun, semua kueri yang merujuk ke tabel bernama `foo`, `Foo`, atau F00 mengakses tabel yang namanya berisi huruf kecil semua, `foo`.

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

Sekarang, kueri yang menggunakan titik akhir klaster menunjukkan efek dari perubahan parameter. Apakah nama tabel dalam kueri adalah huruf besar, huruf kecil, atau huruf campuran, SQL pernyataan mengakses tabel yang namanya semua huruf kecil.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+

mysql> use lctn;
```

```
mysql> show tables;
+----------------+
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

Contoh berikutnya menunjukkan kueri yang sama seperti sebelumnya. Dalam hal ini, kueri menggunakan titik akhir pembaca dan berjalan di salah satu instans DB pembaca. Instans tersebut belum di-boot ulang. Dengan demikian, kueri tersebut masih memiliki pengaturan awal untuk parameter `lower_case_table_names`. Hal ini berarti bahwa kueri dapat mengakses setiap tabel `foo`, `Foo`, dan `FOO`.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+
```

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

Selanjutnya, kita mem-boot ulang salah satu instans pembaca dan menunggu hingga instans tersebut tersedia lagi.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

Selagi terhubung ke titik akhir instans untuk `instance-2470`, satu kueri menunjukkan bahwa parameter baru sedang diterapkan.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+
```

Pada titik ini, dua instans pembaca dalam klaster sedang berjalan dengan pengaturan `lower_case_table_names` yang berbeda. Dengan demikian, setiap koneksi ke titik akhir pembaca dalam klaster menggunakan nilai untuk pengaturan yang tidak dapat diprediksi ini. Instans pembaca lain juga perlu segera di-boot ulang agar keduanya memiliki pengaturan yang konsisten.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

Contoh berikut menegaskan bahwa semua instans pembaca memiliki pengaturan yang sama untuk parameter `lower_case_table_names`. Perintah ini memeriksa nilai pengaturan `lower_case_table_names` pada setiap instans pembaca. Kemudian, perintah yang sama yang menggunakan titik akhir pembaca menunjukkan bahwa setiap koneksi ke titik akhir pembaca menggunakan salah satu instans pembaca, tetapi tidak dapat memprediksi instans yang mana.

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
```

```
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

Dengan perubahan parameter yang diterapkan di mana-mana, kita dapat melihat efek pengaturan `lower_case_table_names=1`. Baik tabel ini disebut sebagai `foo`, `Foo`, maupun `F00`, kueri akan mengonversi nama menjadi `foo` dan mengakses tabel yang sama dalam setiap kasus.

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

# Gagal atas kluster DB Amazon Aurora

Anda dapat melakukan failover manual dari cluster Aurora DB, misalnya, ketika Anda ingin mengganti instance DB penulis yang disediakan dengan instance penulis. Aurora Serverless v2

Aurora gagal ke instans DB primer baru dengan salah satu dari dua cara:

- Dengan mempromosikan instans DB pembaca yang ada ke instance utama yang baru
- Dengan membuat instans primer baru

Jika cluster DB memiliki satu atau lebih instance pembaca, maka pembaca dipromosikan ke instance utama selama peristiwa kegagalan. Untuk meningkatkan ketersediaan cluster DB Anda, kami sarankan Anda membuat setidaknya satu atau lebih instance pembaca dalam dua atau lebih Availability Zone yang berbeda. Untuk informasi lebih lanjut tentang mekanisme failover, lihatToleransi kesalahan untuk klaster DB Aurora.

Anda dapat menggunakan AWS Management Console, AWS CLI, atau RDS API untuk melakukan failover manual.

## Konsol

Untuk gagal melalui cluster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Databases, lalu pilih instans DB di cluster DB yang ingin Anda gagal.
3. Untuk Tindakan, pilih Failover.

   Halaman konfirmasi muncul.
4. Pilih Failover.

   Halaman Database menunjukkan bahwa status cluster DB Failing-over. Status kembali ke Tersedia saat failover selesai, dan peran untuk instans DB primer baru dan sebelumnya ditampilkan.

## AWS CLI

Untuk gagal melalui cluster DB menggunakan AWS CLI, panggil failover-db-clusterperintah. Tentukan parameter berikut:

- `--db-cluster-identifier`— Cluster DB yang ingin Anda gagal.
- `--target-db-instance-identifier`— Nama instans DB untuk mempromosikan ke instans DB utama.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds failover-db-cluster \
    --db-cluster-identifier mydbcluster \
    --target-db-instance-identifier mydbcluster-instance-2
```

Untuk Windows:

```
aws rds failover-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --target-db-instance-identifier mydbcluster-instance-2
```

## RDS API

Untuk memodifikasi cluster DB menggunakan Amazon RDSAPI, panggil ailoverDBCluster operasi F. Tentukan parameter berikut:

- DBClusterIdentifier
- argetDBInstancePengenal T

# Menghapus instans DB atau klaster DB Aurora

Anda dapat menghapus klaster DB Aurora jika tidak lagi membutuhkannya. Menghapus klaster akan menghapus volume klaster yang berisi semua data Anda. Sebelum menghapus klaster, Anda dapat menyimpan snapshot data Anda. Anda dapat memulihkan snapshot di lain waktu dan membuat klaster baru yang berisi data yang sama.

Anda juga dapat menghapus instans DB dari suatu klaster, sekaligus mempertahankan klaster itu sendiri dan data yang disimpannya. Menghapus instans DB dapat membantu Anda mengurangi biaya jika klaster tidak sibuk, atau Anda tidak memerlukan kapasitas perhitungan beberapa instans DB.

Topik


- Menghapus klaster DB Aurora
- Perlindungan penghapusan untuk klaster Aurora
- Menghapus klaster Aurora yang berhenti
- Menghapus Aurora Cluster SQL saya yang dibaca replika
- Snapshot akhir saat menghapus klaster
- Menghapus instans dari klaster DB Aurora


## Menghapus klaster DB Aurora

Aurora tidak menyediakan metode satu langkah untuk menghapus klaster DB. Pilihan desain ini dimaksudkan untuk mencegah Anda kehilangan data secara tidak sengaja atau mengambil aplikasi Anda secara offline. Aplikasi Aurora biasanya bersifat kritis misi dan membutuhkan ketersediaan tinggi. Dengan demikian, Aurora memudahkan untuk memperbesar dan memperkecil skala kapasitas klaster dengan menambah dan menghapus instans DB. Menghapus klaster DB itu sendiri mengharuskan Anda membuat penghapusan terpisah.

Gunakan prosedur umum berikut untuk menghapus semua instans DB dari klaster, kemudian menghapus klaster itu sendiri.

1. Hapus setiap instans pembaca dalam klaster. Gunakan prosedur di Menghapus instans dari klaster DB Aurora.

   Jika klaster memiliki instans pembaca, menghapus salah satu instans hanya akan mengurangi kapasitas komputasi klaster. Menghapus instans pembaca terlebih dahulu akan memastikan

bahwa klaster tetap tersedia di sepanjang prosedur dan tidak melakukan operasi failover yang tidak perlu.

2. Hapus instans penulis dari klaster. Gunakan lagi prosedur di Menghapus instans dari klaster DB Aurora.

   Saat Anda menghapus instans DB, klaster dan volume klaster terkaitnya tidak akan berubah bahkan setelah Anda menghapus semua instans DB.

3. Hapus klaster DB.

   - AWS Management Console – Pilih klaster, lalu pilih Hapus dari menu Tindakan. Anda dapat memilih opsi berikut untuk mempertahankan data dari klaster jika Anda membutuhkannya nanti:
     - Buat snapshot akhir dari volume klaster. Pengaturan default-nya adalah membuat snapshot akhir.
     - Pertahankan cadangan otomatis. Pengaturan default bukan untuk mempertahankan cadangan otomatis.

     > **Note**
     >
     > Cadangan otomatis untuk klaster DB Aurora Serverless v1 tidak dipertahankan.

     Aurora juga meminta Anda mengonfirmasi bahwa Anda berniat untuk menghapus klaster.
   - CLIdan API — Panggil `delete-db-cluster` CLI perintah atau `DeleteDBCluster` API operasi. Anda dapat memilih opsi berikut untuk mempertahankan data dari klaster jika Anda membutuhkannya nanti:
     - Buat snapshot akhir dari volume klaster.
     - Pertahankan cadangan otomatis.

     > **Note**
     >
     > Cadangan otomatis untuk klaster DB Aurora Serverless v1 tidak dipertahankan.

Topik

- Menghapus klaster Aurora kosong
- Menghapus klaster Aurora dengan satu instans DB
- Menghapus klaster Aurora dengan beberapa instans DB

## Menghapus klaster Aurora kosong

Anda dapat menghapus cluster DB kosong menggunakan AWS Management Console, AWS CLI, atau Amazon RDSAPI.

> **Tip**
>
> Anda dapat menyimpan klaster tanpa instans DB untuk menyimpan data Anda tanpa menimbulkan CPU biaya untuk klaster. Anda dapat segera mulai menggunakan klaster tersebut lagi dengan membuat satu atau beberapa instans DB baru untuk klaster tersebut. Anda dapat melakukan operasi administratif spesifik Aurora pada klaster meskipun klaster tersebut tidak memiliki instans DB terkait. Anda hanya tidak dapat mengakses data atau melakukan operasi apa pun yang memerlukan koneksi ke instans DB.

### Konsol

Untuk menghapus klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin dihapus.
3. Untuk Tindakan, pilih Hapus.
4. Untuk membuat snapshot DB akhir untuk klaster DB, pilih Buat snapshot akhir?. Ini adalah pengaturan default.
5. Jika Anda memilih untuk membuat snapshot akhir, masukkan Nama snapshot akhir.
6. Untuk mempertahankan cadangan otomatis, pilih Pertahankan cadangan otomatis. Ini bukan pengaturan default.
7. Masukkan **`delete me`** dalam kotak.
8. Pilih Hapus.

### CLI

Untuk menghapus cluster Aurora DB kosong dengan menggunakan AWS CLI, panggil perintah. delete-db-cluster

Misalkan klaster kosong `deleteme-zero-instances` hanya digunakan untuk pengembangan dan pengujian dan tidak berisi data penting. Dalam hal ini, Anda tidak perlu untuk mempertahankan snapshot volume klaster saat menghapus klaster. Contoh berikut menunjukkan bahwa klaster tidak berisi instans DB, kemudian menghapus klaster kosong tanpa membuat snapshot akhir atau mempertahankan cadangan otomatis.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS API

Untuk menghapus cluster Aurora DB kosong dengan menggunakan Amazon RDSAPI, panggil operasi D eleteDBCluster.

## Menghapus klaster Aurora dengan satu instans DB

Anda dapat menghapus instans DB terakhir, meskipun klaster DB mengaktifkan perlindungan penghapusan. Dalam kasus ini, klaster DB sendiri masih ada dan data Anda dipertahankan. Anda dapat mengakses data lagi dengan melampirkan instans DB baru ke klaster.

Contoh berikut menunjukkan bagaimana perintah `delete-db-cluster` tidak bekerja jika klaster masih memiliki instans DB terkait. Klaster ini memiliki satu instans DB penulis. Saat memeriksa instans DB di klaster, kita memeriksa atribut `IsClusterWriter` masing-masing. Klaster bisa memiliki nol atau satu instans DB penulis. Nilai `true` menandakan instans DB penulis. Nilai `false` menandakan instans DB pembaca. Klaster bisa memiliki nol, satu, atau banyak instans DB pembaca. Dalam hal ini, kita menghapus instans DB penulis menggunakan perintah `delete-db-instance`. Segera setelah instans DB masuk ke status `deleting`, kita juga dapat menghapus klaster. Untuk contoh ini juga, misalkan klaster tidak berisi data yang layak untuk disimpan. Oleh karena itu, kita tidak membuat snapshot dari volume klaster atau mempertahankan cadangan otomatis.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

## Menghapus klaster Aurora dengan beberapa instans DB

Jika klaster Anda berisi beberapa instans DB, biasanya ada satu instans penulis dan satu atau beberapa instans pembaca. Instans pembaca membantu dengan ketersediaan tinggi, dengan

bersiaga untuk mengambil alih jika instans penulis mengalami masalah. Anda juga dapat menggunakan instans pembaca untuk memperbesar skala klaster guna menangani beban kerja padat baca tanpa menambahkan overhead ke instans penulis.

Untuk menghapus klaster dengan beberapa instans DB pembaca, hapus instans pembaca terlebih dahulu, kemudian instans penulis. Menghapus instans penulis meninggalkan klaster dan data di tempat. Anda menghapus klaster melalui tindakan terpisah.

- Untuk prosedur penghapusan instans Aurora DB, lihat Menghapus instans dari klaster DB Aurora.
- Untuk prosedur menghapus instans DB penulis dalam klaster Aurora, lihat Menghapus klaster Aurora dengan satu instans DB.
- Untuk prosedur penghapusan klaster Aurora kosong, lihat Menghapus klaster Aurora kosong.

CLIContoh ini menunjukkan cara menghapus cluster yang berisi instance DB penulis dan instance DB pembaca tunggal. Output `describe-db-clusters` menunjukkan bahwa `instance-7384` adalah instans penulis dan `instance-1039` adalah instans pembaca. Contoh ini menghapus instans pembaca terlebih dahulu, karena menghapus instans penulis saat instans pembaca masih ada akan menyebabkan operasi failover. Mempromosikan instans pembaca menjadi penulis jika Anda juga berencana untuk menghapus instans tersebut bukanlah hal yang masuk akal. Sekali lagi, misalkan instans `db.t2.small` ini hanya digunakan untuk pengembangan dan pengujian, sehingga operasi hapus melewati snapshot akhir dan tidak mempertahankan cadangan otomatis.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039  False
Instance:       instance-7384   True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
```

```
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Contoh berikut menunjukkan cara menghapus klaster DB yang berisi instans DB penulis dan beberapa instans DB pembaca. Contoh tersebut menggunakan output ringkas dari perintah `describe-db-clusters` untuk mendapatkan laporan instans penulis dan pembaca. Sekali lagi, kita menghapus semua instans DB pembaca sebelum menghapus instans DB penulis. Urutan penghapusan instans DB pembaca bukanlah hal penting.

Misalkan klaster dengan beberapa instans DB ini berisi data yang layak disimpan. Maka, perintah `delete-db-cluster` dalam contoh ini mencakup parameter `--no-skip-final-snapshot` dan `--final-db-snapshot-identifier` untuk menentukan detail snapshot yang akan dibuat. Ini juga mencakup parameter `--no-delete-automated-backups` untuk mempertahankan cadangan otomatis.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True
```

```
$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Contoh berikut menunjukkan cara mengonfirmasi bahwa Aurora membuat snapshot yang diminta. Anda dapat meminta detail untuk snapshot tertentu dengan menentukan ID–nya `deleteme-multiple-readers-final-snapshot`. Anda juga bisa mendapatkan laporan tentang semua snapshot untuk klaster yang telah dihapus dengan menentukan ID klaster `deleteme-multiple-readers`. Kedua perintah tersebut menampilkan informasi tentang snapshot yang sama.

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

## Perlindungan penghapusan untuk klaster Aurora

Anda tidak dapat menghapus klaster yang perlindungan penghapusannya diaktifkan. Anda dapat menghapus instans DB dalam klaster, tapi bukan klaster itu sendiri. Dengan demikian, volume klaster yang berisi semua data Anda aman dari penghapusan yang tidak disengaja. Aurora memberlakukan perlindungan penghapusan untuk cluster DB apakah Anda mencoba menghapus cluster menggunakan konsol,, atau. AWS CLI RDS API

Perlindungan penghapusan diaktifkan secara default ketika Anda membuat klaster DB produksi menggunakan AWS Management Console. Namun, perlindungan penghapusan dinonaktifkan secara default jika Anda membuat cluster menggunakan or. AWS CLI API Mengaktifkan atau menonaktifkan perlindungan penghapusan tidak menyebabkan pemadaman. Untuk dapat menghapus klaster, ubah klaster dan nonaktifkan perlindungan penghapusan. Untuk informasi selengkapnya tentang cara mengaktifkan dan menonaktifkan perlindungan penghapusan, lihat Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

> **Tip**
>
> Meskipun semua instans DB dihapus, Anda dapat mengakses data dengan membuat instans DB baru di klaster.

## Menghapus klaster Aurora yang berhenti

Anda tidak dapat menghapus klaster jika berada dalam status `stopped`. Dalam kasus ini, mulailah klaster sebelum menghapusnya. Untuk informasi selengkapnya, lihat Memulai klaster DB Aurora.

## Menghapus Aurora Cluster SQL saya yang dibaca replika

Untuk Aurora MySQL, Anda tidak dapat menghapus instans DB di cluster DB jika kedua kondisi berikut benar:

- Klaster DB adalah replika baca dari klaster DB Aurora lainnya.
- Instans DB adalah satu-satunya instans dalam klaster DB.

Untuk menghapus instans DB dalam kasus ini, pertama, promosikan klaster DB agar tidak lagi menjadi replika baca. Setelah promosi selesai, Anda dapat menghapus instans DB akhir dalam klaster DB. Untuk informasi selengkapnya, lihat Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS.

## Snapshot akhir saat menghapus klaster

Sepanjang bagian ini, contoh ini menunjukkan bagaimana Anda dapat memilih apakah akan mengambil snapshot akhir ketika menghapus klaster Aurora. Jika Anda memilih untuk mengambil snapshot akhir, tetapi nama yang Anda tentukan sama dengan snapshot yang ada, operasi akan berhenti dengan kesalahan. Dalam kasus ini, periksa detail snapshot untuk mengonfirmasi apakah snapshot tersebut mewakili detail Anda saat ini atau apakah snapshot ini snapshot lama. Jika snapshot yang ada tidak memiliki data terbaru yang ingin dipertahankan, ganti nama snapshot dan coba lagi, atau tentukan nama yang berbeda untuk parameter snapshot akhir.

## Menghapus instans dari klaster DB Aurora

Anda dapat menghapus instans DB dari klaster DB Aurora sebagai bagian dari proses penghapusan seluruh klaster. Jika klaster Anda berisi sejumlah instans DB, maka penghapusan klaster

mengharuskan penghapusan setiap instans DB. Anda juga dapat menghapus satu atau beberapa instans pembaca dari klaster sekaligus membiarkan klaster berjalan. Anda dapat melakukannya untuk mengurangi kapasitas komputasi dan biaya terkait jika klaster Anda tidak sibuk.

Untuk menghapus instans DB, tentukan nama instans.

Anda dapat menghapus instans DB menggunakan AWS CLI,, atau RDSAPI. AWS Management Console

> **Note**
>
> Saat Aurora Replica dihapus, titik akhir instansnya akan segera dihilangkan, dan Aurora Replica akan dihilangkan dari titik akhir pembaca. Jika ada pernyataan yang dijalankan dari Aurora Replica yang sedang dihapus, masa tenggangnya adalah tiga menit. Pernyataan yang ada dapat diselesaikan selama masa tenggang. Setelah masa tenggang berakhir, Aurora Replica akan dimatikan dan dihapus.

Untuk klaster DB Aurora, menghapus instans DB tidak selalu menghapus seluruh klaster. Anda dapat menghapus instans DB di klaster Aurora untuk mengurangi kapasitas komputasi dan biaya terkait ketika klaster tidak sibuk. Untuk informasi tentang keadaan khusus bagi klaster Aurora yang memiliki satu instans DB atau nol instans DB, lihat Menghapus klaster Aurora dengan satu instans DB dan Menghapus klaster Aurora kosong.

> **Note**
>
> Anda tidak dapat menghapus klaster DB saat perlindungan penghapusan diaktifkan. Untuk informasi selengkapnya, lihat Perlindungan penghapusan untuk klaster Aurora.
> Anda dapat menonaktifkan perlindungan penghapusan dengan mengubah klaster DB. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

Konsol

Untuk menghapus instans DB dalam klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin dihapus.

3. Untuk Tindakan, pilih Hapus.
4. Masukkan **`delete me`** dalam kotak.
5. Pilih Hapus.

### AWS CLI

Untuk menghapus instance DB dengan menggunakan AWS CLI, panggil delete-db-instanceperintah dan tentukan `--db-instance-identifier` nilainya.

#### Example

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

Untuk Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

### RDS API

Untuk menghapus instans DB dengan menggunakan Amazon RDSAPI, panggil eleteDBInstance operasi D dan tentukan `DBInstanceIdentifier` parameternya.

> **Note**
>
> Ketika status untuk instans DB adalah`deleting`, nilai sertifikat CA-nya tidak muncul di RDS konsol atau output untuk AWS CLI perintah atau RDS API operasi. Untuk informasi selengkapnya tentang sertifikat CA, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

# Menandai sumber daya Amazon Aurora dan Amazon RDS

RDSTag Amazon adalah pasangan nama-nilai yang Anda tentukan dan kaitkan dengan RDS sumber daya Amazon seperti instans DB atau snapshot DB. Nama ini disebut sebagai kunci. Secara opsional, Anda dapat memberikan nilai untuk kunci tersebut.

Anda dapat menggunakan AWS Management Console, AWS CLI, atau Amazon RDS API untuk menambahkan, membuat daftar, dan menghapus tag di RDS sumber daya Amazon. Saat menggunakan CLI orAPI, pastikan untuk memberikan Amazon Resource Name (ARN) agar RDS sumber daya dapat digunakan. Untuk informasi lebih lanjut tentang membangun sebuahARN, lihatMembuat konsep ARN untuk Amazon RDS.

Anda dapat menggunakan tag untuk menambahkan metadata ke sumber daya Amazon dan Amazon Anda. RDS Anda dapat menggunakan tag untuk menambahkan notasi Anda sendiri tentang instans basis data, snapshot, klaster Aurora, dan sebagainya. Melakukannya dapat membantu Anda mendokumentasikan sumber daya Aurora dan Amazon RDS Anda. Anda juga dapat menggunakan tag dengan prosedur pemeliharaan otomatis.

Secara khusus, Anda dapat menggunakan tag ini dengan IAM kebijakan. Anda dapat menggunakannya untuk mengelola akses ke RDS sumber daya Aurora dan Amazon dan untuk mengontrol tindakan apa yang dapat diterapkan pada sumber daya tersebut. Anda dapat menggunakan tag ini untuk melacak biaya dengan mengelompokkan pengeluaran untuk sumber daya serupa yang diberi tag.

Anda dapat menandai sumber daya Aurora dan Amazon RDS berikut:

- Instans DB
- Klaster DB
- Kluster global Aurora
- Titik akhir cluster DB
- Replika baca
- Snapshot DB
- Snapshot klaster DB
- Instans DB terpesan
- Langganan peristiwa
- Grup opsi DB
- Grup parameter DB

- Grup parameter klaster DB
- Grup subnet DB
- RDSProksi
- RDSTitik akhir proxy
- Deployment blue/green
- ETLIntegrasi nol

> **Note**
>
> Saat ini, Anda tidak dapat menandai RDS Proxy dan titik akhir RDS Proxy dengan menggunakan. AWS Management Console

Topik


- Mengapa menggunakan tag RDS sumber daya Amazon?
- Cara kerja tag RDS sumber daya Amazon
- Praktik terbaik untuk menandai sumber daya Amazon RDS
- Menyalin tag ke snapshot klaster DB
- Menambahkan dan menghapus tag di Amazon RDS
- Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.


## Mengapa menggunakan tag RDS sumber daya Amazon?

Anda dapat menggunakan tag untuk melakukan hal berikut:

- Kategorikan RDS sumber daya Anda berdasarkan aplikasi, proyek, departemen, lingkungan, dan sebagainya. Misalnya, Anda dapat menggunakan kunci tag untuk menentukan kategori, di mana nilai tag adalah item dalam kategori ini. Anda dapat membuat tag`environment=prod`. Atau Anda dapat menentukan kunci tag `project` dan nilai tag`Salix`, yang menunjukkan bahwa RDS sumber daya Amazon ditetapkan ke proyek Salix.
- Mengotomatiskan tugas manajemen sumber daya. Misalnya, Anda dapat membuat jendela pemeliharaan untuk instance yang diberi tag `environment=prod` yang berbeda dari jendela untuk instance yang diberi tag. `environment=test` Anda juga dapat mengonfigurasi snapshot DB otomatis untuk instance yang ditandai. `environment=prod`

- Kontrol akses ke RDS sumber daya dalam suatu IAM kebijakan. Anda dapat melakukan ini dengan menggunakan kunci kondisi `aws:ResourceTag/tag-key` global. Misalnya, kebijakan mungkin hanya mengizinkan pengguna dalam `DBAdmin` grup untuk memodifikasi instans DB yang diberi tag. `environment=prod` Untuk informasi tentang mengelola akses ke sumber daya yang ditandai dengan IAM kebijakan, lihat Manajemen identitas dan akses untuk Aurora dan Mengontrol akses ke AWS sumber daya di Panduan Pengguna AWS Identity and Access Management.
- Pantau sumber daya berdasarkan tag. Misalnya, Anda dapat membuat CloudWatch dasbor Amazon untuk instans DB yang ditandai dengan. `environment=prod`
- Lacak biaya dengan mengelompokkan biaya untuk sumber daya yang ditandai serupa. Misalnya, jika Anda menandai RDS sumber daya yang terkait dengan proyek Salix`project=Salix`, Anda dapat menghasilkan laporan biaya dan mengalokasikan pengeluaran untuk proyek ini. Untuk informasi selengkapnya, lihat Cara kerja AWS penagihan dengan tag di Amazon RDS.

## Cara kerja tag RDS sumber daya Amazon

AWS tidak menerapkan makna semantik apa pun pada tag Anda. Tanda ditafsirkan dengan ketat sebagai string karakter.

Topik

- Set tag di Amazon RDS
- Struktur tag di Amazon RDS
- RDSSumber daya Amazon memenuhi syarat untuk penandaan
- Cara kerja AWS penagihan dengan tag di Amazon RDS

### Set tag di Amazon RDS

Setiap RDS sumber daya Amazon memiliki wadah yang disebut kumpulan tag. Wadah mencakup semua tag yang ditetapkan ke sumber daya. Sumber daya memiliki tepat satu set tag.

Satu set tag berisi 0-50 tag. Jika Anda menambahkan tag ke RDS sumber daya dengan kunci yang sama dengan tag sumber daya yang ada, nilai baru akan menimpa yang lama.

### Struktur tag di Amazon RDS

Struktur RDS tag adalah sebagai berikut:

Kunci tag

Kuncinya adalah nama tag yang diperlukan. Nilai string harus 1—128 karakter Unicode panjangnya dan tidak dapat diawali dengan atau. `aws:` `rds:` String hanya dapat berisi kumpulan huruf Unicode, digit, spasi putih,,,,_,., : /=, + dan. - @ Java regex adalah. `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"` Kunci tag peka terhadap huruf besar dan kecil. Dengan demikian, kunci `project` dan `Project` berbeda.

Kunci unik untuk set tag. Misalnya, Anda tidak dapat memiliki pasangan kunci dalam set tag dengan kunci yang sama tetapi dengan nilai yang berbeda, seperti `project=Trinity` dan. `project=Xanadu`

Nilai tag

Nilai adalah nilai string opsional dari tag. Nilai string harus 1-256 karakter Unicode panjangnya. String hanya dapat berisi kumpulan huruf Unicode, digit, spasi putih,,,,_,., : /=, + dan. - @ Java regex adalah. `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"` Kunci dan nilai tag peka terhadap huruf besar dan kecil. Dengan demikian, nilai-nilai `prod` dan `Prod` berbeda.

Nilai tidak perlu unik dalam satu set tag dan bisa null. Misalnya, Anda dapat menggunakan pasangan kunci-nilai dalam satu rangkaian tag `project=Trinity` dan `cost-center=Trinity`.

## RDSSumber daya Amazon memenuhi syarat untuk penandaan

Anda dapat menandai RDS sumber daya Amazon berikut:

- Instans DB
- Klaster DB
- Titik akhir cluster DB
- Replika baca
- Snapshot DB
- Snapshot klaster DB
- Instans DB terpesan
- Langganan peristiwa
- Grup opsi DB
- Grup parameter DB

- Grup parameter klaster DB
- Grup subnet DB
- RDSProksi
- RDSTitik akhir proxy

> **Note**
>
> Saat ini, Anda tidak dapat menandai RDS Proxy dan titik akhir RDS Proxy dengan menggunakan. AWS Management Console

- Deployment blue/green
- ETLIntegrasi nol (pratinjau)

## Cara kerja AWS penagihan dengan tag di Amazon RDS

Gunakan tag untuk mengatur AWS tagihan Anda untuk mencerminkan struktur biaya Anda sendiri. Untuk melakukan ini, daftar untuk mendapatkan Akun AWS tagihan Anda dengan nilai kunci tag disertakan. Kemudian, untuk melihat biaya sumber daya gabungan, atur informasi penagihan Anda sesuai dengan sumber daya Anda dengan nilai kunci tag yang sama. Misalnya, Anda dapat memberi tag pada beberapa sumber daya dengan nama aplikasi tertentu, kemudian mengatur informasi penagihan Anda untuk melihat biaya total aplikasi tersebut di beberapa layanan. Untuk informasi selengkapnya, lihat Menggunakan Tag Alokasi Biaya dalam Panduan Pengguna AWS Billing .

Bagaimana tag alokasi biaya bekerja dengan snapshot cluster DB

Anda dapat menambahkan tag ke snapshot cluster DB. Namun, tagihan Anda tidak akan mencerminkan pengelompokan ini. Agar tag alokasi biaya diterapkan ke snapshot cluster DB, kondisi berikut harus dipenuhi:

- Tag harus dilampirkan ke instance DB induk.
- Instance DB induk harus ada Akun AWS sama dengan snapshot cluster DB.
- Instance DB induk harus ada Wilayah AWS sama dengan snapshot cluster DB.

Snapshot cluster DB dianggap yatim piatu jika tidak ada di Wilayah yang sama dengan instans DB induk, atau jika instans DB induk dihapus. Snapshot DB yatim piatu tidak mendukung tag alokasi biaya. Biaya untuk snapshot yatim piatu digabungkan dalam satu item baris yang tidak ditandai. Snapshot cluster DB lintas akun tidak dianggap yatim piatu ketika kondisi berikut terpenuhi:

- Mereka ada di Wilayah yang sama dengan instans DB induk.
- Instance DB induk dimiliki oleh akun sumber.

> **Note**
>
> Jika instans DB induk dimiliki oleh akun lain, tag alokasi biaya tidak berlaku untuk snapshot lintas akun di akun tujuan.

## Praktik terbaik untuk menandai sumber daya Amazon RDS

Saat Anda menggunakan tag, kami sarankan Anda mematuhi praktik terbaik berikut:

- Konvensi dokumen untuk penggunaan tag yang diikuti oleh semua tim di organisasi Anda. Secara khusus, pastikan nama-nama itu deskriptif dan konsisten. Misalnya, standarisasi pada format `environment:prod` daripada menandai beberapa sumber daya dengan. `env:production`

  > **Important**
  >
  > Jangan menyimpan informasi identitas pribadi (PII) atau informasi rahasia atau sensitif lainnya dalam tag.

- Otomatiskan penandaan untuk memastikan konsistensi. Misalnya, Anda dapat menggunakan teknik berikut:
  - Sertakan tag dalam AWS CloudFormation templat. Saat Anda membuat sumber daya dengan templat, sumber daya ditandai secara otomatis.
  - Tentukan dan terapkan tag menggunakan AWS Lambda fungsi.
  - Buat SSM dokumen yang menyertakan langkah-langkah untuk menambahkan tag ke RDS sumber daya Anda.
- Gunakan tag hanya jika diperlukan. Anda dapat menambahkan hingga 50 tag untuk satu RDS sumber daya, tetapi praktik terbaik adalah menghindari proliferasi dan kompleksitas tag yang tidak perlu.
- Tinjau tag secara berkala untuk relevansi dan akurasi. Hapus atau ubah tag usang sesuai kebutuhan.
- Pertimbangkan untuk membuat AWS tag dengan Editor Tag di AWS Management Console. Anda dapat menggunakan Editor Tag untuk menambahkan tag ke beberapa AWS sumber daya yang

didukung, termasuk RDS sumber daya, secara bersamaan. Untuk informasi selengkapnya, lihat Editor Tag di Panduan Pengguna AWS .

## Menyalin tag ke snapshot klaster DB

Saat Anda membuat atau memulihkan cluster DB, Anda dapat menentukan bahwa tag dari cluster disalin ke snapshot dari cluster DB. Penyalinan tag memastikan bahwa metadata untuk snapshot DB cocok dengan metadata klaster DB sumber. Ini juga memastikan bahwa kebijakan akses apa pun untuk snapshot DB juga cocok dengan kebijakan akses klaster DB sumber. Tag tidak disalin secara default.

Anda dapat menentukan bahwa tag disalin ke snapshot DB untuk tindakan berikut:

- Membuat klaster DB
- Memulihkan cluster DB
- Membuat replika baca
- Menyalin snapshot klaster DB

> **Note**
>
> Dalam beberapa kasus, Anda mungkin menyertakan nilai untuk `--tags` parameter create-db-snapshot AWS CLI perintah. Atau Anda mungkin menyediakan setidaknya satu tag untuk reateDBSnapshot API operasi C. Dalam kasus ini, RDS tidak menyalin tag dari instans DB sumber ke snapshot DB baru. Fungsi ini berlaku meskipun opsi `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) di instans DB sumber diaktifkan.
> Jika menggunakan pendekatan ini, Anda dapat membuat salinan instans DB dari snapshot DB. Pendekatan ini menghindari penambahan tag yang tidak berlaku untuk instans DB baru. Anda membuat snapshot DB Anda menggunakan AWS CLI `create-db-snapshot` perintah (atau `CreateDBSnapshot` RDS API operasi). Setelah membuat snapshot DB, Anda dapat menambahkan tag seperti yang dijelaskan nanti dalam topik ini.

## Menambahkan dan menghapus tag di Amazon RDS

Anda dapat melakukan tindakan berikut:

- Buat tag saat Anda membuat sumber daya, misalnya, saat Anda menjalankan AWS CLI perintah`create-db-instance`.
- Tambahkan tag ke sumber daya yang ada menggunakan perintah`add-tags-to-resource`.
- Daftar tag yang terkait dengan sumber daya tertentu menggunakan perintah`list-tags-for-resource`.
- Perbarui tag menggunakan perintah`add-tags-to-resource`.
- Hapus tag dari sumber daya menggunakan perintah`remove-tags-from-resource`.

Prosedur berikut menunjukkan bagaimana Anda dapat melakukan operasi penandaan khas pada sumber daya yang terkait dengan instans DB dan cluster Aurora DB. Perhatikan bahwa tag disimpan di cache untuk diotorisasi. Untuk alasan ini, ketika Anda menambahkan atau memperbarui tag pada RDS sumber daya Amazon, beberapa menit dapat berlalu sebelum modifikasi tersedia.

Konsol

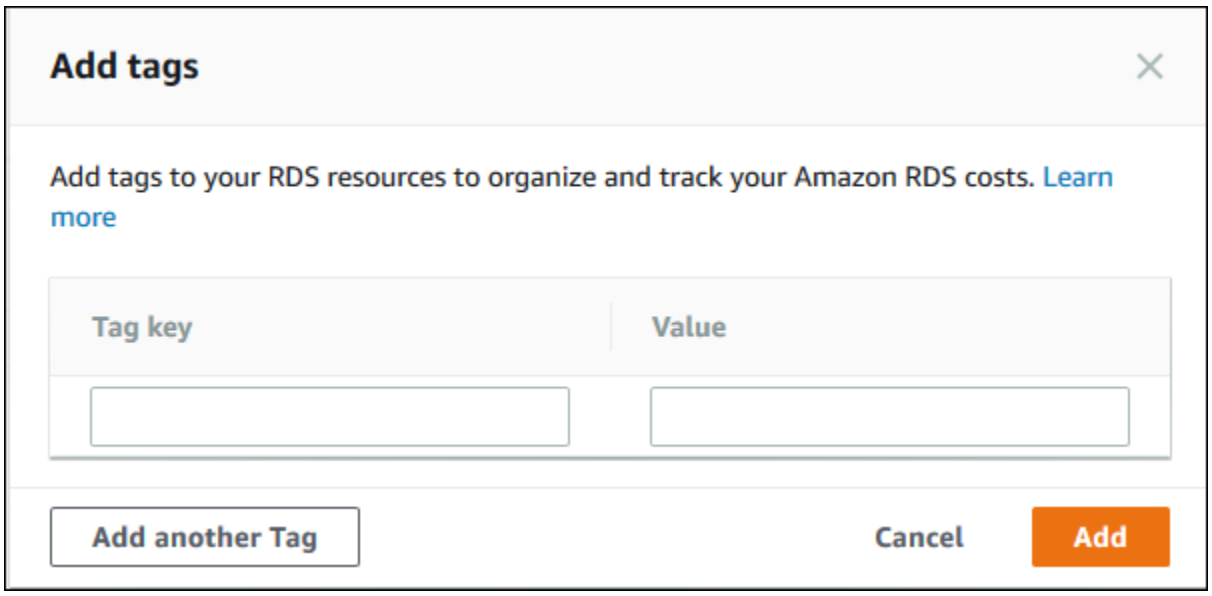
Proses untuk menandai RDS sumber daya Amazon serupa untuk semua sumber daya. Prosedur berikut menunjukkan cara menandai instans Amazon RDS DB.

Untuk menambahkan tag ke instans DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

   > **Note**
   >
   > Untuk memfilter daftar instans DB di panel Basis data, masukkan string teks untuk Filter basis data. Hanya instans DB yang berisi string yang muncul.

3. Pilih nama instans DB yang ingin diberi tag untuk menampilkan detailnya.
4. Di bagian detail, gulir ke bawah hingga bagian Tag.
5. Pilih Tambahkan. Jendela Tambahkan tag akan muncul.



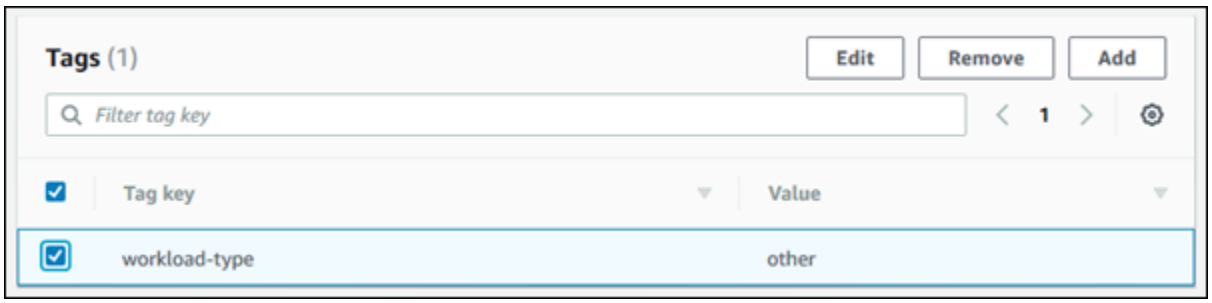

6. Masukkan nilai untuk Kunci tag dan Nilai.
7. Untuk menambahkan tag lain, Anda dapat memilih Tambahkan Tag lain dan memasukkan nilai untuk Kunci tag dan Nilai-nya.

   Ulangi langkah ini seperlunya.
8. Pilih Tambahkan.

Untuk menghapus tag dari instans DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

   > **Note**
   >
   > Untuk memfilter daftar instans DB di panel Basis data, masukkan string teks dalam kotak Filter basis data. Hanya instans DB yang berisi string yang muncul.
3. Pilih nama instans DB untuk menampilkan detailnya.
4. Di bagian detail, gulir ke bawah hingga bagian Tag.
5. Pilih tag yang ingin dihapus.




6. Pilih Hapus, lalu pilih Hapus di jendela Hapus tag.

AWS CLI

Anda dapat menambahkan, mencantumkan, atau menghapus tag untuk instans DB menggunakan AWS CLI.

- Untuk menambahkan satu atau beberapa tag ke RDS sumber daya Amazon, gunakan AWS CLI perintah `add-tags-to-resource`.
- Untuk membuat daftar tag pada RDS sumber daya Amazon, gunakan AWS CLI perintah `list-tags-for-resource`.
- Untuk menghapus satu atau beberapa tag dari RDS sumber daya Amazon, gunakan AWS CLI perintah `remove-tags-from-resource`.

Untuk mempelajari lebih lanjut tentang cara membangun yang diperlukanARN, lihatMembuat konsep ARN untuk Amazon RDS.

RDS API

Anda dapat menambahkan, membuat daftar, atau menghapus tag untuk instans DB menggunakan Amazon RDSAPI.

- Untuk menambahkan tag ke RDS sumber daya Amazon, gunakan `AddTagsToResource`operasi.
- Untuk mencantumkan tag yang ditetapkan ke RDS sumber daya Amazon, gunakan `ListTagsForResource`.
- Untuk menghapus tag dari RDS sumber daya Amazon, gunakan `RemoveTagsFromResource`operasi.

Untuk mempelajari lebih lanjut tentang cara membangun yang diperlukanARN, lihatMembuat konsep ARN untuk Amazon RDS.

Saat bekerja dengan XML menggunakan Amazon RDSAPI, tag menggunakan skema berikut:

```
<Tagging>
      <TagSet>
          <Tag>
              <Key>Project</Key>
              <Value>Trinity</Value>
          </Tag>
          <Tag>
              <Key>User</Key>
              <Value>Jones</Value>
          </Tag>
      </TagSet>
  </Tagging>
```

Tabel berikut menyediakan daftar XML tag yang diizinkan dan karakteristiknya. Nilai untuk `Key` dan `Value` peka huruf besar/kecil. Misalnya, `project=Trinity` dan `PROJECT=Trinity` merupakan tag yang berbeda.

| Elemen pemberian tag | Deskripsi |
| --- | --- |
| TagSet | Kumpulan tag adalah wadah untuk semua tag yang ditetapkan ke RDS sumber daya Amazon. Hanya ada satu rangkaian tag per sumber daya. Anda bekerja dengan TagSet hanya melalui Amazon RDSAPI. |
| Tag | Tag adalah pasangan kunci-nilai yang ditentukan pengguna. Satu rangkaian tag bisa berisi 1 hingga 50 tag. |
| Kunci | Kunci adalah nama tag yang wajib diisi. Untuk pembatasan, lihatStruktur tag di Amazon RDS.<br><br>Nilai string dapat terdiri dari 1 hingga 128 karakter Unicode dan tidak boleh diawali dengan `aws:` atau `rds:`. String hanya boleh berisi kumpulan huruf Unicode, angka, ruang kosong, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). |

| Elemen pemberian tag | Deskripsi |
| --- | --- |
| | Kunci dalam rangkaian tag harus unik. Misalnya, Anda tidak dapat memiliki pasangan kunci dalam set tag dengan kunci yang sama tetapi dengan nilai yang berbeda, seperti. project/Trinity and project/Xanadu |
| Nilai | Nilai adalah nilai tag opsional. Untuk pembatasan, lihatStruktur tag di Amazon RDS.<br><br>Nilai string dapat terdiri dari 1 hingga 256 karakter Unicode dan tidak boleh diawali dengan `aws:` atau `rds:`. String hanya boleh berisi kumpulan huruf Unicode, angka, ruang kosong, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$").<br><br>Nilai rangkaian tag tidak harus unik dan bisa nol. Misalnya, Anda dapat memiliki pasangan kunci-nilai dalam kumpulan tag. project/Trinity and cost-center/Trinity |

## Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

Misalkan Anda membuat sejumlah klaster DB Aurora dalam lingkungan pengembangan atau pengujian. Anda perlu menyimpan semua klaster ini selama beberapa hari. Beberapa klaster menjalankan pengujian di malam hari. Klaster lainnya dapat dihentikan di malam hari dan dimulai lagi keesokan harinya. Contoh berikut menunjukkan cara menetapkan tag untuk klaster yang cocok untuk dihentikan di malam hari. Kemudian contoh berikut menunjukkan bagaimana skrip dapat mendeteksi klaster yang memiliki tag dan kemudian menghentikan klaster tersebut. Dalam contoh ini, porsi nilai dari pasangan kunci-nilai tidaklah penting. Keberadaan tag `stoppable` menandakan bahwa klaster memiliki properti yang ditetapkan pengguna ini.

Untuk menentukan klaster DB Aurora yang akan dihentikan

1. Tentukan cluster ARN yang ingin Anda tetapkan sebagai stoppable.

   Perintah dan APIs untuk penandaan berfungsi denganARNs. Dengan begitu, mereka dapat bekerja dengan lancar di seluruh AWS Wilayah, AWS akun, dan berbagai jenis sumber daya yang mungkin memiliki nama pendek yang identik. Anda dapat menentukan ARN bukan ID

cluster dalam CLI perintah yang beroperasi pada cluster. Ganti nama klaster Anda sendiri *dev-test-cluster*. Dalam perintah berikutnya yang menggunakan ARN parameter, gantikan cluster Anda sendiri. ARN ARNTermasuk ID AWS akun Anda sendiri dan nama AWS Wilayah tempat cluster Anda berada.

```
$ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
  --query "*[].{DBClusterArn:DBClusterArn}" --output text
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
```

2. Tambahkan tag `stoppable` ke klaster ini.

   Pilih nama untuk tag ini. Pendekatan ini berarti Anda dapat menghindari merancang konvensi penamaan yang mengenkode semua informasi yang relevan dalam nama. Dalam konvensi tersebut, Anda dapat mengenkode informasi dalam nama instans DB atau nama-nama sumber daya lainnya. Karena memperlakukan tag sebagai atribut yang ada atau tidak ada, contoh ini menghilangkan bagian `Value=` dari parameter `--tags`.

   ```
   $ aws rds add-tags-to-resource \
     --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
     --tags Key=stoppable
   ```

3. Konfirmasi bahwa tag tersebut ada dalam klaster.

   Perintah ini mengambil informasi tag untuk cluster dalam JSON format dan dalam teks yang dipisahkan tab biasa.

   ```
   $ aws rds list-tags-for-resource \
     --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
   {
       "TagList": [
           {
               "Key": "stoppable",
               "Value": ""
   
           }
       ]
   }
   $ aws rds list-tags-for-resource \
     --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
    text
   TAGLIST stoppable
   ```

4. Untuk menghentikan semua klaster yang ditunjuk sebagai `stoppable`, siapkan daftar semua klaster Anda. Lihat daftar dan periksa apakah setiap klaster diberi tag dengan atribut yang relevan.

   Contoh Linux ini menggunakan shell scripting untuk menyimpan daftar cluster ARNs ke file sementara dan kemudian melakukan CLI perintah untuk setiap cluster.

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
```

Anda dapat menjalankan skrip seperti ini pada akhir setiap hari untuk memastikan bahwa klaster yang tidak penting dihentikan. Anda juga dapat menjadwalkan pekerjaan menggunakan utilitas seperti `cron` melakukan pemeriksaan setiap malam. Misalnya, Anda mungkin melakukan ini jika beberapa klaster dibiarkan berjalan secara tidak sengaja. Di sini, Anda dapat menyesuaikan perintah yang mempersiapkan daftar klaster yang akan diperiksa.

Perintah berikut menghasilkan daftar klaster, tetapi hanya yang berada dalam status `available`. Skrip ini dapat mengabaikan klaster yang sudah dihentikan, karena klaster tersebut akan memiliki nilai status yang berbeda seperti `stopped` atau `stopping`.

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

> **Tip**
>
> Anda dapat menggunakan penetapan tag dan pencarian klaster dengan tag tersebut untuk mengurangi biaya dengan cara lain. Misalnya, gunakan skenario ini dengan klaster DB Aurora yang digunakan untuk pengembangan dan pengujian. Di sini, Anda dapat menunjuk beberapa klaster untuk dihapus di akhir setiap hari, atau hanya instans DB pembaca saja yang dihapus. Atau Anda dapat menunjuk beberapa klaster agar instans DB-nya diubah menjadi kelas instans DB kecil selama waktu penggunaan rendah yang diharapkan.

# Nama Sumber Daya Amazon (ARNs) di Amazon RDS

Sumber daya yang dibuat di Amazon Web Services masing-masing diidentifikasi secara unik dengan Amazon Resource Name ()ARN. Untuk RDS operasi Amazon tertentu, Anda harus mengidentifikasi RDS sumber daya Amazon secara unik dengan menentukannya. ARN Misalnya, saat Anda membuat replika baca instans RDS DB, Anda harus menyediakan instance DB sumber. ARN

Untuk informasi tentang membangun ARN dan mendapatkan yang sudah adaARN, lihat topik berikut.

Topik


- Membuat konsep ARN untuk Amazon RDS
- Mendapatkan yang ada ARN untuk Amazon RDS


## Membuat konsep ARN untuk Amazon RDS

Sumber daya yang dibuat di Amazon Web Services masing-masing diidentifikasi secara unik dengan Amazon Resource Name (ARN). Anda dapat membuat konsep ARN untuk sumber daya Amazon RDS menggunakan sintaks berikut.

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

| Nama Wilayah | Wilayah | Titik Akhir | Protokol | |
|---|---|---|---|---|
| AS Timur (Ohio) | us–east-2 | rds.us-east-2.amazonaws.com | HTTPS | |
| | | rds-fips.us-east-2.api.aws | HTTPS | |
| | | rds.us-east-2.api.aws | HTTPS | |
| | | rds-fips.us-east-2.amazonaws.com | HTTPS | |
| AS Timur (Virginia Utara) | us-east-1 | rds.us-east-1.amazonaws.com | HTTPS | |
| | | rds-fips.us-east-1.api.aws | HTTPS | |
| | | rds-fips.us-east-1.amazonaws.com | HTTPS | |
| | | rds.us-east-1.api.aws | HTTPS | |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol |
|---|---|---|---|
| AS Barat (California Utara) | us-west-1 | rds.us-west-1.amazonaws.com | HTTPS |
| | | rds.us-west-1.api.aws | HTTPS |
| | | rds-fips.us-west-1.amazonaws.com | HTTPS |
| | | rds-fips.us-west-1.api.aws | HTTPS |
| AS Barat (Oregon) | us-west-2 | rds.us-west-2.amazonaws.com | HTTPS |
| | | rds-fips.us-west-2.amazonaws.com | HTTPS |
| | | rds.us-west-2.api.aws | HTTPS |
| | | rds-fips.us-west-2.api.aws | HTTPS |
| Afrika (Cape Town) | af-south-1 | rds.af-south-1.amazonaws.com | HTTPS |
| | | rds.af-south-1.api.aws | HTTPS |
| Asia Pasifik (Hong Kong) | ap-east-1 | rds.ap-east-1.amazonaws.com | HTTPS |
| | | rds.ap-east-1.api.aws | HTTPS |
| Asia Pasifik (Hyderabad) | ap-south-2 | rds.ap-south-2.amazonaws.com | HTTPS |
| | | rds.ap-south-2.api.aws | HTTPS |
| Asia Pasifik (Jakarta) | ap-southeast-3 | rds.ap-southeast-3.amazonaws.com | HTTPS |
| | | rds.ap-southeast-3.api.aws | HTTPS |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol |
|---|---|---|---|
| Asia Pasifik (Malaysia) | ap-southeast-5 | rds.ap-southeast-5.amazonaws.com | HTTPS |
| Asia Pasifik (Melbourne) | ap-southeast-4 | rds.ap-southeast-4.amazonaws.com<br>rds.ap-southeast-4.api.aws | HTTPS<br>HTTPS |
| Asia Pasifik (Mumbai) | ap-south-1 | rds.ap-south-1.amazonaws.com<br>rds.ap-south-1.api.aws | HTTPS<br>HTTPS |
| Asia Pasifik (Osaka) | ap-northeast-3 | rds.ap-northeast-3.amazonaws.com<br>rds.ap-northeast-3.api.aws | HTTPS<br>HTTPS |
| Asia Pasifik (Seoul) | ap-northeast-2 | rds.ap-northeast-2.amazonaws.com<br>rds.ap-northeast-2.api.aws | HTTPS<br>HTTPS |
| Asia Pasifik (Singapura) | ap-southeast-1 | rds.ap-southeast-1.amazonaws.com<br>rds.ap-southeast-1.api.aws | HTTPS<br>HTTPS |
| Asia Pasifik (Sydney) | ap-southeast-2 | rds.ap-southeast-2.amazonaws.com<br>rds.ap-southeast-2.api.aws | HTTPS<br>HTTPS |
| Asia Pasifik (Taipei) | ap-timur-2 | rds.ap-east-2.amazonaws.com | HTTPS |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol |
|---|---|---|---|
| Asia Pasifik (Thailand) | ap-tenggara-7 | rds.ap-southeast-7.amazonaws.com | HTTPS |
| Asia Pacific (Tokyo) | ap-northeast-1 | rds.ap-northeast-1.amazonaws.com<br>rds.ap-northeast-1.api.aws | HTTPS<br>HTTPS |
| Kanada (Pusat) | ca-central-1 | rds.ca-central-1.amazonaws.com<br>rds.ca-central-1.api.aws<br>rds-fips.ca-central-1.api.aws<br>rds-fips.ca-central-1.amazonaws.com | HTTPS<br>HTTPS<br>HTTPS<br>HTTPS |
| Kanada Barat (Calgary) | ca-west-1 | rds.ca-west-1.amazonaws.com<br>rds-fips.ca-west-1.amazonaws.com | HTTPS<br>HTTPS |
| Eropa (Frankfurt) | eu-central-1 | rds.eu-central-1.amazonaws.com<br>rds.eu-central-1.api.aws | HTTPS<br>HTTPS |
| Eropa (Irlandia) | eu-west-1 | rds.eu-west-1.amazonaws.com<br>rds.eu-west-1.api.aws | HTTPS<br>HTTPS |
| Eropa (London) | eu-west-2 | rds.eu-west-2.amazonaws.com<br>rds.eu-west-2.api.aws | HTTPS<br>HTTPS |
| Eropa (Milan) | eu-south-1 | rds.eu-south-1.amazonaws.com<br>rds.eu-south-1.api.aws | HTTPS<br>HTTPS |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol |
|---|---|---|---|
| Eropa (Paris) | eu-west-3 | rds.eu-west-3.amazonaws.com | HTTPS |
| | | rds.eu-west-3.api.aws | HTTPS |
| Eropa (Spanyol) | eu-south-2 | rds.eu-south-2.amazonaws.com | HTTPS |
| | | rds.eu-south-2.api.aws | HTTPS |
| Eropa (Stockholm) | eu-north-1 | rds.eu-north-1.amazonaws.com | HTTPS |
| | | rds.eu-north-1.api.aws | HTTPS |
| Eropa (Zürich) | eu-central-2 | rds.eu-central-2.amazonaws.com | HTTPS |
| | | rds.eu-central-2.api.aws | HTTPS |
| Israel (Tel Aviv) | il-central-1 | rds.il-central-1.amazonaws.com | HTTPS |
| | | rds.il-central-1.api.aws | HTTPS |
| Meksiko (Tengah) | mx-pusat-1 | rds.mx-central-1.amazonaws.com | HTTPS |
| Timur Tengah (Bahrain) | me-south-1 | rds.me-south-1.amazonaws.com | HTTPS |
| | | rds.me-south-1.api.aws | HTTPS |
| Timur Tengah (UAE) | me-central-1 | rds.me-central-1.amazonaws.com | HTTPS |
| | | rds.me-central-1.api.aws | HTTPS |
| Amerika Selatan (Sao Paulo) | sa-east-1 | rds.sa-east-1.amazonaws.com | HTTPS |
| | | rds.sa-east-1.api.aws | HTTPS |

| Nama Wilayah | Wilayah | Titik Akhir | Protokol |
|---|---|---|---|
| AWS GovCloud (AS-Timur) | us-gov-east-1 | rds.us-gov-east-1.amazonaws.com<br>rds.us-gov-east-1.api.aws | HTTPS<br>HTTPS |
| AWS GovCloud (AS-Barat) | us-gov-west-1 | rds.us-gov-west-1.amazonaws.com<br>rds.us-gov-west-1.api.aws | HTTPS<br>HTTPS |

Tabel berikut menunjukkan format yang harus Anda gunakan saat membuat konsep ARN untuk jenis sumber daya Amazon RDS tertentu.

| Jenis sumber daya | Format ARN |
|---|---|
| Instans DB | arn:aws:rds: ::db: *<region>* *<account>* *<name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1` |
| Klaster DB | arn:aws:rds: ::cluster: *<region>* *<account>* *<name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :cluster: my-aurora-cluster-1` |
| Langganan peristiwa | arn:aws:rds: ::es: *<region>* *<account>* *<name>*<br>Misalnya: |

| Jenis sumber daya | Format ARN |
| --- | --- |
| | `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription` |
| Grup parameter DB | arn:aws:rds: ::pg: *<region> <account> <name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs` |
| Grup parameter klaster DB | arn:aws:rds: ::cluster-pg: *<region> <account> <name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone` |
| Instans DB yang dicadangkan | arn:aws:rds: ::ri: *<region> <account> <name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :ri:my-reserved-postgresql` |
| Grup keamanan DB | arn:aws:rds: ::secgrp: *<region> <account> <name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :secgrp:my-public` |
| Snapshot DB otomatis | arn:aws:rds: ::snapshot: rds: *<region> <account> <name>*<br>Misalnya:<br>`arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23` |

| Jenis sumber daya | Format ARN |
| --- | --- |
| Snapshot klaster DB otomatis | arn:aws:rds: ::cluster-snapshot: rds: *<region>* *<account>* *<name>*<br><br>Misalnya:<br><br>`arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16` |
| Snapshot DB manual | arn:aws:rds: ::snapshot: *<region>* *<account>* *<name>*<br><br>Misalnya:<br><br>`arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap` |
| Snapshot klaster DB manual | arn:aws:rds: ::cluster-snapshot: *<region>* *<account>* *<name>*<br><br>Misalnya:<br><br>`arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap` |
| Grup subnet DB | arn:aws:rds: ::subgrp: *<region>* *<account>* *<name>*<br><br>Misalnya:<br><br>`arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10` |

## Mendapatkan yang ada ARN untuk Amazon RDS

Anda bisa mendapatkan ARN RDS sumber daya dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau RDSAPI.

Konsol

Untuk mendapatkan ARN dari AWS Management Console, navigasikan ke sumber daya yang ARN Anda inginkan, dan lihat detail untuk sumber daya itu.

Misalnya, Anda bisa mendapatkan cluster ARN for DB dari tab Konfigurasi detail cluster DB.

AWS CLI

Untuk mendapatkan ARN dari RDS sumber daya tertentu, Anda menggunakan `describe` perintah untuk sumber daya itu. AWS CLI Tabel berikut menunjukkan setiap AWS CLI perintah, dan ARN properti yang digunakan dengan perintah untuk mendapatkan perintahARN.

| AWS CLI perintah | ARNproperti |
|---|---|
| describe-event-subscriptions | EventSubscriptionArn |
| describe-certificates | CertificateArn |
| describe-db-parameter-groups | DBParameterGroupArn |
| describe-db-cluster-parameter-kelompok | DBClusterParameterGroupArn |
| describe-db-instances | DBInstanceArn |
| describe-db-security-groups | DBSecurityGroupArn |
| describe-db-snapshots | DBSnapshotArn |
| describe-events | SourceArn |
| describe-reserved-db-instances | R eservedDBInstance Arn |
| describe-db-subnet-groups | DBSubnetGroupArn |
| describe-db-clusters | DBClusterArn |
| describe-db-cluster-snapshots | DBClusterSnapshotArn |

Misalnya, AWS CLI perintah berikut mendapatkan instance ARN untuk DB.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Untuk Windows:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Output dari perintah tersebut adalah sebagai berikut:

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS API

ARNUntuk mendapatkan RDS sumber daya tertentu, Anda dapat memanggil RDS API operasi berikut dan menggunakan ARN properti yang ditunjukkan berikut.

| RDSAPIoperasi | ARNproperti |
|---|---|
| DescribeEventSubscriptions | EventSubscriptionArn |
| DescribeCertificates | CertificateArn |
| escribeDBParameterGrup D | DBParameterGroupArn |
| D escribeDBCluster Parameter Groups | DBClusterParameterGroupArn |

| RDSAPIoperasi | ARNproperti |
| --- | --- |
| D escribeDBInstances | DBInstanceArn |
| escribeDBSecurityGrup D | DBSecurityGroupArn |
| D escribeDBSnapshots | DBSnapshotArn |
| DescribeEvents | SourceArn |
| DescribeReservedDBInstances | R eservedDBInstance Arn |
| escribeDBSubnetGrup D | DBSubnetGroupArn |
| D escribeDBClusters | DBClusterArn |
| D escribeDBCluster Cuplikan | DBClusterSnapshotArn |

# Pembaruan Amazon Aurora

Amazon Aurora merilis pembaruan secara berkala. Pembaruan diterapkan pada klaster DB Amazon Aurora selama periode pemeliharaan sistem. Waktu saat pembaruan diterapkan tergantung pada pengaturan Region dan jendela pemeliharaan untuk cluster DB, dan juga jenis pembaruan. Pembaruan memerlukan pengaktifan ulang basis data, sehingga Anda biasanya mengalami waktu henti selama 20–30 detik. Setelah waktu henti ini, Anda dapat lanjut menggunakan klaster DB Anda. Anda dapat melihat atau mengubah pengaturan periode pemeliharaan dari AWS Management Console.

> **Note**
>
> Waktu yang diperlukan untuk melakukan boot ulang instans DB Anda bergantung pada proses pemulihan crash, aktivitas basis data pada saat boot ulang, dan perilaku mesin DB spesifik Anda. Untuk mempersingkat waktu boot ulang, sebaiknya kurangi aktivitas basis data sebanyak mungkin selama proses boot ulang. Pengurangan aktivitas basis data akan menurunkan aktivitas rollback untuk transaksi saat bergerak.

Untuk informasi tentang pembaruan sistem operasi untuk Amazon Aurora, lihat Pembaruan sistem operasi untuk cluster Aurora DB.

Beberapa pembaruan dikhususkan untuk mesin basis data yang didukung Aurora. Untuk informasi selengkapnya tentang pembaruan mesin basis data, lihat tabel berikut.

| Mesin basis data | Pembaruan |
| --- | --- |
| Amazon Aurora Saya SQL | Lihat Pembaruan mesin database untuk Amazon Aurora My SQL |
| Amazon Aurora Postgre SQL | Lihat Pembaruan mesin database untuk Amazon Aurora Postgre SQL |

## Mengidentifikasi versi Amazon Aurora Anda

Amazon Aurora menyertakan fitur tertentu yang bersifat umum untuk Aurora dan tersedia untuk semua klaster DB Aurora. Aurora mencakup fitur lain yang khusus untuk mesin basis data tertentu

yang didukung Aurora. Fitur-fitur ini hanya tersedia untuk cluster Aurora DB yang menggunakan mesin database itu, seperti Aurora Postgre. SQL

Instans DB Aurora memberikan dua nomor versi, yaitu nomor versi Aurora dan nomor versi mesin basis data Aurora. Nomor versi Aurora menggunakan format berikut.

```
<major version>.<minor version>.<patch version>
```

Untuk mendapatkan nomor versi Aurora dari instans DB Aurora menggunakan mesin basis data tertentu, gunakan salah satu kueri berikut.

| Mesin basis data | Pertanyaan |
| --- | --- |
| Amazon Aurora Saya SQL | `SELECT AURORA_VERSION();`<br>`SHOW @@aurora_version;` |
| Amazon Aurora Postgre SQL | `SELECT AURORA_VERSION();` |

# Amazon RDS Extended Support dengan

RDS Extended Support memungkinkan Anda untuk terus menjalankan database pada versi mesin utama melewati akhir tanggal dukungan standar dengan biaya tambahan.

Anda hanya dapat mendaftarkan database di RDS Extended Support dengan mengaktifkan RDS Extended Support saat pertama kali membuat atau memulihkan instans DB. Anda tidak dapat memperbarui status pendaftaran RDS Extended Support pada instans DB yang ada kecuali Anda memulihkannya.

Jika Anda mengaktifkan RDS Extended Support selama pembuatan atau pemulihan instans DB, maka setelah RDS Amazon Aurora Extended Support. Pendaftaran otomatis ke RDS Extended Support tidak mengubah mesin database dan tidak memengaruhi waktu aktif atau kinerja instans DB Anda.

RDS Extended Support menyediakan pembaruan dan dukungan teknis berikut:

- Pembaruan keamanan penting dan tinggi CVEs untuk instans DB atau cluster DB Anda, termasuk mesin database
- Perbaikan bug dan tambalan untuk masalah kritis
- Kemampuan untuk membuka kasus dukungan dan menerima bantuan pemecahan masalah dalam perjanjian tingkat layanan Amazon RDS standar

Penawaran berbayar ini memberi Anda lebih banyak waktu untuk meningkatkan ke versi mesin utama yang didukung. Misalnya, akhir tanggal dukungan standar Aurora untuk Aurora MySQL versi 2 adalah 31 Oktober 2024. Namun, Anda belum siap untuk meng-upgrade secara manual ke Aurora MySQL versi 3 sebelum tanggal tersebut. Dalam hal ini, Amazon Aurora secara otomatis mendaftarkan klaster Anda di RDS Extended Support pada 31 Oktober 2024, dan Anda dapat terus menjalankan Aurora MySQL versi 2. Mulai 1 Desember 2024, Amazon Aurora secara otomatis menagih Anda untuk RDS Extended Support.

RDS Extended Support tersedia hingga 3 tahun setelah tanggal akhir kehidupan komunitas RDS tanggal untuk versi mesin utama (3 tahun dan 4 bulan untuk Aurora MySQL versi 2). Setelah waktu ini, jika Anda belum memutakhirkan versi mesin utama Anda ke versi yang didukung, Aurora akan secara otomatis meningkatkan versi mesin utama Anda. Kami menyarankan agar Anda meningkatkan ke versi mesin utama sesegera mungkin.

Untuk informasi selengkapnya tentang akhir Aurora dari tanggal dukungan standar dan akhir RDS dari tanggal Dukungan Diperpanjang, lihat Kalender rilis untuk versi utama Aurora MySQL dan kalender Rilis untuk versi utama Aurora PostgreSQL.

Topik


- Ikhtisar Amazon RDS Extended Support
- Biaya Amazon RDS Extended Support
- Versi dengan Amazon RDS Extended Support
- Aurora dan tanggung jawab pelanggan dengan Amazon RDS Extended Support
- Aurora DB atau cluster global dengan Amazon RDS Extended Support
- Melihat pendaftaran klaster global di Amazon RDS Extended Support
- Melihat tanggal dukungan untuk versi engine di Amazon RDS Extended Support
- Aurora DB atau cluster global dengan Amazon Extended Support RDS


# Ikhtisar Amazon RDS Extended Support

Setelah Aurora berakhir dari tanggal dukungan standar, jika Anda tidak menonaktifkan RDS Extended Support selama pembuatan atau pemulihan instans DB Anda di RDS Extended Support. Aurora secara otomatis memutakhirkan instans DB Anda ke versi minor terakhir yang dirilis sebelum tanggal dukungan standar Aurora berakhir, jika Anda belum menjalankan versi tersebut. Amazon Aurora tidak akan memutakhirkan versi minor Anda sampai setelah Aurora berakhir dari tanggal dukungan standar untuk versi mesin utama Anda.

Anda dapat membuat database baru dengan versi mesin utama yang telah mencapai akhir dari tanggal dukungan standar. Aurora secara otomatis mendaftarkan database baru ini di RDS Extended Support dan menagih Anda untuk penawaran ini.

Jika Anda meningkatkan ke mesin yang masih di bawah dukungan standar Aurora sebelum RDS Aurora berakhir dari tanggal dukungan standar, Amazon Amazon Aurora tidak akan mendaftarkan mesin Anda di Extended Support.

Jika pemulihan gagal, Aurora akan secara otomatis mendaftarkan mesin Anda di RDS Extended Support dengan versi yang kompatibel dengan snapshot.

Anda dapat mengakhiri pendaftaran di RDS Extended Support kapan saja.

Untuk informasi selengkapnya tentang akhir Aurora dari tanggal dukungan standar dan akhir RDS dari tanggal Dukungan Diperpanjang, lihat Kalender rilis untuk versi utama Aurora MySQL dan kalender Rilis untuk versi utama Aurora PostgreSQL.

# Biaya Amazon RDS Extended Support

Anda akan dikenakan biaya untuk semua mesin yang terdaftar di Extended RDS Support mulai hari setelah Aurora berakhir pada tanggal dukungan standar. Untuk akhir Aurora dari tanggal dukungan standar, lihat. Versi mayor Amazon Aurora

Biaya tambahan untuk RDS Extended Support secara otomatis berhenti ketika Anda melakukan salah satu tindakan berikut:

- Tingkatkan ke versi mesin yang tercakup dalam dukungan standar.
- Hapus database yang menjalankan versi utama melewati akhir Aurora dari tanggal dukungan standar.

Pengisian daya akan dimulai kembali jika versi mesin target Anda memasuki RDS Extended Support di masa mendatang.

Misalnya, Postgre SQL 11 memasuki Extended Support pada 1 Maret 2024, tetapi biaya tidak dimulai hingga 1 April 2024. Anda RDS meningkatkan SQL database Postgre 11 Anda untuk Postgre . SQL Anda hanya akan dikenakan biaya selama 30 hari Extended Support pada RDS Postgre 11. SQL Anda terus menjalankan Postgre SQL 12 pada instans DB ini melewati RDS akhir tanggal dukungan standar 28 Februari 2025. Database Anda akan kembali dikenakan biaya RDS Extended Support mulai tanggal 1 Maret 2025.

Untuk informasi selengkapnya, lihat Amazon Aurora.

## Menghindari biaya untuk Amazon RDS Extended Support

Anda dapat menghindari biaya untuk RDS Extended Support dengan mencegah Aurora membuat atau memulihkan DB atau cluster global melewati akhir Aurora dari tanggal dukungan standar. Untuk melakukan ini, gunakan AWS CLI atau RDSAPI.

Dalam AWS CLI, `open-source-rds-extended-support-disabled` tentukan `--engine-lifecycle-support` opsi. Dalam RDSAPI, `open-source-rds-extended-support-disabled` tentukan `LifeCycleSupport` parameter. Untuk informasi selengkapnya, lihat Aurora DB atau cluster global atau Aurora DB atau cluster global.

# Versi dengan Amazon RDS Extended Support

RDS Extended Support tersedia untuk Aurora MySQL dan untuk Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Versi mayor Amazon Aurora.

RDS Extended Support hanya tersedia pada versi minor tertentu. Versi minor hanya diindikasikan sebagai pendukung RDS Extended Support setelah versi utama mencapai tanggal akhir kehidupan Komunitas. Untuk informasi selengkapnya, lihat Kalender rilis untuk Aurora MySQL di Catatan Rilis Aurora MySQL dan kalender Rilis untuk Aurora PostgreSQL di Catatan Rilis PostgreSQL Aurora.

RDS Extended Support hanya tersedia diAurora Serverless v2. Itu tidak tersedia diAurora Serverless v1.

Anda juga dapat melihat informasi tentang tanggal dukungan untuk versi engine dengan menggunakan AWS CLI atau RDS API. Untuk informasi selengkapnya, lihat Melihat tanggal dukungan untuk versi engine di Amazon RDS Extended Support.

# Aurora dan tanggung jawab pelanggan dengan Amazon RDS Extended Support

Konten berikut menjelaskan tanggung jawab Aurora dan tanggung jawab Anda dengan Extended RDS Support.

Topik

- Amazon Aurora Amazon
- Tanggung jawab Anda

## Amazon Aurora Amazon

Setelah Aurora berakhir pada tanggal dukungan standar, Amazon Aurora akan menyediakan patch, perbaikan bug, dan upgrade untuk engine yang terdaftar di Extended Support. RDS Ini akan terjadi hingga 3 tahun, atau sampai Anda berhenti menggunakan mesin, mana yang terjadi lebih dulu.

Tambalan akan untuk Kritis dan Tinggi CVEs seperti yang didefinisikan oleh peringkat CVSS keparahan National Vulnerability Database (NVD). Lihat informasi yang lebih lengkap di Metrik Kerentanan.

## Tanggung jawab Anda

Anda bertanggung jawab untuk menerapkan tambalan, perbaikan bug, dan peningkatan yang diberikan untuk yang terdaftar di Extended Support. RDS Amazon Aurora berhak untuk mengubah, mengganti, atau menarik tambalan, perbaikan bug, dan peningkatan tersebut kapan saja. Jika tambalan diperlukan untuk mengatasi masalah keamanan atau stabilitas kritis, Aurora berhak memperbarui atau cluster global dengan tambalan, atau mengharuskan Anda menginstal tambalan.

Anda juga bertanggung jawab untuk meningkatkan mesin Anda ke versi engine yang lebih baru sebelum RDS akhir tanggal Extended Support. RDSAkhir tanggal Extended Support biasanya 3 tahun setelah akhir kehidupan komunitas. . Untuk RDS akhir tanggal Extended Support untuk versi mesin utama database Anda, lihatVersi mayor Amazon Aurora.

Jika Anda tidak memutakhirkan mesin Anda, maka setelah tanggal Extended Support RDS berakhir, Amazon Aurora akan mencoba meningkatkan mesin Anda ke versi mesin yang lebih baru yang didukung di bawah dukungan standar Aurora . Jika pemutakhiran gagal, Amazon Aurora berhak untuk menghapus DB cluster atau cluster global yang menjalankan mesin melewati akhir Aurora dari tanggal dukungan standar. Namun, sebelum melakukannya, Aurora akan menyimpan data Anda dari mesin itu.

# Aurora DB atau cluster global dengan Amazon RDS Extended Support

Saat Anda Aurora DB atau cluster global, pilih Enable RDS Extended Support di konsol, atau gunakan opsi Extended Support AWS CLI di atau parameter di. RDS API Saat Anda mendaftarkan atau klaster global di Amazon Extended RDS Support, instans tersebut terdaftar secara permanen di RDS Extended Support selama masa pakai atau klaster global.

Jika Anda menggunakan konsol, Anda harus memilih Enable RDS Extended Support. Pengaturan tidak dipilih secara default.

Jika Anda menggunakan AWS CLI atau RDS API dan tidak menentukan pengaturan RDS Extended Support, Amazon secara RDS default mengaktifkan Extended Support. RDS Ketika Anda mengotomatisasi dengan menggunakan AWS CloudFormationatau layanan lain, perilaku default ini mempertahankan ketersediaan database Anda melewati akhir Aurora dari tanggal dukungan standar.

Anda dapat mencegah pendaftaran di Extended RDS Support dengan menggunakan AWS CLIatau RDSAPIuntuk membuat atau cluster global.

Topik


- RDSPerilaku Support yang Diperpanjang
- Pertimbangan untuk RDS Extended Support
- Aurora DB atau cluster global dengan Extended Support RDS


## RDSPerilaku Support yang Diperpanjang

Tabel berikut merangkum apa yang terjadi ketika versi mesin utama mencapai ujung dukungan standar Aurora.

| RDSStatus Support Diperpanjang* | Perilaku |
| --- | --- |
| Diaktifkan | Amazon RDS menagih Anda untuk RDS Extended Support. |
| Nonaktif | Amazon RDS memutakhirkan DB atau cluster global ke versi engine yang didukung. Upgrade ini berlangsung pada atau segera setelah Aurora berakhirnya tanggal dukungan standar. |

* Di RDS konsol, status RDS Extended Support muncul sebagai Ya atau Tidak. Di AWS CLI atau RDSAPI, status RDS Extended Support muncul sebagai `open-source-rds-extended-support` atau`open-source-rds-extended-support-disabled`.

## Pertimbangan untuk RDS Extended Support

Sebelum Aurora DB atau cluster global, pertimbangkan item berikut:

- Setelah akhir Aurora dari tanggal dukungan standar telah berlalu, Anda dapat mencegah pembuatan baru atau cluster global baru dan menghindari biaya Extended Support. RDS Untuk melakukan ini, gunakan AWS CLI atau RDSAPI. Dalam AWS CLI, `open-source-rds-extended-support-disabled` tentukan `--engine-lifecycle-support` opsi. Dalam RDSAPI, `open-source-rds-extended-support-disabled` tentukan `LifeCycleSupport` parameter. Jika Anda menentukan `open-source-rds-extended-support-disabled` dan akhir Aurora dari tanggal dukungan standar telah berlalu, membuat DB atau cluster global akan selalu gagal.

- RDSExtended Support diatur pada tingkat cluster. Anggota cluster akan selalu memiliki pengaturan yang sama untuk RDS Extended Support `--engine-lifecycle-support` di RDS konsol, di AWS CLI, dan `EngineLifecycleSupport` di RDSAPI.

Untuk informasi selengkapnya, lihat Versi Amazon Aurora.

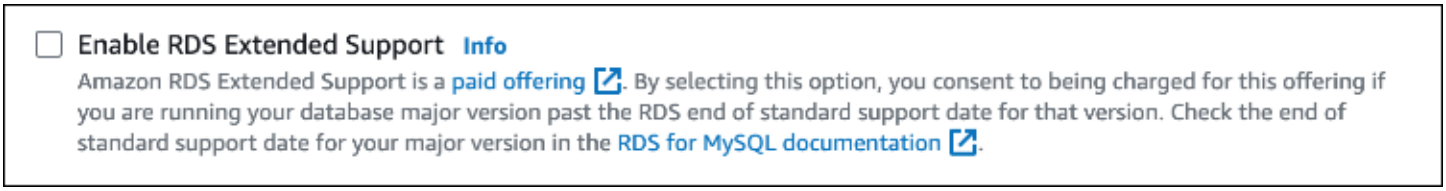
## Aurora DB atau cluster global dengan Extended Support RDS

Anda dapat Aurora DB atau cluster global dengan versi RDS Extended Support menggunakan AWS Management Console,, AWS CLI atau. RDS API

Konsol

Saat Anda membuat cluster Aurora DB atau cluster global instans DB atau cluster , di bagian opsi Engine, pilih Enable Extended RDS Support. Pengaturan ini tidak dipilih secara default.

Gambar berikut menunjukkan pengaturan Enable RDS Extended Support:




AWS CLI

Saat Anda menjalankan AWS CLI perintah create-db-clusteror or create-global-clustercreate-db-instance, pilih RDS Extended Support dengan menentukan `open-source-rds-extended-support` opsi tersebut`--engine-lifecycle-support`. Secara default, opsi ini diatur keo`pen-source-rds-extended-support`.

Untuk mencegah pembuatan Aurora DB baru atau cluster global setelah akhir Aurora dari tanggal dukungan standar, `open-source-rds-extended-support-disabled` tentukan opsi. `--engine-lifecycle-support` Dengan demikian, Anda akan menghindari biaya RDS Extended Support terkait.

RDS API

Saat Anda menggunakan RDS API operasi Amazon C reateDBCluster atau CreateGlobalClusterC reateDBInstance atau , pilih RDS Extended Support dengan menyetel `EngineLifecycleSupport`

parameternya. `open-source-rds-extended-support` Secara default, parameter ini diatur keo`pen-source-rds-extended-support`.

Untuk mencegah pembuatan Aurora DB baru atau cluster global setelah akhir Aurora dari tanggal dukungan standar, `open-source-rds-extended-support-disabled` tentukan parameternya. `EngineLifecycleSupport` Dengan demikian, Anda akan menghindari biaya RDS Extended Support terkait.

Untuk informasi selengkapnya, lihat topik berikut.

- Untuk membuat cluster Aurora DB, ikuti instruksi untuk mesin DB Anda. Membuat klaster DB Amazon Aurora
- Untuk membuat cluster global, ikuti petunjuk untuk mesin DB Anda diMembuat basis data global Amazon Aurora.

## Melihat pendaftaran klaster global di Amazon RDS Extended Support

Anda dapat melihat pendaftaran global di RDS Extended Support menggunakan, the, atau RDS API. AWS Management Console AWS CLI

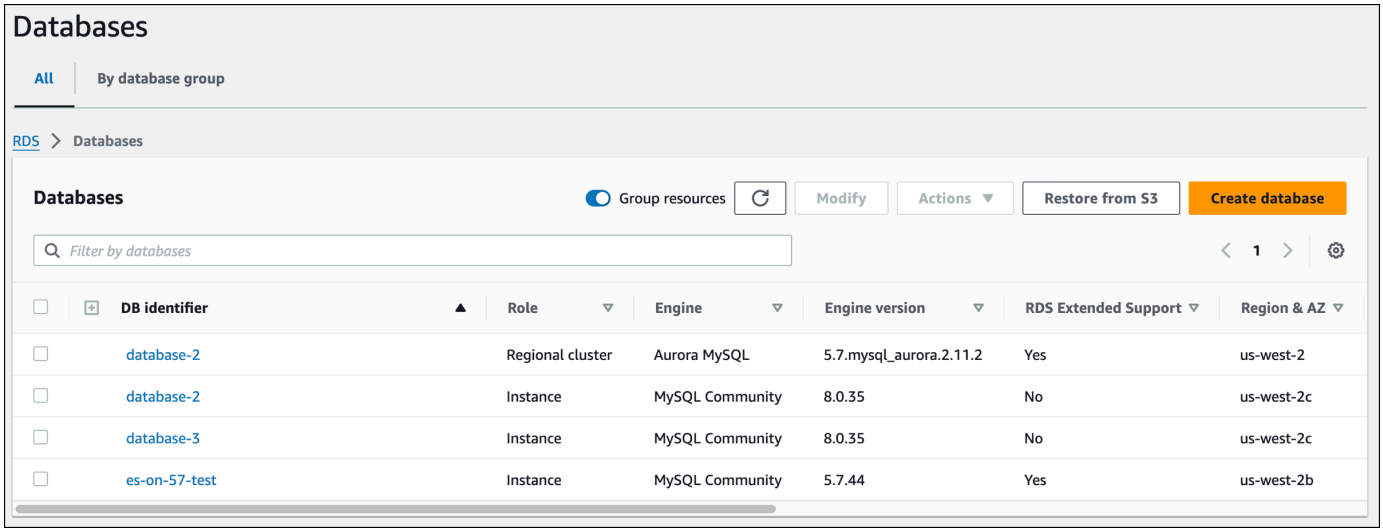
> **Note**
>
> Kolom RDS Extended Support di konsol, `-engine-lifecycle-support` opsi - di AWS CLI, dan `EngineLifecycleSupport` parameter di RDS API hanya menunjukkan pendaftaran di RDS Extended Support. Biaya untuk RDS Extended Support hanya dimulai ketika versi mesin DB Anda telah mencapai akhir Aurora dari dukungan standar. Untuk informasi selengkapnya, lihat Versi mayor Amazon Aurora.
> Misalnya, Anda memiliki database Aurora PostgreSQL 11 yang terdaftar di RDS Extended Support. Pada tanggal 1 April 2024, Amazon RDS mulai menagih Anda untuk RDS Extended Support untuk database ini. Pada 31 Juli 2024, Anda meningkatkan database ini ke Aurora PostgreSQL 12. Status RDS Extended Support untuk database ini tetap diaktifkan. Namun, biaya RDS Extended Support untuk database ini berhenti karena Aurora PostgreSQL 12 belum mencapai akhir dukungan standar Aurora. Amazon RDS tidak akan menagih Anda untuk RDS Extended Support untuk database ini hingga 1 Maret 2025, saat itulah dukungan standar Aurora berakhir untuk Aurora PostgreSQL 12.

## Konsol

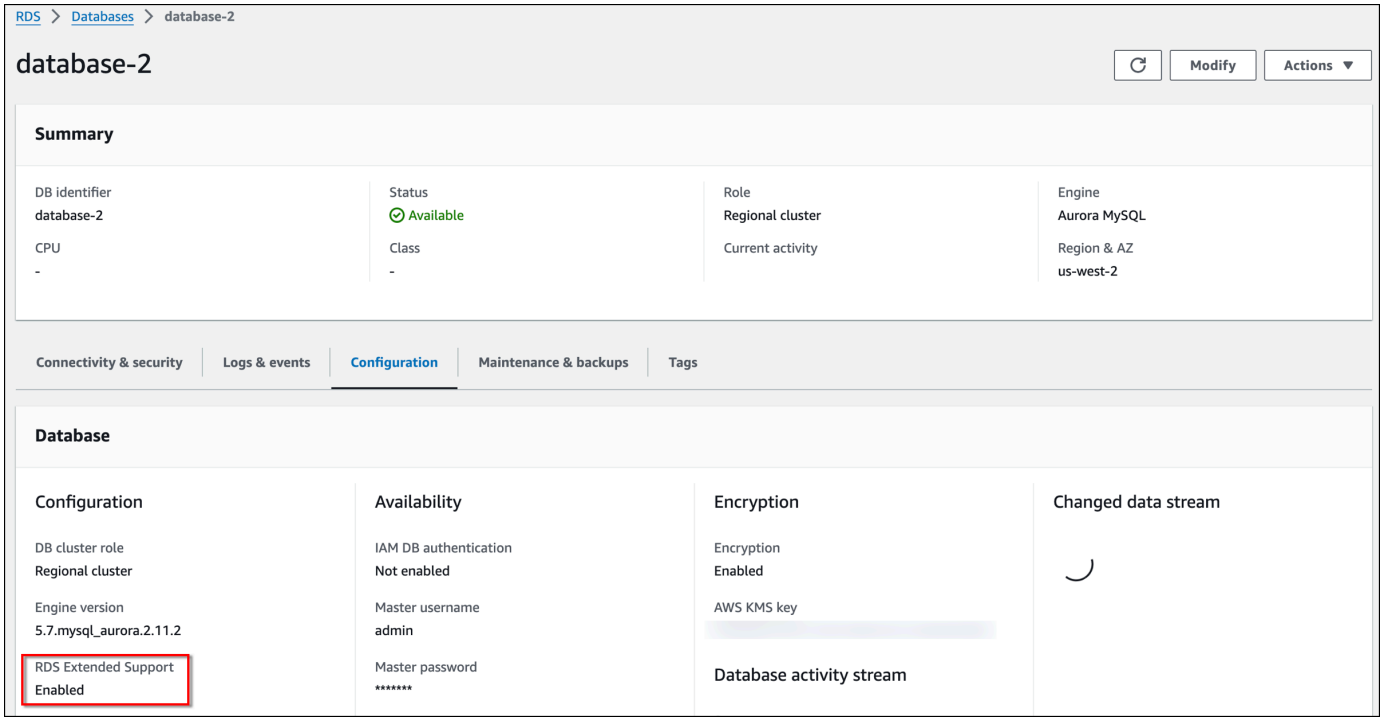
Untuk melihat pendaftaran di RDS Extended Support

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data. Nilai di bawah RDS Extended Support menunjukkan jika Aurora DB atau cluster global terdaftar di RDS Extended Support. Jika tidak ada nilai yang muncul, maka RDS Extended Support tidak tersedia untuk database Anda.

> **Tip**
>
> Jika kolom RDS Extended Support tidak muncul, pilih ikon Preferensi, lalu aktifkan RDS Extended Support.




3. Anda juga dapat melihat pendaftaran pada tab Konfigurasi untuk setiap database. Pilih database di bawah pengenal DB. Pada tab Configuration, lihat di bawah Extended Support untuk melihat apakah database terdaftar atau tidak.




## AWS CLI

Untuk melihat pendaftaran database Anda di RDS Extended Support dengan menggunakan AWS CLI, jalankan describe-db-clustersperintah describe-global-clustersdescribe-db-instances DB cluster).

Jika RDS Extended Support tersedia untuk database, maka responsnya menyertakan parameter`EngineLifecycleSupport`. Nilai tersebut `open-source-rds-extended-support` menunjukkan bahwa Aurora DB atau cluster global terdaftar di RDS Extended Support. Nilai tersebut `open-source-rds-extended-support-disabled` menunjukkan bahwa pendaftaran global di RDS Extended Support dinonaktifkan.

Contoh

Perintah berikut mengembalikan informasi untuk semua cluster Aurora DB Anda:

```
aws rds describe-db-clusters
```

Tanggapan berikut menunjukkan bahwa mesin Aurora PostgreSQL yang berjalan pada cluster Aurora DB terdaftar di RDS Extended Support: `database-1`

```
{
```

```
    "DBClusterIdentifier": "database-1",
    ...
    "Engine": "aurora-postgresql",
    ...
    "EngineLifecycleSupport": "open-source-rds-extended-support"
}
```

### API RDS

Jika RDS Extended Support tersedia untuk database, maka responsnya menyertakan parameter`EngineLifecycleSupport`. Nilai tersebut `open-source-rds-extended-support` menunjukkan bahwa Aurora DB atau cluster global terdaftar di RDS Extended Support. Nilai tersebut `open-source-rds-extended-support-disabled` menunjukkan bahwa pendaftaran global di RDS Extended Support dinonaktifkan.

## Melihat tanggal dukungan untuk versi engine di Amazon RDS Extended Support

Anda dapat melihat informasi tentang tanggal dukungan untuk versi engine untuk atau klaster global di Amazon RDS Extended Support dengan menggunakan atau RDS API. AWS CLI Informasi ini dapat membantu Anda merencanakan peningkatan.

AWS CLI perintah dan operasi RDS API mengembalikan tanggal mulai dan berakhir untuk dukungan standar RDS Extended Support. Tanggal-tanggal ini juga dapat ditemukan di tabel versi mesin utama. Untuk informasi lebih lanjut, lihatVersi mayor Amazon Aurora.

### AWS CLI

Untuk melihat tanggal mulai dan berakhir untuk dukungan standar Aurora dan RDS Extended Support untuk versi mesin utama Anda dengan AWS CLI menggunakan, describe-db-major-enginejalankan perintah -versions.

Perintah ini mengembalikan parameter yang relevan berikut:

- `SupportedEngineLifecycles`— Parameter ini adalah array objek yang mencakup`LifecycleSupportName`,`LifecycleSupportStartDate`, dan`LifecycleSupportEndDate`.

- `LifecycleSupportName`— Parameter ini menunjukkan jenis dukungan versi mesin: Extended Support `open-source-rds-extended-support` ().
- `LifecycleSupportStartDate`— Parameter ini mencantumkan tanggal mulai untuk dukungan standar Aurora atau RDS Extended Support untuk versi mesin utama, tergantung pada nilai. `LifecycleSupportName`
- `LifecycleSupportEndDate`— Parameter ini mencantumkan tanggal akhir untuk dukungan standar Aurora atau RDS Extended Support untuk versi mesin utama, tergantung pada nilai. `LifecycleSupportName`

Contoh

Contoh respons menunjukkan tanggal mulai dan berakhir untuk siklus hidup mesin yang didukung `open-source-rds-standard-support` dan `open-source-rds-extended-support` untuk Aurora MySQL versi 2 (MySQL 5.7). RDS Extended Support tersedia untuk Aurora MySQL versi 2 (MySQL 5.7).

```
{
    "DBMajorEngineVersions": [
        {
            "Engine": "aurora-mysql",
            "MajorEngineVersion": "5.7",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2018-02-06T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2024-10-31T23:59:59.999000+00:00"
                },
                {
                    "LifecycleSupportName": "open-source-rds-extended-support",
                    "LifecycleSupportStartDate": "2024-11-01T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2027-02-28T23:59:59.999000+00:00"
                }
            ]
        },
        ...
    ]
}
```

### API RDS

Untuk melihat tanggal mulai dan berakhirnya dukungan standar Aurora dan RDS Extended Support untuk versi engine utama Anda dengan menggunakan RDS API, gunakan operasi Deskripsikan. DBMajor EngineVersions

Jika RDS Extended Support tersedia untuk versi engine, maka responsnya menyertakan parameter `SupportedEngineLifeCycles` sebagai array dengan dua objek. Satu objek termasuk tanggal mulai dan berakhir untuk dukungan standar Aurora. Objek kedua mencakup tanggal mulai dan berakhir untuk RDS Extended Support.

Jika RDS Extended Support tidak tersedia untuk versi engine, maka respons hanya menyertakan parameter `SupportedEngineLifeCycles` sebagai array dengan satu objek. Objek ini mencakup tanggal mulai dan berakhir untuk dukungan standar Aurora.

## Aurora DB atau cluster global dengan Amazon Extended Support RDS

Saat memulihkan Aurora DB atau cluster global, pilih Enable RDS Extended Support di konsol, atau gunakan opsi Extended Support AWS CLI di atau parameter di. RDS API Saat Anda mendaftarkan Aurora DB atau klaster global di Extended RDS Support, instans ini terdaftar secara permanen di RDS Extended Support selama masa pakai global.

Default untuk pengaturan RDS Extended Support tergantung pada apakah Anda menggunakan konsol AWS CLI, konsol, atau RDS API untuk memulihkan database Anda. Jika Anda menggunakan konsol, Anda tidak memilih Enable RDS Extended Support, dan versi mesin utama yang Anda pulihkan melewati akhir Aurora dari dukungan standar, lalu Amazon Aurora secara otomatis meningkatkan instans DB Anda ke versi engine yang lebih baru. Jika Anda menggunakan AWS CLI atau RDS API dan Anda tidak menentukan pengaturan RDS Extended Support, Amazon secara RDS default mengaktifkan Extended Support. RDS Ketika Anda mengotomatisasi dengan menggunakan AWS CloudFormationatau layanan lain, perilaku default ini mempertahankan ketersediaan database Anda melewati akhir Aurora dari tanggal dukungan standar. Anda dapat menonaktifkan RDS Extended Support dengan menggunakan AWS CLI atau file RDSAPI.

Topik

- RDSPerilaku Support yang Diperpanjang
- Pertimbangan untuk RDS Extended Support

- Kembalikan Aurora DB atau cluster global dengan Extended Support RDS

## RDSPerilaku Support yang Diperpanjang

Tabel berikut merangkum apa yang terjadi ketika versi mesin utama dari Aurora DB atau cluster global yang Anda pulihkan telah mencapai akhir Aurora dari dukungan standar.

| RDSStatus Support Diperpanjang* | Perilaku |
|---|---|
| Diaktifkan | Amazon RDS menagih Anda untuk RDS Extended Support. |
| Dinonaktifkan | Setelah pemulihan selesai, Amazon RDS secara otomatis memutakhirkan atau cluster global ke versi engine yang lebih baru (dalam jendela pemeliharaan masa depan). |

* Di RDS konsol, status RDS Extended Support muncul sebagai Ya atau Tidak. Di AWS CLI atau RDSAPI, status RDS Extended Support muncul sebagai `open-source-rds-extended-support` atau`open-source-rds-extended-support-disabled`.

## Pertimbangan untuk RDS Extended Support

Sebelum memulihkan Aurora DB atau cluster global, pertimbangkan item berikut:

- Setelah akhir Aurora dari tanggal dukungan standar telah berlalu, jika Anda ingin memulihkan DB atau cluster global dari Amazon S3, Anda hanya dapat melakukannya dengan menggunakan atau. AWS CLI RDS API Gunakan `--engine-lifecycle-support` opsi dalam AWS CLI perintah restore-db-cluster-from-s3 atau `EngineLifecycleSupport` parameter dalam operasi RestoreDBCluster RDS API fromS3.
- Jika Anda ingin mencegah Aurora memulihkan database Anda ke versi Extended RDS Support, tentukan `open-source-rds-extended-support-disabled` di bagian atau. AWS CLI RDS API Dengan demikian, Anda akan menghindari biaya RDS Extended Support terkait.

  Jika Anda menentukan setelan ini, Aurora akan secara otomatis memutakhirkan database Anda yang dipulihkan ke versi utama yang lebih baru dan didukung. Jika pemutakhiran gagal dalam pemeriksaan pra-pemutakhiran, Amazon Aurora akan kembali dengan aman ke versi mesin RDS Extended Support. Database ini akan tetap dalam mode RDS Extended Support, dan Aurora

akan menagih Anda untuk Extended RDS Support hingga Anda meng-upgrade database secara manual.

- RDSExtended Support diatur pada tingkat cluster. Anggota cluster akan selalu memiliki pengaturan yang sama untuk RDS Extended Support `--engine-lifecycle-support` di RDS konsol, di AWS CLI, dan `EngineLifecycleSupport` di RDSAPI.

Untuk informasi selengkapnya, lihat Versi Amazon Aurora.

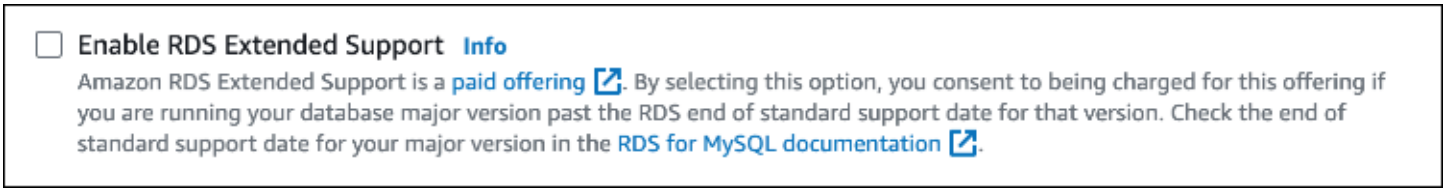
## Kembalikan Aurora DB atau cluster global dengan Extended Support RDS

Anda dapat memulihkan Aurora DB atau cluster global dengan versi RDS Extended Support menggunakan AWS Management Console,, atau AWS CLI file. RDS API

Konsol

Saat memulihkan cluster Aurora DB atau cluster global instans DB atau cluster Extended RDS Support di bagian opsi Engine. Jika Anda tidak memilih pengaturan ini dan versi mesin utama yang Anda pulihkan melewati akhir dukungan standar Aurora, Aurora secara otomatis meningkatkan cluster RDS Aurora DB atau dukungan standar Aurora.

Gambar berikut menunjukkan pengaturan Enable RDS Extended Support:




AWS CLI

Saat Anda menjalankan AWS CLI perintah restore-db-cluster-from-snapshot restore-db-instance-from-db-snapshot dengan menentukan opsi. `open-source-rds-extended-support` `--engine-lifecycle-support`

Jika Anda ingin menghindari biaya yang terkait dengan RDS Extended Support, atur `--engine-lifecycle-support` opsi ke`open-source-rds-extended-support-disabled`. Secara default, opsi ini diatur ke`open-source-rds-extended-support`.

Anda juga dapat menentukan nilai ini menggunakan AWS CLI perintah berikut:

- restore-db-cluster-from-s3

- restore-db-cluster-to-point-in-time

RDS API

Saat Anda menggunakan RDS API operasi R estoreDBCluster FromSnapshot R estoreDBInstance Amazon, pilih RDS Extended Support dengan menyetel `EngineLifecycleSupport` parameter ke`open-source-rds-extended-support`.

Jika Anda ingin menghindari biaya yang terkait dengan RDS Extended Support, atur `EngineLifecycleSupport` parameternya ke`open-source-rds-extended-support-disabled`. Secara default, parameter ini diatur ke`open-source-rds-extended-support`.

Anda juga dapat menentukan nilai ini menggunakan RDS API operasi berikut:

- R estoreDBCluster dari3
- R estoreDBCluster ToPointInTime

Untuk informasi lebih lanjut tentang memulihkan cluster Aurora DB, ikuti petunjuk untuk mesin DB Anda. Mencadangkan dan memulihkan klaster DB Amazon Aurora

# Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database

Deployment blue/green menyalin lingkungan basis data produksi ke lingkungan pementasan yang terpisah dan tersinkron. Dengan menggunakan Amazon Aurora Blue/Green Deployment, Anda dapat membuat perubahan pada database di lingkungan pementasan tanpa memengaruhi lingkungan produksi. Misalnya, Anda dapat memutakhirkan versi mesin DB mayor atau minor atau mengubah parameter database di lingkungan pementasan. Saat Anda siap, Anda dapat mempromosikan lingkungan pementasan menjadi lingkungan basis data produksi baru, dengan waktu henti biasanya di bawah satu menit.

Amazon Aurora menciptakan lingkungan pementasan dengan mengkloning volume penyimpanan Aurora yang mendasarinya di lingkungan produksi. Volume cluster di lingkungan pementasan hanya menyimpan perubahan tambahan yang dibuat pada lingkungan itu.

> **Note**
>
> Saat ini, Penerapan Biru/Hijau didukung untuk RDS Aurora MySQL dan Aurora PostgreSQL. Untuk ketersediaan engine Amazon RDS, lihat Menggunakan Amazon RDS Blue/Green Deployment untuk pembaruan database di Panduan Pengguna Amazon RDS.

Topik


- Ikhtisar Amazon Aurora Penerapan Biru/Hijau
- Membuat penyebaran biru/hijau di Amazon RDS Aurora
- Melihat penyebaran biru/hijau di Amazon RDS Aurora
- Mengalihkan penerapan biru/hijau di Amazon RDS Aurora
- Menghapus penyebaran biru/hijau di Amazon RDS Aurora

# Ikhtisar Amazon Aurora Penerapan Biru/Hijau

Dengan menggunakan Amazon Aurora Blue/Green Deployment, Anda dapat membuat dan menguji perubahan database sebelum menerapkannya di lingkungan produksi. Deployment blue/green menciptakan lingkungan pementasan yang menyalin lingkungan produksi. Dalam deployment blue/green, lingkungan biru adalah lingkungan produksi saat ini. Lingkungan hijau adalah lingkungan pementasan dan tetap sinkron dengan lingkungan produksi saat ini.

Anda dapat membuat perubahan pada klaster DB Aurora di lingkungan hijau tanpa memengaruhi beban kerja produksi. Misalnya, Anda dapat meningkatkan versi mesin DB mayor atau minor atau mengubah parameter basis data di lingkungan pementasannya. Anda dapat menguji perubahan di lingkungan hijau secara menyeluruh. Saat siap, Anda dapat beralih ke lingkungan untuk mentransisikan lingkungan hijau menjadi lingkungan produksi baru. Switchover biasanya memakan waktu kurang dari satu menit tanpa kehilangan data dan tidak perlu mengubah aplikasi.

Karena lingkungan hijau adalah salinan topologi lingkungan dari produksi, klaster DB dan semua instans DB-nya disalin dalam deployment. Lingkungan hijau juga mencakup fitur yang digunakan oleh cluster DB, seperti snapshot cluster DB, Performance Insights, Enhanced Monitoring, dan Aurora Serverless v2.

> **Note**
>
> Penerapan Biru/Hijau didukung untuk Aurora MySQL dan Aurora PostgreSQL. Untuk ketersediaan Amazon RDS, lihat Ikhtisar Penyebaran Biru/Hijau Amazon RDS di Panduan Pengguna Amazon RDS.

Topik


- Wilayah dan ketersediaan versi
- Manfaat menggunakan Deployment Blue/Green Amazon RDS
- Alur kerja deployment blue/green
- Mengotorisasi akses ke penyebaran biru/hijau
- Keterbatasan dan pertimbangan untuk penerapan Aurora blue/green
- Praktik terbaik untuk Aurora penerapan biru/hijau

## Wilayah dan ketersediaan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin basis data, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya, lihat the section called “Deployment Blue/Green”.

## Manfaat menggunakan Deployment Blue/Green Amazon RDS

Dengan menggunakan Blue/Green Deployments, you can stay current on security patches, improve database performance, and adopt newer database features with short, predictable downtime. Blue/green penerapan Amazon RDS mengurangi risiko dan waktu henti untuk pembaruan basis data, seperti peningkatan versi mesin utama atau minor.

Deployment blue/green memberikan manfaat berikut:

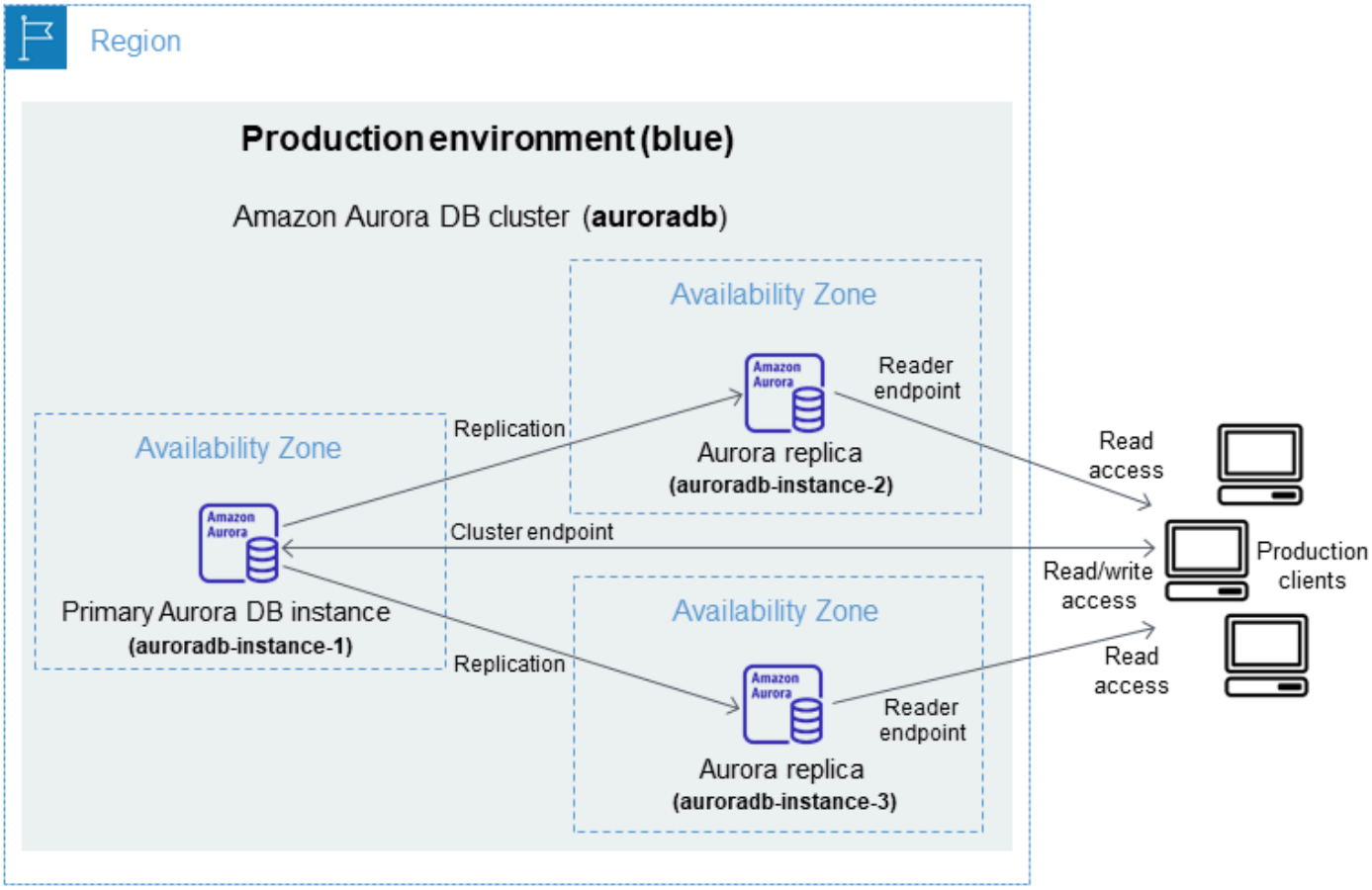
- Memudahkan pembuatan lingkungan pementasan siap produksi.
- Mereplikasi otomatis perubahan basis data dari lingkungan produksi ke lingkungan pementasan.
- Menguji perubahan basis data di lingkungan pementasan yang aman tanpa memengaruhi lingkungan produksi.
- Anda dapat mengikuti perkembangan terbaru dengan patch basis data dan pembaruan sistem.
- Menerapkan dan menguji fitur basis data yang lebih baru.
- Melakukan switchover pada lingkungan pementasan untuk menjadi lingkungan produksi baru tanpa perubahan pada aplikasi.
- Melakukan switchover dengan aman melalui penggunaan pagar pembatas switchover default.
- Tidak ada kehilangan data selama switchover.
- Melakukan switchover dengan cepat, biasanya kurang dari satu menit tergantung beban kerja Anda.

## Alur kerja deployment blue/green

Selesaikan langkah-langkah utama berikut saat Anda menggunakan deployment blue/green untuk pembaruan klaster DB Aurora.

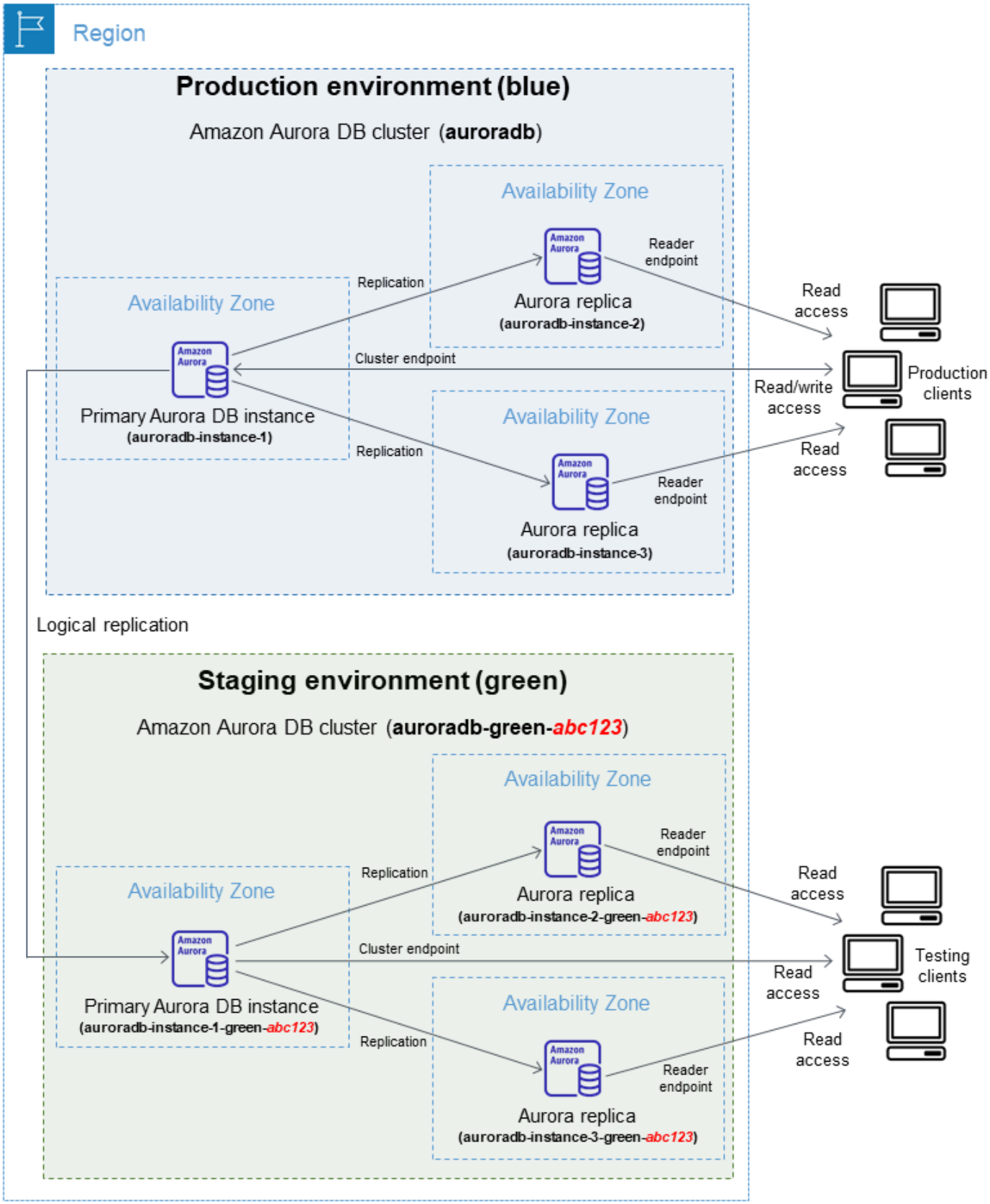
1. Identifikasi klaster DB produksi yang memerlukan pembaruan.

   Gambar berikut menunjukkan contoh klaster DB produksi.




2. Buat deployment blue/green. Untuk petunjuknya, lihat Membuat penyebaran biru/hijau di Amazon RDS Aurora.

   Gambar berikut menunjukkan contoh blue/green deployment of the production environment from step 1. While creating the blue/green penyebaran, RDS menyalin topologi lengkap dan konfigurasi cluster Aurora DB untuk menciptakan lingkungan hijau. Nama-nama klaster DB dan instans DB yang disalin ditambahkan dengan `-green-`*`random-characters`*. Lingkungan pementasan dalam gambar berisi cluster DB (auroradb-green-). *`abc123`* Ini juga berisi tiga instance DB di cluster DB (auroradb-instance1-green-, auroradb-instance2-green-, dan auroradb-instance3-green-)*`abc123`*. *`abc123`* *`abc123`*


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Amazon Aurora
Reader endpoint
Replication
Availability Zone
Aurora replica
(auroradb-instance-2)
Read access
Cluster endpoint
Production clients
Read/write access
Primary Aurora DB instance
(auroradb-instance-1)
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Amazon Aurora
Reader endpoint
Replication
Availability Zone
Aurora replica
(auroradb-instance-2-green-abc123)
Read access
Cluster endpoint
Testing clients
Read access
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3-green-abc123)

Saat membuat deployment blue/green, Anda dapat menentukan versi mesin DB yang lebih tinggi dan grup parameter klaster DB yang berbeda untuk klaster DB di lingkungan hijau. Anda juga dapat menentukan grup parameter DB yang berbeda untuk instans DB di klaster DB.

RDS juga mengonfigurasi replikasi dari instans DB primer di lingkungan biru ke instans DB primer di lingkungan hijau.

> **Important**
>
> Untuk Aurora MySQL versi 3, setelah Anda membuat penyebaran biru/hijau, cluster DB di lingkungan hijau tidak mengizinkan operasi tulis secara default. Namun, ini tidak berlaku untuk pengguna yang memiliki `CONNECTION_ADMIN` hak istimewa, termasuk pengguna master Aurora. Pengguna dengan hak istimewa ini dapat mengesampingkan perilaku tersebut. `read_only` Untuk informasi selengkapnya, lihat Model hak akses berbasis peran.

3. Buat perubahan pada lingkungan pementasan.

   Misalnya, Anda dapat mengubah kelas instans DB yang digunakan oleh satu atau lebih instance DB di lingkungan hijau.

   Untuk informasi tentang memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

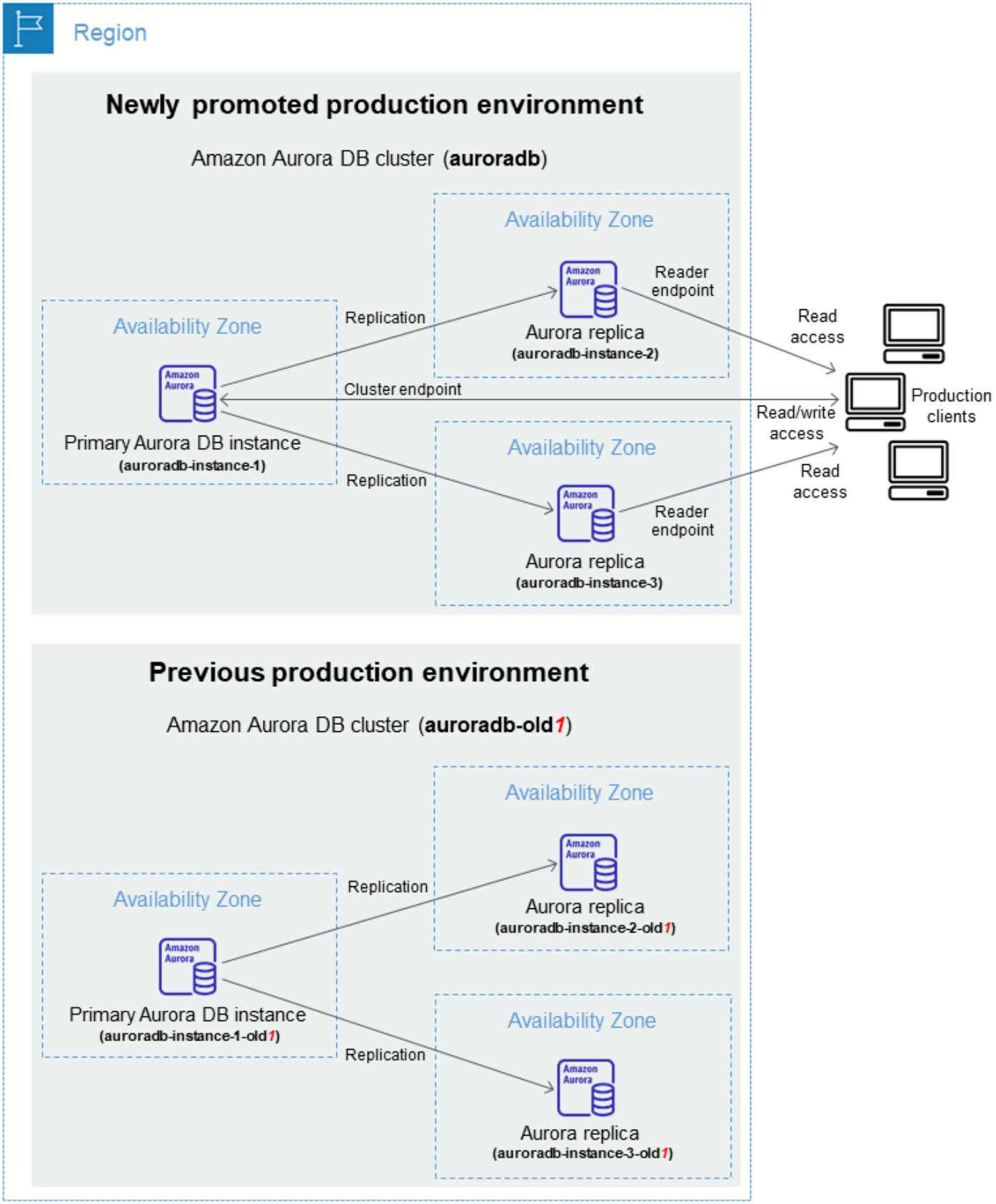
4. Uji lingkungan pementasan Anda.

   Selama pengujian, sebaiknya pertahankan basis data Anda di lingkungan hijau agar hanya dapat dibaca saja. Aktifkan operasi tulis di lingkungan hijau dengan hati-hati karena dapat mengakibatkan konflik replikasi. Hal ini juga dapat menghasilkan data yang tidak diinginkan dalam basis data produksi setelah switchover. Untuk mengaktifkan operasi tulis untuk Aurora MySQL, atur `read_only` parameternya ke`0`, lalu reboot instance DB. Untuk Aurora PostgreSQL, atur parameter ke level sesi. `default_transaction_read_only off`

5. Saat siap, beralih ke transisi lingkungan pementasan menjadi lingkungan produksi baru. Untuk petunjuknya, lihat Mengalihkan penerapan biru/hijau di Amazon RDS Aurora.

   Switchover menyebabkan waktu henti. Waktu henti biasanya kurang dari satu menit, tetapi bisa lebih lama tergantung beban kerja Anda.

   Gambar berikut menunjukkan klaster DB setelah switchover.


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Availability Zone
Amazon Aurora
Reader endpoint
Replication
Aurora replica
(auroradb-instance-2)
Read access
Cluster endpoint
Production clients
Read/write access
Primary Aurora DB instance
(auroradb-instance-1)
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Availability Zone
Replication
Aurora replica
(auroradb-instance-2-old1)
Primary Aurora DB instance
(auroradb-instance-1-old1)
Availability Zone
Replication
Aurora replica
(auroradb-instance-3-old1)

Setelah switchover, klaster DB Aurora di lingkungan hijau menjadi klaster DB produksi baru. Nama dan titik akhir di lingkungan produksi saat ini ditetapkan ke lingkungan produksi yang baru dialihkan, tidak memerlukan perubahan pada aplikasi Anda. Akibatnya, lalu lintas produksi Anda sekarang mengalir ke lingkungan produksi baru. Klaster DB dan instans DB di lingkungan biru diganti namanya dengan menambahkan `-old`*n* ke nama saat ini, dengan *n* adalah angka. Misalnya, anggap nama instans DB di lingkungan biru adalah `auroradb-instance-1`. Setelah switchover, nama instans DB bisa jadi `auroradb-instance-1-old1`.

Dalam contoh pada gambar, perubahan berikut terjadi selama switchover:

- Klaster DB lingkungan hijau `auroradb-green-abc123` menjadi klaster DB produksi bernama `auroradb`.
- Instans DB lingkungan hijau bernama `auroradb-instance1-green-abc123` menjadi instans DB produksi `auroradb-instance1`.
- Instans DB lingkungan hijau bernama `auroradb-instance2-green-abc123` menjadi instans DB produksi `auroradb-instance2`.
- Instans DB lingkungan hijau bernama `auroradb-instance3-green-abc123` menjadi instans DB produksi `auroradb-instance3`.
- Klaster DB lingkungan biru bernama `auroradb` menjadi `auroradb-old1`.
- Instans DB lingkungan biru bernama `auroradb-instance1` menjadi `auroradb-instance1-old1`.
- Instans DB lingkungan biru bernama `auroradb-instance2` menjadi `auroradb-instance2-old1`.
- Instans DB lingkungan biru bernama `auroradb-instance3` menjadi `auroradb-instance3-old1`.

6. Jika Anda tidak lagi memerlukan deployment blue/green, Anda dapat menghapusnya. Untuk petunjuknya, lihat Menghapus penyebaran biru/hijau di Amazon RDS Aurora.

   Setelah switchover, lingkungan produksi sebelumnya tidak dihapus sehingga Anda dapat menggunakannya untuk pengujian regresi, jika perlu.

## Mengotorisasi akses ke penyebaran biru/hijau

Pengguna harus memiliki izin yang diperlukan untuk melakukan operasi yang terkait dengan deployment blue/green. Anda dapat membuat kebijakan IAM yang memberi izin kepada pengguna

dan peran untuk menjalankan operasi API tertentu pada sumber daya yang diperlukan. Anda kemudian dapat melampirkan kebijakan tersebut ke set izin IAM atau peran yang memerlukan izin tersebut. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.

Pengguna yang membuat deployment blue/green harus memiliki izin untuk menjalankan operasi RDS berikut:

- `rds:CreateBlueGreenDeployment`
- `rds:AddTagsToResource`
- `rds:CreateDBCluster`
- `rds:CreateDBInstance`
- `rds:CreateDBClusterEndpoint`

Pengguna yang melakukan switchover pada deployment blue/green harus memiliki izin untuk menjalankan operasi RDS berikut:

- `rds:SwitchoverBlueGreenDeployment`
- `rds:ModifyDBCluster`
- `rds:PromoteReadReplicaDBCluster`

Pengguna yang menghapus deployment blue/green harus memiliki izin untuk menjalankan operasi RDS berikut:

- `rds:DeleteBlueGreenDeployment`
- `rds:DeleteDBCluster`
- `rds:DeleteDBInstance`
- `rds:DeleteDBClusterEndpoint`

Aurora menyediakan dan memodifikasi sumber daya di lingkungan pementasan atas nama Anda. Sumber daya ini mencakup instance DB yang menggunakan konvensi penamaan yang ditentukan secara internal. Oleh karena itu, kebijakan IAM terlampir tidak dapat berisi pola nama sumber daya sebagian sepertimy`-db-prefix-*`. Hanya wildcard (*) yang didukung. Secara umum, sebaiknya gunakan tag sumber daya dan atribut lain yang didukung untuk mengontrol akses ke sumber daya ini, bukan wildcard. Untuk informasi selengkapnya, lihat Kunci tindakan, sumber daya, dan kondisi untuk Amazon RDS.

# Keterbatasan dan pertimbangan untuk penerapan Aurora blue/green

Penerapan biru/hijau di Amazon RDS memerlukan pertimbangan yang cermat terhadap faktor-faktor seperti slot replikasi, manajemen sumber daya, ukuran instans, dan potensi dampak pada kinerja database. Bagian berikut memberikan panduan untuk membantu Anda mengoptimalkan strategi penerapan untuk memastikan waktu henti minimal, transisi yang mulus, dan pengelolaan lingkungan database Anda yang efektif.

Topik


- Batasan untuk blue/green penerapan
- Pertimbangan untuk penerapan blue/green


## Batasan untuk blue/green penerapan

Batasan berikut berlaku untuk blue/green penerapan.

Topik


- Batasan umum untuk deployment blue/green
- Aurora MySQL untuk batasan MySQL untuk penerapan blue/green
- Aurora PostgreSQL RDS untuk batasan PostgreSQL


### Batasan umum untuk deployment blue/green

Batasan umum berikut berlaku untuk blue/green penerapan:

- Anda tidak dapat menghentikan dan memulai cluster yang merupakan bagian dari blue/green penerapan.
- Penerapan biru/hijau tidak mendukung pengelolaan kata sandi pengguna utama dengan. AWS Secrets Manager
- Jika Anda mencoba memaksa backtrack pada cluster DB biru, blue/green penerapan akan rusak dan peralihan diblokir.
- Selama switchover, lingkungan biru dan hijau tidak boleh memiliki integrasi nol-ETL dengan Amazon Redshift. Anda harus menghapus integrasi tersebut terlebih dahulu dan switchover, lalu membuat ulang integrasi.

- Penjadwal Acara (`event_scheduler`parameter) harus dinonaktifkan di lingkungan hijau saat Anda membuat blue/green penerapan. Ini mencegah peristiwa dihasilkan di lingkungan hijau dan menyebabkan inkonsistensi.
- Kebijakan Auto Scaling yang dikonfigurasi pada cluster DB biru tidak disalin ke lingkungan hijau. Anda harus mengkonfigurasi ulang mereka setelah peralihan, terlepas dari apakah mereka awalnya diatur di lingkungan biru atau hijau.
- Anda tidak dapat mengubah klaster DB yang tidak terenkripsi menjadi klaster DB yang terenkripsi.
- Anda tidak dapat mengubah cluster DB biru ke versi engine yang lebih tinggi daripada cluster DB hijau yang sesuai.
- Sumber daya di lingkungan biru dan lingkungan hijau harus berada dalam Akun AWS yang sama.
- Deployment blue/green tidak didukung untuk fitur berikut:
  - Proksi Amazon RDS
  - Replika baca Lintas Wilayah
  - Klaster DB Aurora Serverless v1
  - Klaster DB yang merupakan bagian dari basis data global Aurora
  - AWS CloudFormation

Aurora MySQL untuk batasan MySQL untuk penerapan blue/green

blue/green

- Cluster DB sumber tidak dapat berisi database apa pun bernama`tmp`. Basis data dengan nama ini tidak akan disalin ke lingkungan hijau.
- Cluster DB biru tidak bisa menjadi replika binlog eksternal.
- Jika cluster DB sumber yang memiliki backtrack diaktifkan, cluster DB hijau dibuat tanpa dukungan backtracking. Ini karena backtracking tidak berfungsi dengan replikasi log biner (binlog), yang diperlukan untuk penerapan. blue/green Untuk informasi selengkapnya, lihat the section called “Melakukan backtracking klaster DB”.
- Penerapan biru/hijau tidak mendukung Driver AWS JDBC untuk MySQL. Untuk informasi selengkapnya, lihat Batasan yang Diketahui pada GitHub.

Aurora PostgreSQL RDS untuk batasan PostgreSQL

- Tabel yang tidak tercatat tidak direplikasi ke lingkungan hijau kecuali `rds.logically_replicate_unlogged_tables` parameter disetel ke 1 pada cluster DB biru. Jangan mengubah nilai parameter ini setelah Anda membuat blue/green penerapan untuk menghindari kemungkinan kesalahan replikasi pada tabel yang tidak tercatat.
- Cluster DB biru tidak bisa menjadi sumber logis (penerbit) atau replika (pelanggan).
- Jika klaster DB biru dikonfigurasi sebagai server asing dari ekstensi pembungkus data asing (FDW), Anda harus menggunakan nama titik akhir klaster, alih-alih alamat IP. Hal ini memungkinkan konfigurasi untuk tetap berfungsi setelah switchover.
- Dalam blue/green penyebaran, setiap database memerlukan slot replikasi logis. Seiring bertambahnya jumlah database, overhead sumber daya meningkat dan berpotensi menyebabkan kelambatan replikasi, terutama jika . Dampaknya tergantung pada faktor-faktor seperti beban kerja database dan jumlah koneksi.
- Penerapan biru/hijau didukung untuk Babelfish untuk Aurora PostgreSQL hanya untuk versi 15.7 dan versi 15 yang lebih tinggi, dan 16.3 dan versi 16 yang lebih tinggi.
- Jika ingin menangkap rencana eksekusi di Aurora Replicas, Anda harus memberikan titik akhir klaster DB biru saat memanggil fungsi `apg_plan_mgmt.create_replica_plan_capture`. Ini memastikan bahwa pengambilan rencana terus berfungsi setelah switchover. Untuk informasi selengkapnya, lihat the section called “Mengambil rencana eksekusi Aurora PostgreSQL di Replika”.
- Proses penerapan replikasi logis di lingkungan hijau adalah single-threaded. Jika lingkungan biru menghasilkan volume lalu lintas tulis yang tinggi, lingkungan hijau mungkin tidak dapat mengikutinya. Hal ini dapat menyebabkan kelambatan replikasi atau kegagalan, terutama untuk beban kerja yang menghasilkan throughput penulisan tinggi yang berkelanjutan. Pastikan untuk menguji beban kerja Anda secara menyeluruh. Untuk skenario yang memerlukan peningkatan versi mayor dan menangani beban kerja tulis volume tinggi, pertimbangkan pendekatan alternatif seperti menggunakan AWS Database Migration Service (AWS DMS) atau replikasi logis yang dikelola sendiri.
- Membuat partisi baru melibatkan operasi bahasa definisi data (DDL) seperti`CREATE TABLE`, yang tidak direplikasi dari lingkungan biru ke lingkungan hijau. Namun, tabel partisi yang ada dan datanya akan direplikasi ke lingkungan hijau.
- Batasan berikut berlaku untuk ekstensi PostgreSQL:
  - `pg_partman`Ekstensi harus dinonaktifkan di lingkungan biru saat Anda membuat blue/green penerapan. Ekstensi tersebut menjalankan operasi DDL seperti `CREATE TABLE`, yang memecah replikasi logis dari lingkungan biru ke lingkungan hijau.

- `pg_cron`Ekstensi harus tetap dinonaktifkan pada semua database hijau setelah blue/green penerapan dibuat. Ekstensi tersebut memiliki pekerja latar belakang yang berjalan sebagai superuser dan melewati pengaturan hanya baca di lingkungan hijau, yang dapat menyebabkan konflik replikasi.
- Ekstensi `apg_plan_mgmt` harus memiliki parameter `apg_plan_mgmt.capture_plan_baselines` yang diatur ke `off` di semua basis data hijau untuk menghindari konflik kunci primer jika rencana yang identik ditangkap di lingkungan biru. Untuk informasi selengkapnya, lihat the section called “Gambaran umum manajemen rencana kueri Aurora PostgreSQL”.
- `pgactive`Ekstensi `pglogical` dan harus dinonaktifkan di lingkungan biru saat Anda membuat blue/green penerapan. Setelah Anda beralih ke lingkungan hijau menjadi lingkungan produksi baru, Anda dapat mengaktifkan ekstensi lagi. Selain itu, basis data biru tidak bisa menjadi pelanggan logis dari instans eksternal.
- Jika Anda menggunakan `pgAudit` ekstensi, ekstensi harus tetap berada di pustaka bersama (`shared_preload_libraries`) pada grup parameter DB khusus untuk instance DB biru dan hijau. Untuk informasi selengkapnya, lihat the section called “Menyiapkan pgAudit ekstensi”.

Keterbatasan spesifik replikasi logis untuk penerapan blue/green

Tabel berikut menjelaskan batasan replikasi logis yang berlaku untuk deployment blue/green Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Restrictions di dokumentasi replikasi logis PostgreSQL.

| Batasan | Penjelasan |
| --- | --- |
| Pernyataan bahasa definisi data (DDL), seperti `CREATE TABLE` dan `CREATE SCHEMA`, tidak direplikasi dari lingkungan biru | Jika Aurora mendeteksi perubahan DDL di lingkungan biru, basis data hijau Anda memasukkan status Replikasi terdegradasi. Anda harus menghapus blue/green penyebaran dan semua database hijau, lalu membuatnya kembali. |

| Batasan | Penjelasan |
| --- | --- |
| ke lingkungan hijau. | |
| Pernyataan bahasa kontrol data (DCL), seperti `GRANT` dan`REVOKE`, tidak direplikasi dari lingkungan biru ke lingkungan hijau. | Jika Aurora Amazon mendeteksi upaya untuk mengeksekusi pernyataan DCL di lingkungan biru, Anda akan melihat pesan peringatan. Tidak ada konfigurasi atau API yang tersedia untuk mengubah perilaku ini, karena ini merupakan batasan dari proses blue/green penerapan. |
| Operasi `NEXTVAL` pada objek urutan tidak disinkronkan antara lingkungan biru dan lingkungan hijau. | Selama switchover, Aurora menambah nilai urutan di lingkungan hijau agar sesuai dengan yang ada di lingkungan biru. Jika Anda memiliki ribuan urutan, hal ini dapat menunda switchover. |
| Objek besar di lingkungan biru tidak direplikasi ke lingkungan hijau. Ini termasuk objek besar yang ada dan objek besar yang baru dibuat atau dimodifikasi selama proses blue/green penyebaran. | Jika Aurora mendeteksi pembuatan atau perubahan objek besar di lingkungan biru yang disimpan dalam tabel sistem `pg_largeobject` , basis data hijau Anda memasukkan status Replikasi terdegradasi. Anda harus menghapus blue/green penyebaran dan semua database hijau, lalu membuatnya kembali. |

| Batasan | Penjelasan |
| --- | --- |
| Menyegarkan tampilan terwujud merusak replikasi . | Pemandangan terwujud yang menyegarkan di lingkungan biru memecah replikasi ke lingkungan hijau. Menahan diri dari menyegarkan pandangan terwujud di lingkungan biru. Setelah peralihan, Anda dapat menyegarkannya secara manual menggunakan perintah REFRESH MATERIALIZED VIEW, atau menjadwalkan penyegaran. |
| Operasi UPDATE dan DELETE tidak diizinkan pada tabel yang tidak memiliki kunci primer. | Sebelum Anda membuat blue/green penerapan, pastikan bahwa semua tabel memiliki kunci utama atau penggunaan`REPLICA IDENTITY FULL`. Namun, gunakan hanya `REPLICA IDENTITY FULL` jika tidak ada kunci primer atau unik, karena memengaruhi kinerja replikasi. Untuk informasi selengkapnya, lihat Dokumentasi PostgreSQL. |

## Pertimbangan untuk penerapan blue/green

Amazon RDS melacak sumber daya dalam blue/green penerapan dengan `DbiResourceId` dan `DbClusterResourceId` dari setiap sumber daya. ID sumber daya ini adalah pengenal Wilayah AWS-unik dan tidak dapat diubah untuk sumber daya.

ID sumber daya terpisah dari ID  cluster DB. Masing-masing tercantum dalam konfigurasi database di konsol RDS.

Nama (ID klaster) sumber daya berubah saat Anda mengalihkan blue/green penerapan, tetapi setiap sumber daya menyimpan ID sumber daya yang sama. Misalnya, pengidentifikasi klaster DB mungkin adalah `mycluster` di lingkungan biru. Setelah switchover, klaster DB yang sama mungkin diganti namanya menjadi `mycluster-old1`. Namun, ID sumber daya klaster DB tidak berubah selama switchover. Jadi, ketika Anda mengganti sumber daya hijau menjadi sumber daya produksi baru, sumber daya mereka IDs tidak cocok dengan sumber daya biru IDs yang sebelumnya dalam produksi.

Setelah Anda mengalihkan blue/green penerapan, pertimbangkan untuk memperbarui sumber daya IDs ke sumber daya produksi yang baru ditransisi untuk fitur dan layanan terintegrasi yang Anda gunakan dengan sumber daya produksi. Secara khusus, pertimbangkan pembaruan berikut:

- Jika Anda melakukan pemfilteran menggunakan RDS API dan sumber daya IDs, sesuaikan sumber daya yang IDs digunakan dalam pemfilteran setelah peralihan.

- Jika Anda menggunakan CloudTrail untuk mengaudit sumber daya, sesuaikan konsumen CloudTrail untuk melacak sumber daya baru IDs setelah peralihan. Untuk informasi selengkapnya, lihat Memantau panggilan Amazon Aurora API AWS CloudTrail.
- Jika Anda menggunakan Stream Aktivitas Basis Data untuk sumber daya di lingkungan biru, sesuaikan aplikasi Anda untuk memantau peristiwa basis data aliran baru setelah switchover. Untuk informasi selengkapnya, lihat Daerah yang Didukung dan mesin Aurora DB untuk aliran aktivitas database.
- Jika Anda menggunakan API Performance Insights, sesuaikan sumber daya IDs dalam panggilan ke API setelah peralihan. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

  Anda dapat memantau basis data dengan nama yang sama setelah switchover, tetapi basis data tersebut tidak berisi data sebelum switchover.
- Jika Anda menggunakan sumber daya IDs dalam kebijakan IAM, pastikan Anda menambahkan sumber daya sumber daya IDs yang baru dialihkan bila diperlukan. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.
- Jika Anda memiliki peran IAM yang terkait dengan Anda, pastikan untuk mengasosiasikannya kembali setelah peralihan. Peran terlampir tidak secara otomatis disalin ke lingkungan hijau.
- Jika Anda mengautentikasi klaster DB menggunakan autentikasi basis data IAM, pastikan kebijakan IAM yang digunakan untuk akses basis data memiliki basis data biru dan hijau yang tercantum di elemen `Resource` kebijakan. Ini diperlukan agar dapat terhubung ke basis data hijau setelah switchover. Untuk informasi selengkapnya, lihat the section called “Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM”.
- Jika Anda ingin memulihkan snapshot cluster DB manual untuk cluster DB yang merupakan bagian dari blue/green penerapan, pastikan Anda memulihkan snapshot cluster DB yang benar dengan memeriksa waktu ketika snapshot diambil. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.
- Setelah Anda beralih, AWS Database Migration Service (AWS DMS) tugas replikasi tidak dapat dilanjutkan karena pos pemeriksaan dari lingkungan biru tidak valid di lingkungan hijau. Anda harus membuat ulang tugas DMS dengan pos pemeriksaan baru untuk melanjutkan replikasi.
- Amazon Aurora menciptakan lingkungan hijau dengan mengkloning volume penyimpanan Aurora yang mendasarinya di lingkungan biru. Volume klaster hijau hanya menyimpan perubahan tambahan yang dilakukan pada lingkungan hijau. Jika Anda menghapus klaster DB di lingkungan biru, ukuran volume penyimpanan Aurora yang mendasarinya di lingkungan hijau meningkat ke

ukuran penuh. Untuk informasi selengkapnya, lihat the section called “Mengkloning volume untuk klaster DB Aurora”.

- Saat Anda menambahkan instans DB ke klaster DB di lingkungan hijau deployment blue/green, instans DB baru tidak akan menggantikan instans DB di lingkungan biru saat Anda switchover. Namun, instans DB baru dipertahankan di klaster DB dan menjadi instans DB di lingkungan produksi baru.
- Saat Anda menghapus instans DB di cluster DB di lingkungan hijau blue/green deployment, you can't create a new DB instance to replace it in the blue/green penerapan.

  Jika Anda membuat instans DB baru dengan nama dan ARN yang sama dengan instans DB yang dihapus, instans tersebut memiliki `DbiResourceId` yang berbeda, jadi instans tersebut bukan bagian dari lingkungan hijau.

  Perilaku berikut terjadi jika Anda menghapus instans DB di klaster DB di lingkungan hijau:

  - Jika ada instans DB di lingkungan biru dengan nama yang sama, instans tersebut tidak akan switchover ke instans DB di lingkungan hijau. Instans DB ini tidak akan diganti namanya dengan menambahkan `-old`*n* ke nama instans DB.
  - Aplikasi apa pun yang menunjuk ke instans DB di lingkungan biru terus menggunakan instans DB yang sama setelah switchover.

## Praktik terbaik untuk Aurora penerapan biru/hijau

Berikut ini adalah praktik terbaik untuk penerapan biru/hijau.

Topik

- Praktik terbaik umum untuk penerapan biru/hijau
- untuk penerapan biru/hijau
- Praktik terbaik Aurora PostgreSQL untuk penerapan biru/hijau

### Praktik terbaik umum untuk penerapan biru/hijau

Pertimbangkan praktik terbaik umum berikut saat Anda membuat penerapan biru/hijau.

- Uji klaster DB Aurora secara menyeluruh di lingkungan hijau sebelum switchover.
- Simpan basis data Anda di lingkungan hijau dengan kondisi hanya baca. Sebaiknya Anda mengaktifkan operasi tulis di lingkungan hijau dengan hati-hati karena dapat mengakibatkan konflik

replikasi. Hal ini juga dapat menghasilkan data yang tidak diinginkan dalam basis data produksi setelah switchover.

- Jika Anda menggunakan penerapan biru/hijau untuk menerapkan perubahan skema, buat hanya perubahan yang kompatibel dengan replikasi.

  Misalnya, Anda dapat menambahkan kolom baru di akhir tabel tanpa mengganggu replikasi dari penerapan biru ke penerapan hijau. Namun, perubahan skema, seperti penggantian nama kolom atau nama tabel, memecah replikasi ke deployment hijau.

  Untuk informasi selengkapnya tentang perubahan yang kompatibel dengan replikasi, lihat Replication with Differing Table Definitions on Source and Replica di dokumentasi MySQL dan Restrictions dalam dokumentasi replikasi logis PostgreSQL.
- Gunakan titik akhir klaster, titik akhir pembaca, atau titik akhir kustom untuk semua koneksi di kedua lingkungan. Jangan gunakan titik akhir instans atau titik akhir kustom dengan daftar statis atau pengecualian.
- Saat Anda mengalihkan deployment blue/green, ikuti praktik terbaik switchover. Untuk informasi selengkapnya, lihat the section called “Praktik terbaik switchover”.

### untuk penerapan biru/hijau

Pertimbangkan praktik terbaik berikut saat Anda membuat penerapan biru/hijau dari .

- Jika lingkungan hijau mengalami kelambatan replika, pertimbangkan hal berikut:
  - Nonaktifkan retensi log biner jika tidak diperlukan, atau nonaktifkan sementara hingga replikasi menyusul. Untuk melakukannya, atur kembali parameter cluster `binlog_format` DB `0` dan reboot instance DB penulis hijau.
  - Setel sementara `innodb_flush_log_at_trx_commit` parameter ke dalam kelompok parameter DB hijau. Setelah replikasi menyusul, kembalikan ke nilai default sebelum peralihan. 1 Jika terjadi shutdown atau crash yang tidak terduga dengan nilai parameter sementara, bangun kembali lingkungan hijau untuk menghindari kerusakan data yang tidak terdeteksi. Untuk informasi selengkapnya, lihat the section called “Mengonfigurasi seberapa sering buffer log di-flush”.

## Praktik terbaik Aurora PostgreSQL untuk penerapan biru/hijau

Pertimbangkan praktik terbaik berikut saat Anda membuat penerapan biru/hijau dari cluster DB PostgreSQL Aurora.

- Pantau cache penulisan replikasi logis Aurora PostgreSQL dan buat penyesuaian pada buffer cache jika perlu. Untuk informasi selengkapnya, lihat the section called “Memantau cache penulisan replikasi logis”.
- Tingkatkan nilai parameter `logical_decoding_work_mem` DB di lingkungan biru. Tindakan ini memungkinkan lebih sedikit decoding pada disk, alih-alih menggunakan memori. Untuk informasi selengkapnya, lihat the section called “Menyesuaikan memori kerja untuk pendekodean logis”.
  - Anda dapat memantau overflow transaksi yang ditulis ke disk menggunakan `ReplicationSlotDiskUsage` CloudWatch metrik. Metrik ini menawarkan wawasan tentang penggunaan disk slot replikasi, membantu mengidentifikasi kapan data transaksi melebihi kapasitas memori dan disimpan di disk. Anda dapat memantau memori yang dapat dibebaskan dengan `FreeableMemory` CloudWatch metrik. Untuk informasi selengkapnya, lihat the section called “Metrik tingkat instans untuk Amazon Aurora”.
  - Di Aurora PostgreSQL versi 14 dan lebih tinggi, Anda dapat memantau ukuran file luapan logis menggunakan tampilan sistem. `pg_stat_replication_slots`
- Perbarui semua ekstensi PostgreSQL Anda ke versi terbaru sebelum membuat deployment blue/green. Untuk informasi selengkapnya, lihat the section called “Meningkatkan ekstensi PostgreSQL”.
- Jika Anda menggunakan `aws_s3` ekstensi, berikan akses cluster DB hijau ke Amazon S3 melalui peran IAM setelah lingkungan hijau dibuat. Hal ini memungkinkan perintah impor dan ekspor untuk terus berfungsi setelah switchover. Untuk petunjuknya, lihat the section called “Menyiapkan akses ke bucket Amazon S3”.
- Jika Anda menentukan versi engine yang lebih tinggi untuk lingkungan hijau, jalankan `ANALYZE` operasi di semua database untuk menyegarkan `pg_statistic` tabel. Statistik pengoptimal tidak ditransfer selama peningkatan versi utama, jadi Anda harus membuat ulang semua statistik untuk menghindari masalah kinerja. Untuk praktik terbaik tambahan selama peningkatan versi utama, lihatthe section called “Melakukan upgrade versi utama”.
- Hindari mengonfigurasi pemicu sebagai `ENABLE REPLICA` atau `ENABLE ALWAYS` jika pemicu digunakan pada sumber untuk memanipulasi data. Jika tidak, sistem replikasi menyebarkan perubahan dan mengeksekusi pemicu, yang mengarah ke duplikasi.
- Transaksi yang berjalan lama dapat menyebabkan kelambatan replika yang signifikan. Untuk mengurangi lag replika, pertimbangkan untuk melakukan hal berikut:

- Kurangi transaksi jangka panjang dan subtransaksi yang dapat ditunda hingga setelah lingkungan hijau mengejar lingkungan biru.
- Kurangi operasi massal di lingkungan biru sampai setelah lingkungan hijau mengejar lingkungan biru.
- Memulai operasi pembekuan vakum manual pada tabel sibuk sebelum membuat penerapan biru/hijau. Untuk petunjuk, lihat Melakukan pembekuan vakum manual.
- Dalam PostgreSQL versi 12 dan lebih tinggi, nonaktifkan parameter `index_cleanup` pada tabel besar atau sibuk untuk meningkatkan efisiensi pemeliharaan rutin pada database biru. Untuk informasi selengkapnya, lihat Memvakum tabel secepat mungkin.

  > **Note**
  >
  > Melewatkan pembersihan indeks secara teratur selama menyedot debu dapat menyebabkan indeks kembung, yang dapat menurunkan kinerja pemindaian. Sebagai praktik terbaik, gunakan pendekatan ini hanya saat menggunakan penerapan biru/hijau. Setelah penerapan selesai, kami sarankan untuk melanjutkan pemeliharaan dan pembersihan indeks reguler.

- Replica lag dapat terjadi jika instance DB biru dan hijau berukuran terlalu kecil untuk beban kerja. Pastikan instans DB Anda tidak mencapai batas sumber dayanya untuk jenis instans. Untuk informasi selengkapnya, lihat the section called “Menganalisis penggunaan sumber daya dengan CloudWatch metrik”.

- Replikasi yang lambat dapat menyebabkan pengirim dan penerima sering memulai ulang, yang menunda sinkronisasi. Untuk memastikan bahwa mereka tetap aktif, nonaktifkan batas waktu dengan menyetel `wal_sender_timeout` parameter ke `0` dalam lingkungan biru, dan `wal_receiver_timeout` parameter ke `0` dalam lingkungan hijau.
- Tinjau kinerja pernyataan UPDATE dan DELETE Anda dan evaluasi apakah membuat indeks pada kolom yang digunakan dalam klausa WHERE dapat mengoptimalkan kueri ini. Ini dapat meningkatkan kinerja saat operasi diputar ulang di lingkungan hijau. Untuk informasi selengkapnya, lihat the section called “Memeriksa filter predikat untuk kueri yang menghasilkan peristiwa tunggu”.
- Jika Anda menggunakan pemicu, pastikan mereka tidak mengganggu pembuatan, pembaruan, dan penghapusan, `pg_catalog.pg_publicationpg_catalog.pg_subscription`, dan `pg_catalog.pg_replication_slots` objek yang namanya dimulai dengan 'rds'.

- Jika Babelfish diaktifkan pada cluster DB sumber, parameter berikut harus memiliki pengaturan yang sama di grup parameter cluster DB target untuk lingkungan hijau seperti pada grup parameter cluster DB sumber:
  - rds.babelfish_status
  - babelfishpg_tds.tds_default_numeric_precision
  - babelfishpg_tds.tds_default_numeric_scale
  - babelfishpg_tsql.default_locale
  - babelfishpg_tsql.migration_mode
  - babelfishpg_tsql.server_collation_name

  Untuk informasi selengkapnya tentang parameter ini, lihat the section called “Pengaturan grup parameter klaster DB untuk Babelfish ”.

# Membuat penyebaran biru/hijau di Amazon RDS Aurora

RDS menyalin topologi dan fitur lingkungan biru ke area pementasan. Jika instans DB biru telah membaca replika, mereka disalin sebagai replika dari instance hijau. Penyimpanan yang dialokasikan dari semua replika hijau cocok dengan instance primer hijau, sementara parameter penyimpanan lainnya diwarisi dari replika biru.

Saat membuat deployment blue/green, Anda menentukan klaster DB yang akan disalin dalam deployment. Klaster DB yang Anda pilih adalah klaster DB produksi, dan menjadi klaster DB di lingkungan biru. RDS menyalin topologi lingkungan biru ke area pementasan, beserta fitur yang dikonfigurasinya. Klaster DB disalin ke lingkungan hijau, dan RDS mengonfigurasi replikasi dari klaster DB di lingkungan biru ke klaster DB di lingkungan hijau. RDS juga menyalin semua instans DB di klaster DB.

Topik


- Mempersiapkan deployment blue/green
- Menentukan perubahan saat membuat deployment blue/green
- Membuat deployment blue/green
- Pengaturan untuk membuat penerapan biru/hijau


## Mempersiapkan deployment blue/green

Topik


- Mempersiapkan cluster DB MySQL Aurora untuk penyebaran biru/hijau
- Mempersiapkan cluster DB PostgreSQL Aurora untuk penerapan biru/hijau


## Mempersiapkan cluster DB MySQL Aurora untuk penyebaran biru/hijau

Sebelum Anda membuat deployment blue/green untuk klaster DB Aurora MySQL, klaster harus dikaitkan dengan grup parameter klaster DB kustom dengan pencatatan log biner (`binlog_format`) diaktifkan. Pencatatan log biner diperlukan untuk replikasi dari lingkungan biru ke lingkungan hijau. Meskipun format binlog apa pun berfungsi, kami merekomendasikan `ROW` untuk mengurangi risiko inkonsistensi replikasi. Untuk informasi tentang cara membuat grup parameter klaster DB kustom dan mengatur parameter, lihat the section called “Grup parameter klaster DB”.

> **Note**
>
> Mengaktifkan pencatatan log biner akan meningkatkan jumlah operasi I/O disk tulis untuk klaster DB. Anda dapat memantau penggunaan IOPS dengan `VolumeWriteIOPs` CloudWatch metrik.

Setelah Anda mengaktifkan logging biner, pastikan untuk me-reboot cluster DB sehingga perubahan Anda berlaku. Deployment blue/green mengharuskan instans penulis disinkronkan dengan grup parameter klaster DB, jika tidak, pembuatan akan gagal. Untuk informasi selengkapnya, lihat Mem-boot ulang instans DB dalam klaster Aurora.

Selain itu, kami menyarankan untuk mengubah periode retensi log biner ke nilai selain `NULL` untuk mencegah file log biner dibersihkan. Untuk informasi selengkapnya, lihat the section called “Mengatur dan menampilkan konfigurasi log biner”.

## Mempersiapkan cluster DB PostgreSQL Aurora untuk penerapan biru/hijau

Sebelum Anda membuat penyebaran biru/hijau untuk cluster DB PostgreSQL Aurora, pastikan untuk melakukan hal berikut.

- Kaitkan klaster dengan grup parameter klaster DB kustom yang mengaktifkan replikasi logis (`rds.logical_replication`). Replikasi logika diperlukan untuk replikasi dari lingkungan biru ke lingkungan hijau.

Saat Anda mengaktifkan replikasi logis, Anda juga perlu menyetel parameter cluster tertentu, sepertimax_replication_slots,max_logical_replication_workers, danmax_worker_processes. Untuk instruksi untuk mengaktifkan replikasi logis dan menyetel parameter ini, lihatthe section called “Menyiapkan replikasi logis”.

Selain itu, pastikan bahwa parameter synchronous_commit diatur ke on.

Setelah Anda mengkonfigurasi parameter yang diperlukan, reboot cluster DB sehingga perubahan Anda berlaku. Deployment blue/green mengharuskan instans penulis disinkronkan dengan grup parameter klaster DB, jika tidak, pembuatan akan gagal. Untuk informasi selengkapnya, lihat Mem-boot ulang instans DB dalam klaster Aurora.

- Konfirmasikan bahwa cluster DB Anda menjalankan versi Aurora PostgreSQL yang kompatibel dengan Penerapan Biru/Hijau. Untuk daftar versi yang kompatibel, lihat the section called “Penerapan Biru/Hijau dengan Aurora Postgre SQL”.
- Pastikan bahwa semua tabel di klaster DB memiliki kunci primer. Replikasi logis PostgreSQL tidak mengizinkan operasi UPDATE atau DELETE pada tabel yang tidak memiliki kunci primer.

## Menentukan perubahan saat membuat deployment blue/green

Anda dapat membuat perubahan berikut pada klaster DB di lingkungan hijau saat membuat deployment blue/green.

Anda dapat membuat penyesuaian pada klaster dan instans DB-nya di lingkungan hijau setelah di-deploy. Misalnya, Anda dapat menentukan versi mesin yang lebih tinggi atau grup parameter yang berbeda.

Untuk informasi tentang memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Topik

- Menentukan versi mesin yang lebih tinggi
- Menentukan grup parameter DB yang berbeda

### Menentukan versi mesin yang lebih tinggi

Anda dapat menentukan versi mesin yang lebih tinggi jika ingin menguji peningkatan mesin DB. Setelah switchover, basis data ditingkatkan ke versi mesin DB mayor atau minor yang Anda tentukan.

### Menentukan grup parameter DB yang berbeda

Tentukan grup parameter klaster DB yang berbeda dari yang digunakan oleh klaster DB. Anda dapat menguji bagaimana perubahan parameter memengaruhi klaster DB di lingkungan hijau atau menentukan grup parameter untuk versi mesin DB mayor baru jika terjadi peningkatan.

Jika Anda menentukan grup parameter klaster DB yang berbeda, grup parameter yang ditentukan dikaitkan dengan klaster DB di lingkungan hijau. Jika Anda tidak menentukan grup parameter klaster DB yang berbeda, klaster DB di lingkungan hijau dikaitkan dengan grup parameter yang sama dengan klaster DB biru.

## Membuat deployment blue/green

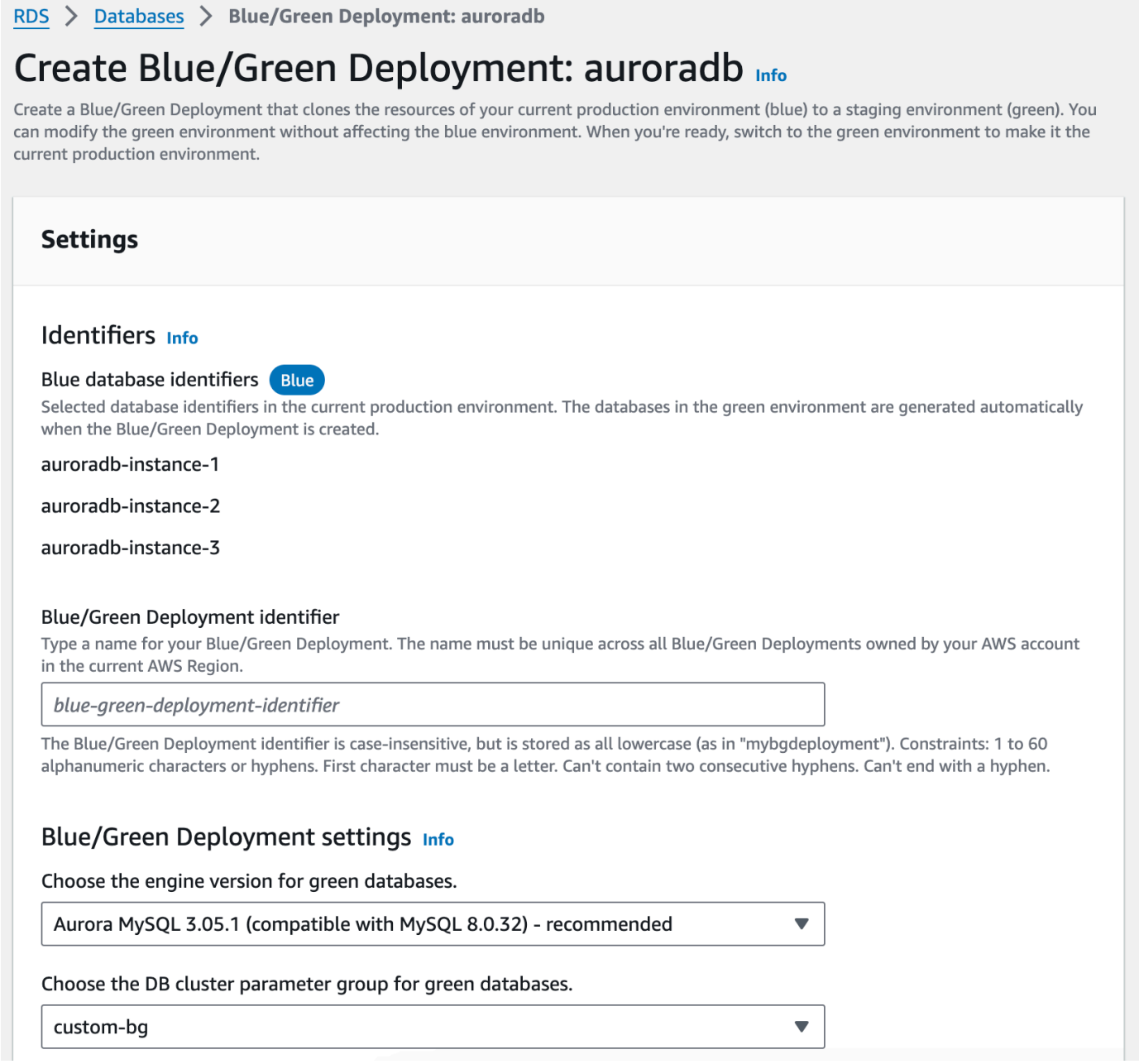
Anda dapat membuat penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

Konsol

Untuk membuat deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB yang ingin disalin ke lingkungan hijau.
3. Pilih Tindakan, Buat penyebaran biru/hijau.

   Halaman Create blue/green deployment muncul.




4. Tinjau pengidentifikasi database biru. Pastikan bahwa mereka cocok dengan instans DB yang Anda harapkan di lingkungan biru. Jika tidak, pilih Batalkan.
5. Untuk nama penyebaran biru/hijau, masukkan nama untuk penyebaran biru/hijau Anda.
6. Di bagian yang tersisa, tentukan pengaturan untuk lingkungan hijau. Untuk informasi tentang setiap pengaturan, lihat the section called “Setelan tersedia”.

   Anda dapat membuat penyesuaian lain pada basis data di lingkungan hijau setelah di-deploy.
7. Pilih Buat.

AWS CLI

Untuk membuat penyebaran biru/hijau menggunakan AWS CLI, gunakan perintah. create-blue-green-deployment Untuk informasi tentang semua opsi yang tersedia, lihatthe section called “Setelan tersedia”.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

Untuk Windows:

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

API RDS

Untuk membuat penerapan biru/hijau dengan menggunakan Amazon RDS API, gunakan operasi. `CreateBlueGreenDeployment` Lihat informasi tentang setiap opsi di the section called “Setelan tersedia”.

## Pengaturan untuk membuat penerapan biru/hijau

Tabel berikut menjelaskan pengaturan yang dapat Anda pilih saat Anda membuat penyebaran biru/hijau. Untuk informasi selengkapnya tentang AWS CLI opsi, lihat create-blue-green-deployment. Untuk informasi selengkapnya tentang parameter RDS API, lihat CreateBlueGreenDeployment.

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
| --- | --- | --- |
| Pengidentifikasi Penerapan Biru/Hijau | Nama untuk penyebaran biru/hijau. | Opsi CLI: |

| Setelan konsol | Deskripsi pengaturan | Opsi CLI dan parameter API RDS |
|---|---|---|
| | | `--blue-green-deployment-name`<br><br>Parameter API:<br><br>`BlueGreenDeploymentName` |
| Pengidentifikasi basis data biru | Pengidentifikasi cluster yang ingin Anda salin ke lingkungan hijau. Saat menggunakan CLI atau API, tentukan cluster Amazon Resource Name (ARN). | Opsi CLI:<br><br>`--source`<br><br>Parameter API:<br><br>`Source` |
| Grup parameter cluster DB untuk database hijau | Sebuah kelompok parameter untuk mengasosiasikan dengan database di lingkungan hijau. | Opsi CLI:<br><br>`--target-db-cluster-parameter-group-name`<br><br>Parameter API:<br><br>`TargetDBClusterParameterGroupName` |
| Versi mesin untuk database hijau | Tingkatkan cluster di lingkungan hijau ke versi mesin DB yang ditentukan.<br><br>Jika Anda memilih Aurora PostgreSQL DB cluster RDS untuk instance PostgreSQL DB logis. Untuk informasi selengkapnya, lihat the section called “Keterbatasan spesifik replikasi logis untuk penerapan blue/green ”. | Opsi CLI:<br><br>`--target-engine-version`<br><br>Parameter API RDS:<br><br>`TargetEngineVersion` |

# Melihat penyebaran biru/hijau di Amazon RDS Aurora

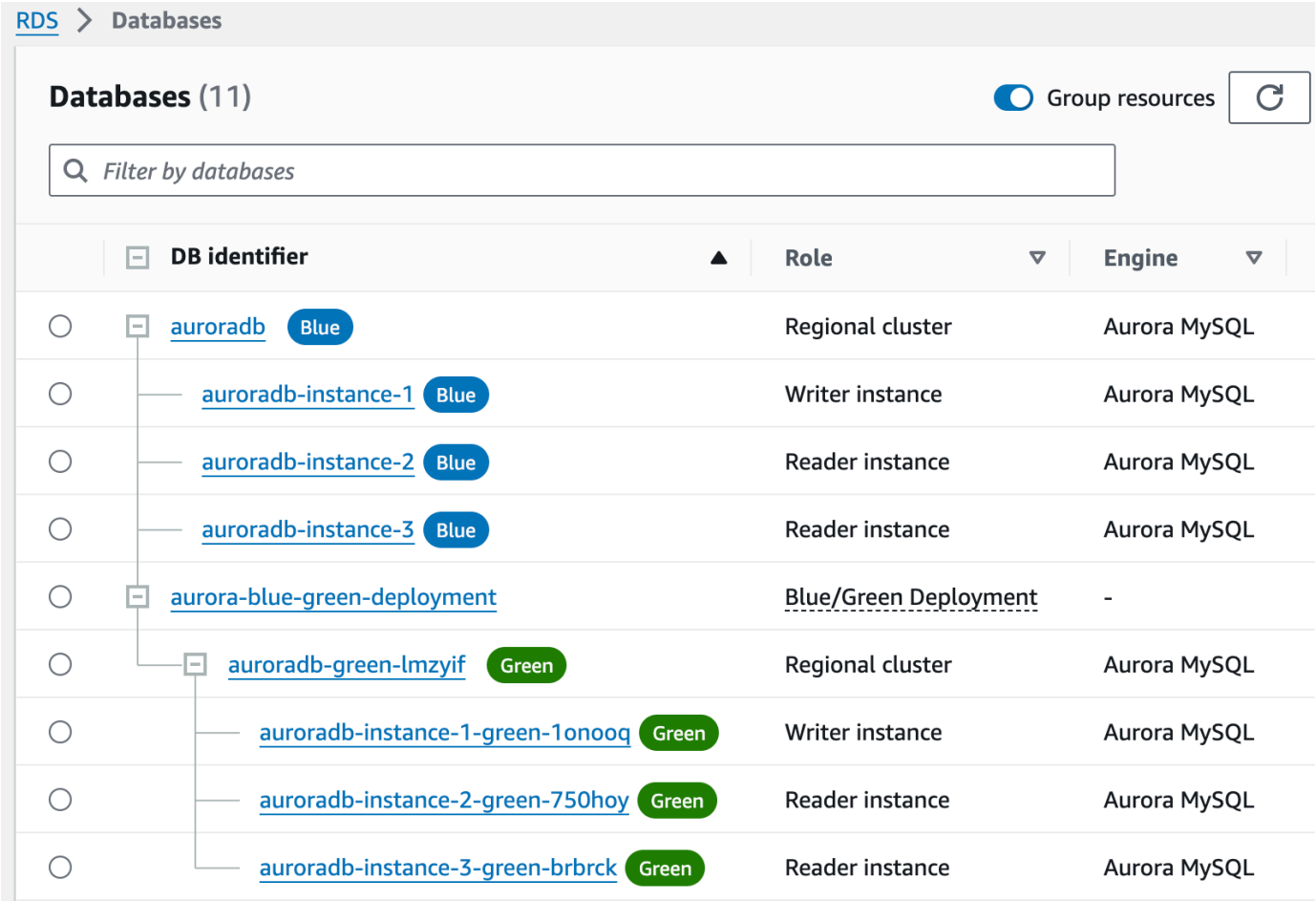
Anda dapat melihat detail tentang penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

Anda juga dapat melihat dan berlangganan acara untuk informasi tentang deployment blue/green. Untuk informasi selengkapnya, lihat Peristiwa deployment blue/green.

## Konsol

Untuk melihat detail tentang deployment blue/green

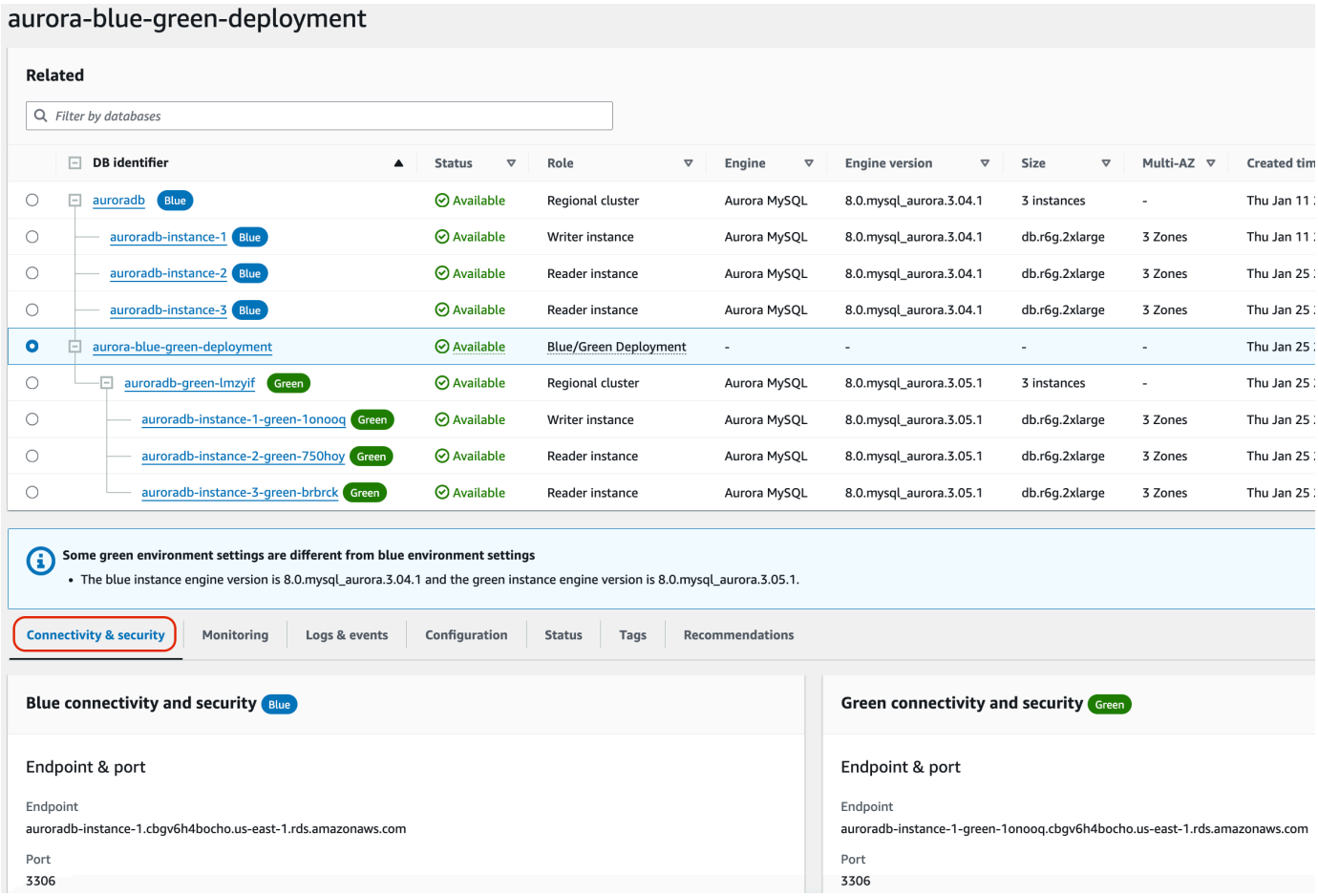
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, dan kemudian temukan deployment blue/green dalam daftar.




   Nilai Peran untuk Deployment adalah Deployment Blue/Green.
3. Pilih nama deployment blue/green hijau yang ingin Anda lihat untuk menampilkan detailnya.

Setiap tab memiliki bagian untuk deployment biru dan bagian untuk deployment hijau. Misalnya, pada tab Konfigurasi, versi mesin DB mungkin berbeda di lingkungan biru dan di lingkungan hijau jika Anda memutakhirkan versi mesin DB di lingkungan hijau.

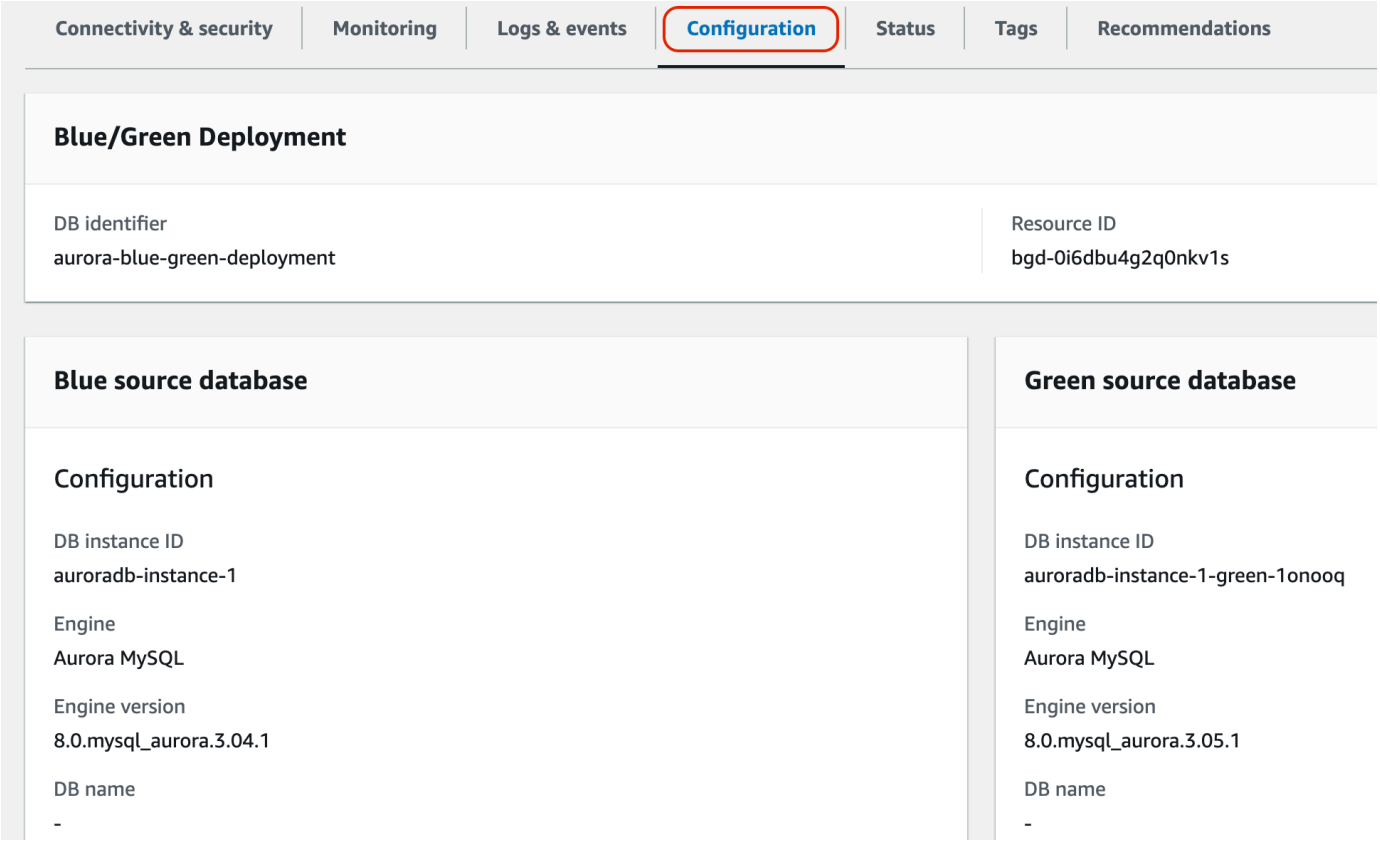
Gambar berikut menunjukkan contoh tab Konektivitas & keamanan:



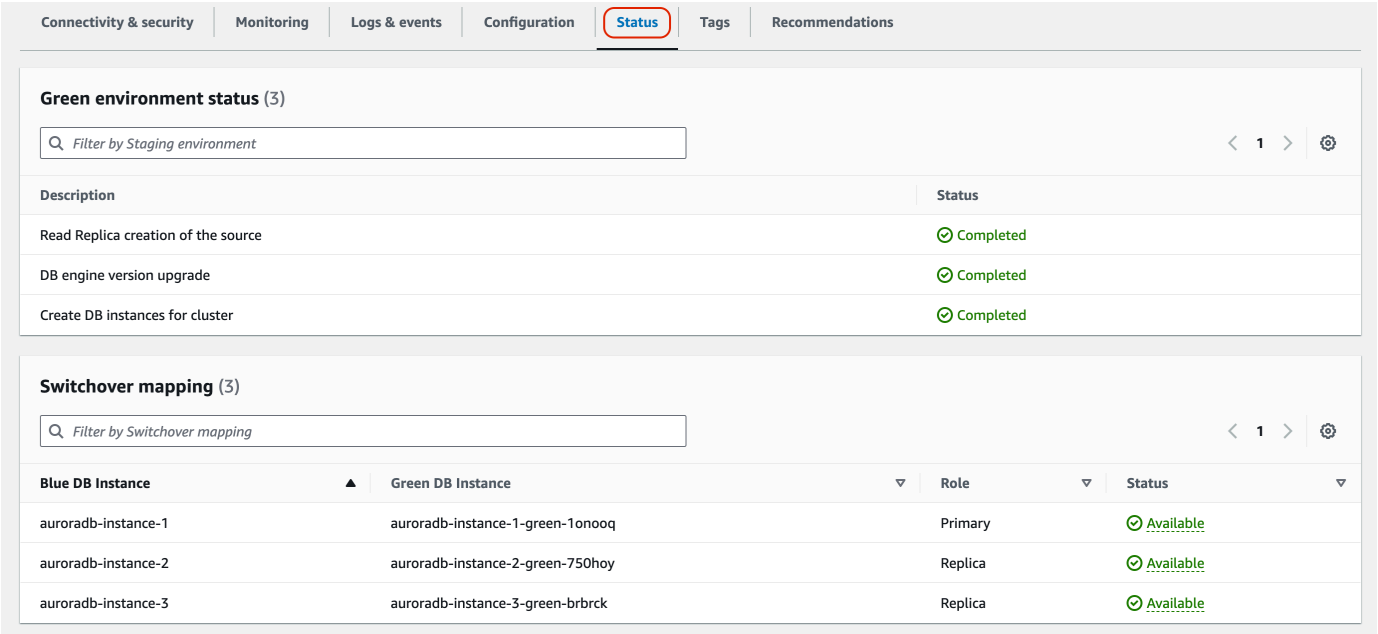

Tab Konektivitas & keamanan juga mencakup bagian yang disebut Replikasi, yang menunjukkan keadaan replikasi dan jeda replika saat ini antara lingkungan biru dan hijau. Jika status replikasi adalah `Replicating`, deployment blue/green berhasil direplikasi.

Untuk Aurora PostgreSQL RDS di lingkungan biru. `Replication degraded` Untuk informasi selengkapnya, lihat the section called “Keterbatasan spesifik replikasi logis untuk penerapan blue/green ”.

Gambar berikut menunjukkan contoh tab Konfigurasi:




Gambar berikut menunjukkan contoh tab Status:



### AWS CLI

Untuk melihat detail tentang penyebaran biru/hijau dengan menggunakan AWS CLI, gunakan perintah. describe-blue-green-deployments

Example Melihat detail tentang deployment blue/green dengan memfilter namanya

Saat Anda menggunakan describe-blue-green-deploymentsperintah, Anda dapat memfilter pada file`--blue-green-deployment-name`.

Contoh berikut menunjukkan detail untuk deployment blue/green bernama *`my-blue-green-deployment`*.

```
aws rds describe-blue-green-deployments \
  --filters Name=blue-green-deployment-name,Values=my-blue-green-deployment
```

Example Melihat detail tentang deployment blue/green dengan menentukan pengidentifikasinya

Saat Anda menggunakan describe-blue-green-deploymentsperintah, Anda dapat menentukan `--blue-green-deployment-identifier` opsi.

Contoh berikut menunjukkan detail untuk deployment blue/green dengan pengidentifikasi *`bgd-1234567890abcdef`*.

```
aws rds describe-blue-green-deployments \
  --blue-green-deployment-identifier bgd-1234567890abcdef
```

### API RDS

Untuk melihat detail tentang deployment blue/green menggunakan Amazon RDS API, gunakan operasi `DescribeBlueGreenDeployments` dan tentukan `BlueGreenDeploymentIdentifier`.

## Mengalihkan penerapan biru/hijau di Amazon RDS Aurora

Switchover mentransisikan cluster DB, termasuk instans DB-nya, di lingkungan hijau menjadi cluster DB produksi. Sebelum Anda switchover, lalu lintas produksi diarahkan ke klaster di lingkungan biru. Setelah Anda switchover, lalu lintas produksi diarahkan ke klaster DB di lingkungan hijau.

Mengalihkan penerapan biru/hijau tidak sama dengan mempromosikan cluster hijau dalam penerapan biru/hijau. Jika Anda secara manual mempromosikan cluster DB hijau dengan memilih

Promosikan dari menu Tindakan, replikasi antara lingkungan biru dan hijau akan rusak dan penerapan biru/hijau memasuki status konfigurasi Tidak valid.

Topik


- Waktu habis switchover
- Pagar pembatas switchover
- Tindakan switchover
- Praktik terbaik switchover
- Memverifikasi CloudWatch metrik sebelum peralihan
- Memantau kelambatan replika sebelum peralihan
- Melakukan switchover pada deployment blue/green
- Setelah switchover


## Waktu habis switchover

Anda dapat menentukan periode waktu habis switchover antara 30 detik dan 3.600 detik (satu jam). Jika switchover memakan waktu lebih lama dari durasi yang ditentukan, maka perubahan apa pun akan dikembalikan dan tidak ada perubahan pada lingkungan mana pun. Periode waktu habis default adalah 300 detik (lima menit).

## Pagar pembatas switchover

Saat Anda memulai switchover, Amazon RDS menjalankan beberapa pemeriksaan dasar untuk menguji kesiapan lingkungan biru dan hijau untuk switchover. Pemeriksaan ini dikenal sebagai pagar pembatas switchover. Pagar pembatas switchover ini mencegah switchover jika lingkungan belum siap. Oleh karena itu, pagar pembatas tersebut dapat mencegah waktu henti yang lebih lama dari yang diharapkan dan mencegah hilangnya data antara lingkungan biru dan hijau yang mungkin terjadi jika switchover dimulai.

Amazon RDS menjalankan pemeriksaan pagar pembatas berikut pada lingkungan hijau:

- Kondisi replikasi – Memeriksa apakah status replikasi klaster DB hijau berkondisi baik. klaster DB hijau adalah replika dari klaster DB biru.
- Jeda replikasi – Memeriksa apakah jeda replika klaster DB hijau berada dalam batas yang diizinkan untuk switchover. Batas yang diizinkan didasarkan pada periode waktu habis yang ditentukan.

Jeda replika menunjukkan seberapa jauh klaster DB tertinggal dari klaster DB biru. Untuk informasi selengkapnya, lihat the section called “Memantau kelambatan replika sebelum peralihan”.

- Penulisan aktif – Memastikan tidak ada penulisan aktif pada klaster DB hijau.

Amazon RDS menjalankan pemeriksaan pagar pembatas berikut pada lingkungan biru:

- Replikasi eksternal — Untuk Aurora , pastikan bahwa lingkungan biru bukan sumber logis (penerbit) atau replika (pelanggan) yang dikelola sendiri. Jika ya, kami sarankan Anda melepaskan slot replikasi dan langganan yang dikelola sendiri di semua database di lingkungan biru, lanjutkan dengan peralihan, lalu buat ulang untuk melanjutkan replikasi. Untuk Aurora MySQL RDS untuk MySQL , periksa apakah database biru bukan replika binlog eksternal. Jika ya, pastikan itu tidak aktif mereplikasi.
- Penulisan aktif yang berjalan lama – Memastikan tidak ada penulisan aktif yang berjalan lama pada klaster DB biru karena dapat meningkatkan jeda replika.
- Pernyataan DDL yang berjalan lama – Memastikan tidak ada pernyataan DDL yang berjalan lama pada instans DB primer biru karena dapat meningkatkan jeda replika.
- Perubahan PostgreSQL yang tidak didukung - Untuk Aurora PostgreSQL . Untuk informasi selengkapnya, lihat the section called “Keterbatasan spesifik replikasi logis untuk penerapan blue/green ”.

  Jika Amazon RDS mendeteksi perubahan PostgreSQL yang tidak didukung, Amazon akan mengubah status replikasi menjadi dan memberi tahu Anda bahwa peralihan tidak `Replication degraded` tersedia untuk penerapan dan semua database hijau. blue/green deployment. To proceed with switchover, we recommend that you delete and recreate the blue/green Untuk melakukannya, pilih Tindakan, Hapus dengan basis data hijau.

## Tindakan switchover

Saat Anda melakukan switchover pada deployment blue/green, RDS melakukan tindakan berikut:

1. Menjalankan pemeriksaan pagar pembatas untuk memverifikasi apakah lingkungan biru dan hijau siap untuk switchover.
2. Menghentikan operasi tulis baru pada instans DB primer di kedua lingkungan.
3. Memutuskan koneksi ke instans DB di kedua lingkungan dan tidak mengizinkan koneksi baru.
4. Menunggu replikasi untuk mengejar ketertinggalan di lingkungan hijau sehingga lingkungan hijau sinkron dengan lingkungan biru.

5. Mengganti nama DB dan instan DB dan klaster di kedua lingkungan.

   RDS mengganti nama DB instans DB dan klaster di lingkungan hijau agar cocok dengan DB instans DB dan klaster di lingkungan biru. Misalnya, asumsikan nama instans DB di lingkungan biru adalah `mydb`. Asumsikan juga nama instans DB yang sesuai di lingkungan hijau adalah `mydb-green-abc123`. Selama switchover, nama instans DB di lingkungan hijau berubah menjadi `mydb`.

   RDS mengganti nama DB instans DB dan klaster di lingkungan biru dengan menambahkan `-old`*n* ke nama saat ini, dengan *n* adalah angka. Misalnya, asumsikan nama instans DB di lingkungan biru adalah `mydb`. Setelah switchover, nama instans DB bisa jadi `mydb-old1`.

   RDS juga mengganti nama titik akhir di lingkungan hijau agar sinkron dengan titik akhir yang sesuai di lingkungan biru sehingga perubahan aplikasi tidak diperlukan.
6. Memungkinkan koneksi ke basis data di kedua lingkungan.
7. Memungkinkan operasi tulis pada instans DB primer di lingkungan produksi baru.

   Setelah switchover, cluster produksi sebelumnya hanya mengizinkan operasi baca ke dan reboot instance DB. `0` Bahkan jika Anda mengaktifkan penulisan di cluster DB, itu tetap hanya-baca sampai Anda menghapus penerapan biru/hijau.

Anda dapat memantau status peralihan menggunakan Amazon. EventBridge Untuk informasi selengkapnya, lihat the section called “Peristiwa deployment blue/green”.

Jika Anda memiliki tag yang dikonfigurasi di lingkungan biru, tag ini disalin ke lingkungan produksi baru selama peralihan. Untuk informasi selengkapnya tentang tag, lihat Menandai sumber daya Amazon Aurora dan Amazon RDS.

Jika switchover dimulai lalu berhenti sebelum selesai karena alasan apa pun, maka perubahan apa pun akan dikembalikan, dan tidak ada perubahan yang diterapkan pada lingkungan mana pun.

## Praktik terbaik switchover

Sebelum melakukan switchover, Anda sangat dianjurkan untuk mengikuti praktik terbaik dengan menyelesaikan tugas-tugas berikut:

- Uji sumber daya secara menyeluruh di lingkungan hijau. Pastikan sumber daya berfungsi dengan baik dan efisien.

- Pantau CloudWatch metrik Amazon yang relevan. Untuk informasi selengkapnya, lihat the section called “Memverifikasi CloudWatch metrik sebelum peralihan”.
- Identifikasi waktu terbaik untuk melakukan switchover.

  Selama switchover, penulisan terputus dari basis data di kedua lingkungan. Identifikasi waktu ketika lalu lintas berada pada titik terendah di lingkungan produksi Anda. Transaksi yang berjalan lama, seperti aktif DDLs, dapat meningkatkan waktu peralihan Anda, menghasilkan waktu henti yang lebih lama untuk beban kerja produksi Anda.

  Jika ada banyak koneksi pada instans DB cluster DB Anda, pertimbangkan untuk menguranginya secara manual ke jumlah minimum yang diperlukan untuk aplikasi Anda sebelum Anda mengalihkan blue/green deployment. One way to achieve this is to create a script that monitors the status of the blue/green penerapan dan mulai membersihkan koneksi ketika mendeteksi bahwa status telah berubah. `SWITCHOVER_IN_PROGRESS`
- Pastikan DB instans DB dan klaster di kedua lingkungan berada dalam status `Available`.
- Pastikan klaster DB di lingkungan hijau berkondisi baik dan bereplikasi.
-

  Jika tidak, aplikasi akan terus mengirimkan lalu lintas tulis ke lingkungan biru setelah switchover.
- Untuk penerapan Aurora PostgreSQL biru/hijau untuk penyebaran biru/hijau PostgreSQL yang menggunakan replikasi logis, lakukan hal berikut:
  - Tinjau batasan replikasi logis dan lakukan tindakan apa pun yang diperlukan sebelum peralihan. Untuk informasi selengkapnya, lihat the section called “Keterbatasan spesifik replikasi logis untuk penerapan biru/green ”.
  - Jalankan operasi `ANALYZE` untuk menyegarkan tabel `pg_statistics`. Ini mengurangi risiko masalah kinerja setelah peralihan.

> **Note**
>
> Selama switchover, Anda tidak dapat mengubah instans DB apa pun yang disertakan dalam switchover.

## Memverifikasi CloudWatch metrik sebelum peralihan

Sebelum Anda mengalihkan penerapan biru/hijau, kami sarankan Anda memeriksa nilai s dari metrik berikut di Amazon. CloudWatch

- `DatabaseConnections` – Gunakan metrik ini untuk memperkirakan tingkat aktivitas pada deployment blue/green, dan pastikan nilainya berada pada tingkat yang dapat diterima untuk deployment Anda sebelum Anda switchover. Jika Wawasan Performa diaktifkan, `DBLoad` adalah metrik yang lebih akurat.
- `ActiveTransactions` – Jika `innodb_monitor_enable` diatur ke `all` dalam grup parameter DB untuk salah satu instans DB Anda, gunakan metrik ini untuk melihat apakah ada banyak transaksi aktif yang mungkin memblokir switchover.

Untuk informasi selengkapnya, lihat the section called “CloudWatch metrik untuk Aurora”.

## Memantau kelambatan replika sebelum peralihan

Sebelum Anda beralih ke penerapan biru/hijau, pastikan bahwa lag replika mendekati nol untuk mengurangi waktu henti.

### Aurora MySQL RDS untuk MySQL dan untuk MariaDB

Untuk penerapan biru/hijau MariaDB, `AuroraBinlogReplicaLag` CloudWatch periksa metrik di lingkungan hijau untuk mengidentifikasi lag replika saat ini. Untuk informasi selengkapnya, lihat the section called “Mendiagnosis dan mengatasi jeda di antara replika baca”.

Untuk , periksa metrik di lingkungan biru untuk mengidentifikasi lag replika `OldestReplicationSlotLag` CloudWatch saat ini. Untuk informasi selengkapnya, lihat the section called “Metrik tingkat instans untuk Amazon Aurora”.

Selain itu, Anda dapat menjalankan query SQL berikut di lingkungan biru:

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

`confirmed_flush_lsn`Ini mewakili nomor urutan log terakhir (LSN) yang dikirim ke replika. `pg_current_wal_lsn`Ini mewakili di mana database sekarang. Sebuah `lsn_distance` dari 0 berarti replika tertangkap.

## Melakukan switchover pada deployment blue/green

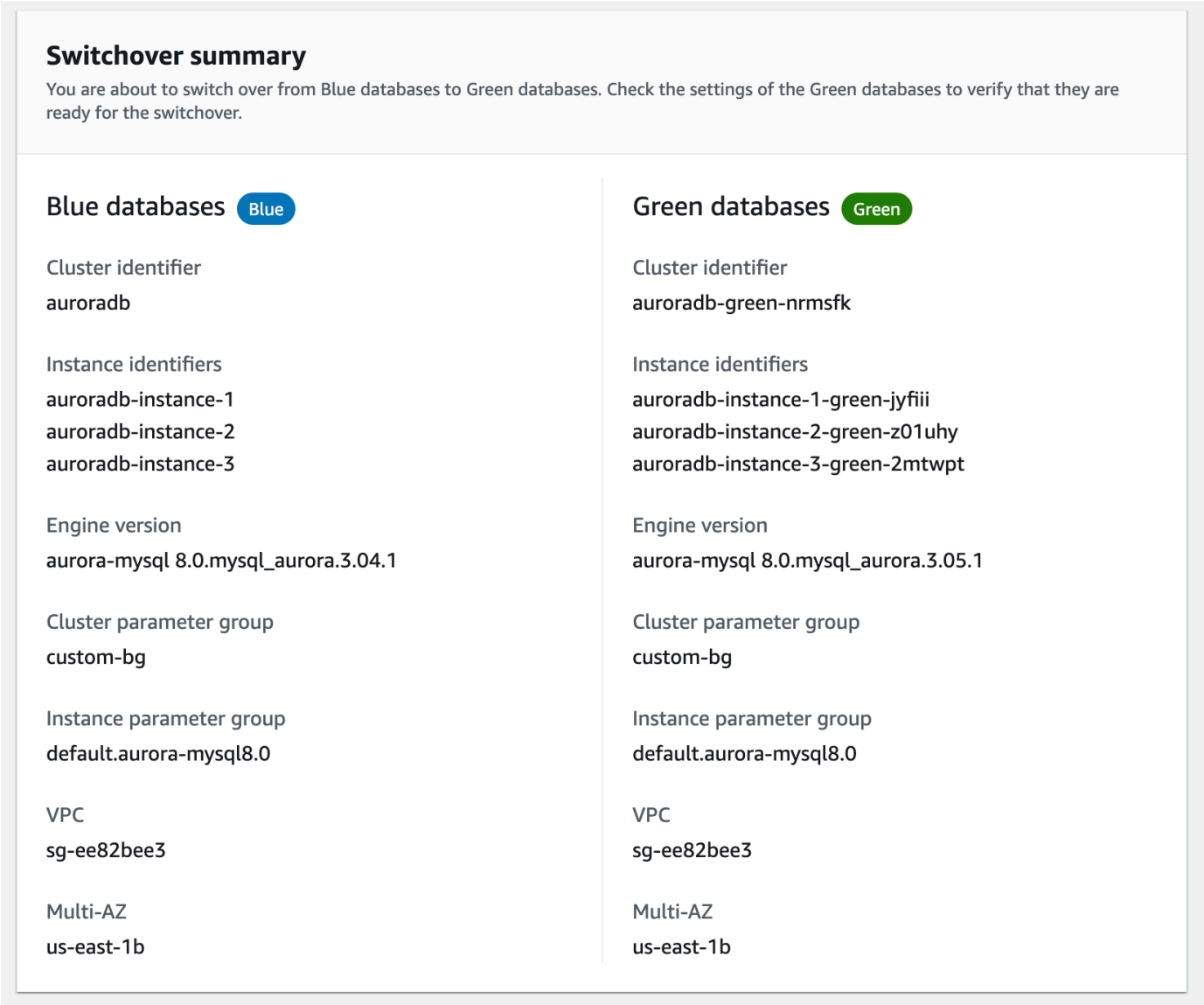
Anda dapat mengalihkan penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

Konsol

Untuk melakukan switchover pada deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih deployment blue/green yang ingin Anda switchover.
3. Untuk Tindakan, pilih Switchover.

   Halaman Switchover muncul.




4. Di halaman Switchover, tinjau ringkasan switchover. Pastikan sumber daya di kedua lingkungan sesuai dengan yang Anda harapkan. Jika tidak, pilih Batalkan.
5. Untuk Pengaturan waktu habis, masukkan batas waktu untuk switchover.
6. Jika klaster menjalankan Aurora PostgreSQL, tinjau dan konfirmasi rekomendasi pra-switchover. Untuk informasi selengkapnya, lihat the section called “Keterbatasan spesifik replikasi logis untuk penerapan blue/green ”.
7. Pilih Switchover.

AWS CLI

Untuk beralih penyebaran biru/hijau dengan menggunakan AWS CLI, gunakan switchover-blue-green-deploymentperintah dengan opsi berikut:

- `--blue-green-deployment-identifier`— Tentukan ID sumber daya dari penerapan biru/hijau.
- `--switchover-timeout` – Tentukan waktu habis untuk switchover, dalam hitungan detik. Angka default-nya adalah 300.

Example Melakukan switchover pada deployment blue/green

Untuk Linux, macOS, atau Unix:

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

Untuk Windows:

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

API RDS

Untuk melakukan switchover pada deployment blue/green menggunakan API Amazon RDS, gunakan operasi `SwitchoverBlueGreenDeployment` dengan parameter berikut:

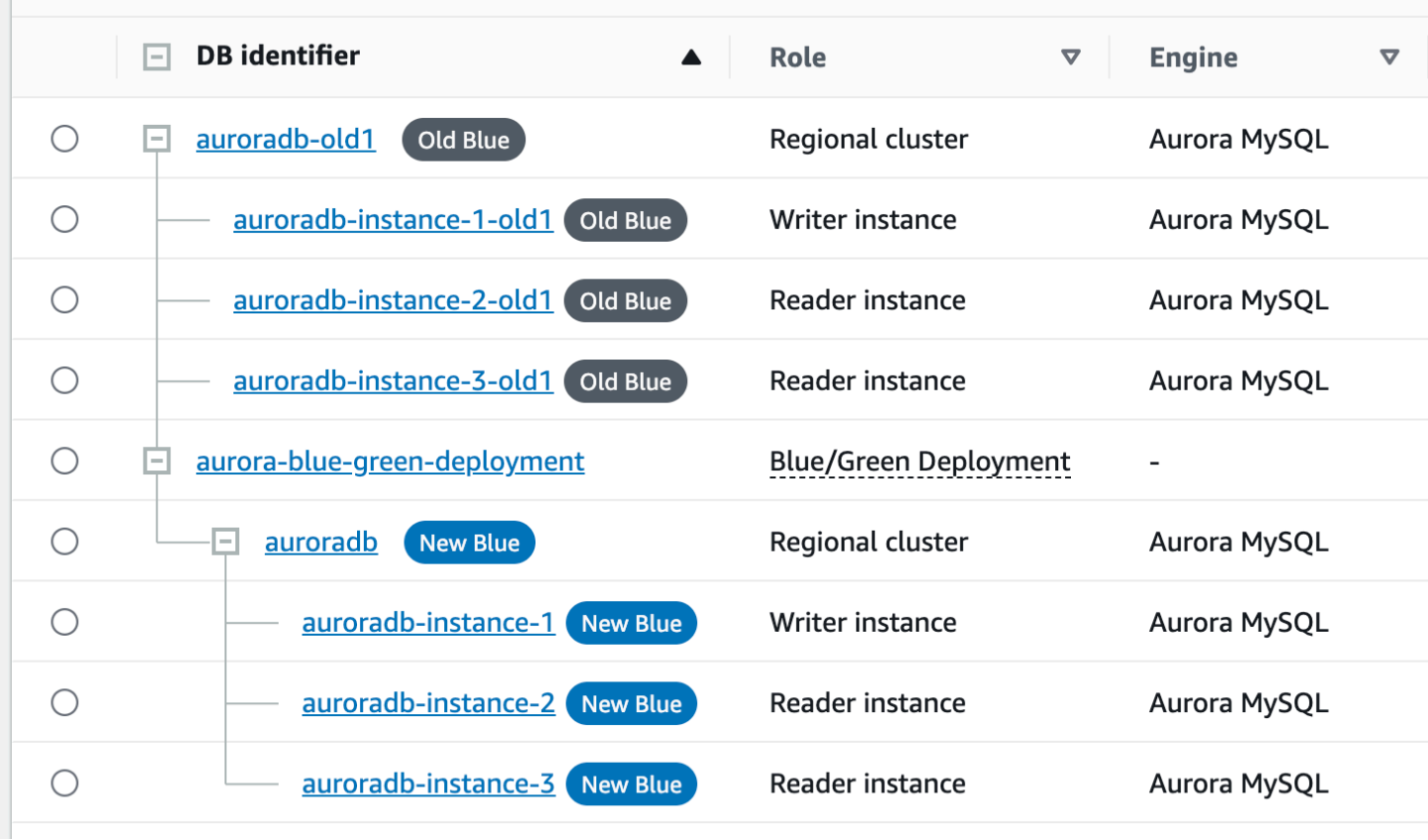
- `BlueGreenDeploymentIdentifier`— Tentukan ID sumber daya dari penerapan biru/hijau.
- `SwitchoverTimeout` – Tentukan waktu habis untuk switchover, dalam hitungan detik. Angka default-nya adalah 300.

## Setelah switchover

Setelah switchover, DB instans DB dan klaster di lingkungan biru sebelumnya akan dipertahankan. Biaya standar berlaku untuk sumber daya ini. Replikasi dan pencatatan log biner antara lingkungan biru dan hijau berhenti.

RDS mengganti nama DB instans DB dan klaster di lingkungan biru dengan menambahkan `-old`*n* ke nama sumber daya saat ini, dengan *n* adalah angka. Cluster DB biru dipaksa menjadi status hanya-baca. Bahkan jika Anda mengaktifkan operasi tulis, itu tetap hanya-baca sampai Anda menghapus penerapan biru/hijau. RDS menamai cluster DB dan instans DB di lingkungan hijau. `-new`*n*

Jika Anda menghapus sumber daya penyebaran biru/hijau, RDS mempertahankan sumber daya dan sumber daya. `-old`*n* `-new`*n*




## Memperbarui node induk untuk konsumen

RDS menawarkan replika baca yang dikelola sepenuhnya. Namun, ini juga menyediakan opsi untuk mengatur replika yang dikelola sendiri, juga dikenal sebagai replika eksternal. Replika eksternal memungkinkan Anda menggunakan sumber daya pihak ketiga sebagai target replikasi.

Setelah Anda mengalihkan penyebaran biru/hijau, jika cluster DB instans DB memiliki replika eksternal atau konsumen log biner sebelum peralihan, Anda harus memperbarui node induk mereka setelah peralihan untuk mempertahankan kontinuitas replikasi.

Untuk memperbarui node induk

1. Setelah peralihan, instance DB penulis yang sebelumnya berada di lingkungan hijau memancarkan peristiwa yang berisi nama file log master dan posisi log master. Untuk menemukan acara, navigasikan ke konsol RDS dan pilih Acara dari panel navigasi kiri.
2. Filter berdasarkan peristiwa di mana sumbernya adalah nama instance DB penulis hijau tua, sebelum beralih.
3. Temukan acara yang berisi koordinat log biner. Pesan acara mirip dengan:`Binary log coordinates in green environment after switchover: file mysql-bin-changelog.000003 and position 40134574`.
4. Pastikan bahwa konsumen atau replika telah menerapkan semua log biner dari lingkungan biru tua. Kemudian, gunakan koordinat log biner yang disediakan untuk melanjutkan replikasi pada konsumen. Misalnya, jika Anda menjalankan replika MySQL, Anda dapat menggunakan EC2 perintah berikut:

   MySQL 8.0.22 dan versi mayor dan minor yang lebih rendah

   ```
   CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-changelog.000003', MASTER_LOG_POS=40134574;
   ```

   MySQL 8.0.23 dan versi mayor dan minor yang lebih tinggi

   ```
   CHANGE REPLICATION SOURCE TO SOURCE_HOST='{new-writer-endpoint}',
    SOURCE_LOG_FILE='mysql-bin-changelog.000003', SOURCE_LOG_POS=40134574;
   ```

# Menghapus penyebaran biru/hijau di Amazon RDS Aurora

Anda dapat menghapus deployment blue/green sebelum atau setelah switchover.

Saat Anda menghapus deployment blue/green sebelum switchover, Amazon RDS secara opsional menghapus klaster DB di lingkungan hijau:

- Jika Anda memilih untuk menghapus klaster DB di lingkungan hijau (`--delete-target`), klaster harus menonaktifkan perlindungan penghapusan.
- Jika Anda tidak menghapus klaster DB di lingkungan hijau (`--no-delete-target`), klaster tersebut dipertahankan, tetapi tidak lagi menjadi bagian dari deployment blue/green. Untuk ,

replikasi berlanjut antar lingkungan. Untuk Aurora PostgreSQL, lingkungan hijau dipromosikan ke lingkungan mandiri, sehingga replikasi berhenti.

Opsi untuk menghapus basis data hijau tidak tersedia di konsol setelah switchover. Saat Anda menghapus penerapan biru/hijau menggunakan AWS CLI, Anda tidak dapat menentukan `--delete-target` opsi jika status penerapannya. `SWITCHOVER_COMPLETED`

> **Important**
>
> Setelah Anda menghapus penerapan biru/hijau, RDS menghapus perlindungan hanya-baca dari cluster DB produksi sebelumnya. Jika `read_only` parameter dinonaktifkan untuk cluster DB, ia mulai mengizinkan operasi tulis lagi.

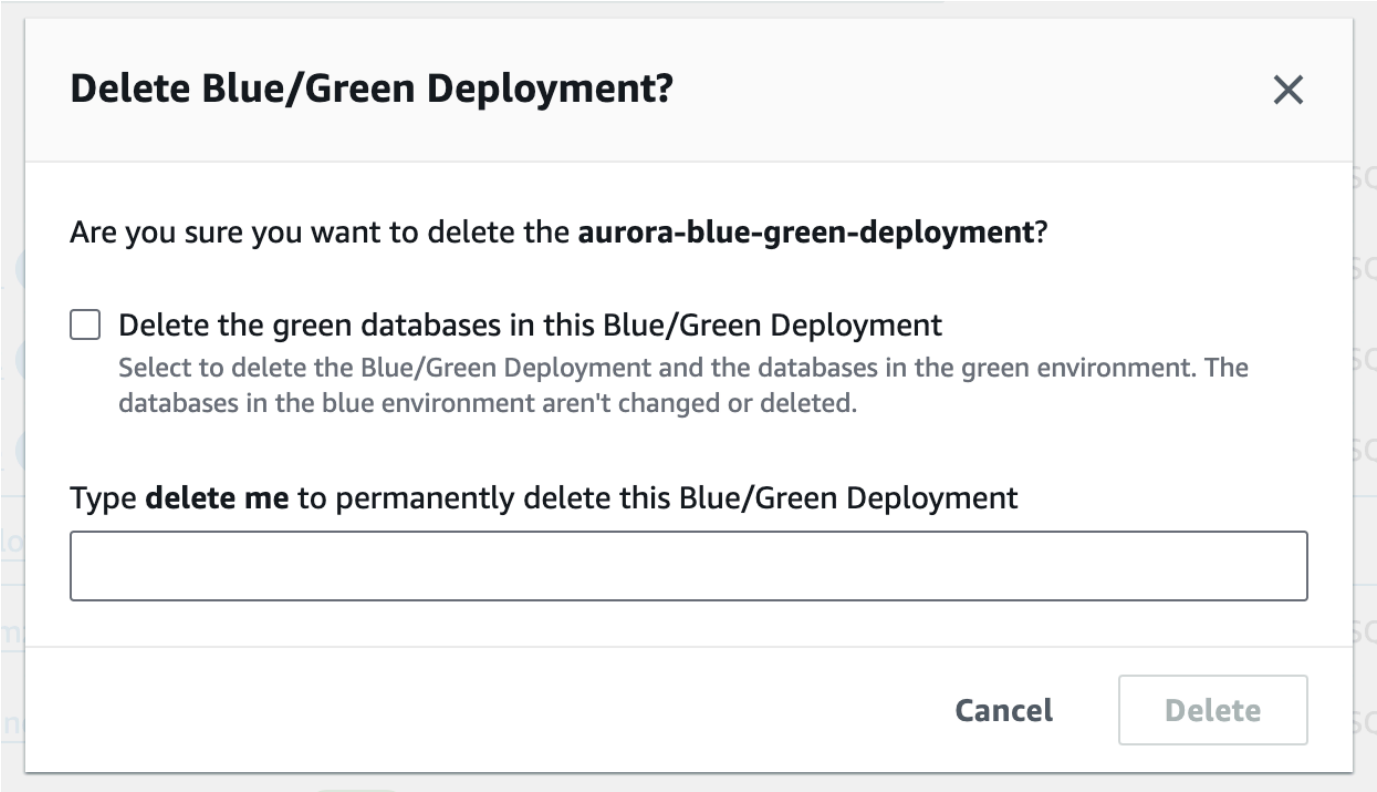
Anda dapat menghapus penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

## Konsol

Untuk menghapus deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih deployment blue/green yang ingin dihapus.
3. Untuk Tindakan, pilih Hapus.

   Penerapan Hapus biru/hijau? jendela muncul.




Untuk menghapus database hijau, pilih Hapus database hijau dalam penyebaran biru/hijau ini.

4. Masukkan **`delete me`** di kotak.
5. Pilih Hapus.

## AWS CLI

Untuk menghapus penyebaran biru/hijau dengan menggunakan AWS CLI, gunakan delete-blue-green-deploymentperintah dengan opsi berikut:

- `--blue-green-deployment-identifier`— ID sumber daya dari penyebaran biru/hijau yang akan dihapus.
- `--delete-target` – Menentukan bahwa klaster DB di lingkungan hijau dihapus. Anda tidak dapat menentukan opsi ini jika deployment blue/green memiliki status `SWITCHOVER_COMPLETED`.
- `--no-delete-target` – Menentukan bahwa klaster DB di lingkungan hijau dipertahankan.

Example Menghapus deployment blue/green dan klaster DB di lingkungan hijau

Untuk Linux, macOS, atau Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

Untuk Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example Menghapus deployment blue/green tetapi mempertahankan klaster DB di lingkungan hijau

Untuk Linux, macOS, atau Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

Untuk Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

## API RDS

Untuk menghapus deployment blue/green menggunakan API Amazon RDS, gunakan operasi `DeleteBlueGreenDeployment` dengan parameter berikut:

- `BlueGreenDeploymentIdentifier`— ID sumber daya dari penyebaran biru/hijau yang akan dihapus.
- `DeleteTarget` – Menentukan `TRUE` untuk menghapus klaster DB di lingkungan hijau atau `FALSE` untuk mempertahankannya. Tidak bisa `TRUE` jika deployment blue/green memiliki status `SWITCHOVER_COMPLETED`.

# Mencadangkan dan memulihkan klaster DB Amazon Aurora

Topik ini menyediakan informasi tentang mencadangkan dan memulihkan klaster DB Amazon Aurora.

> **Tip**
>
> Fitur ketersediaan tinggi Aurora dan kapabilitas pencadangan otomatis membantu menjaga data Anda tetap aman tanpa memerlukan pengaturan yang ekstensif dari Anda. Sebelum Anda menerapkan strategi pencadangan, pelajari cara Aurora menyimpan banyak salinan data Anda dan membantu Anda mengaksesnya di beberapa instans dan Wilayah DB. AWS Untuk detailnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.

Topik


- Gambaran umum pencadangan dan pemulihan klaster DB Aurora
- Mempertahankan cadangan otomatis
- Memahami penggunaan penyimpanan cadangan Amazon Aurora
- Membuat snapshot klaster DB
- Memulihkan dari snapshot klaster DB
- Penyalinan snapshot cluster DB
- Berbagi snapshot klaster DB
- Mengekspor data klaster DB ke Amazon S3
- Mengekspor data snapshot klaster DB ke Amazon S3
- Memulihkan klaster DB ke waktu tertentu
- Menghapus snapshot klaster DB
- Tutorial: Memulihkan klaster DB Amazon Aurora dari snapshot klaster DB

# Gambaran umum pencadangan dan pemulihan klaster DB Aurora

Topik berikut menjelaskan cadangan Aurora dan cara memulihkan klaster DB Aurora Anda.

Daftar Isi


- Cadangan
  - Penggunaan AWS Backup
- Jendela cadangan
- Memulihkan data
- Kloning basis data untuk Aurora
- Backtrack


## Cadangan

Aurora mencadangkan volume klaster Anda secara otomatis dan mempertahankan data yang dipulihkan selama periode retensi cadangan. Cadangan otomatis Aurora bersifat kontinu dan inkremental, sehingga Anda dapat memulihkan dengan cepat ke titik mana pun dalam periode retensi cadangan. Tidak ada dampak performa atau gangguan layanan basis data saat data cadangan ditulis. Anda dapat menentukan periode retensi cadangan selama 1–35 hari saat Anda membuat atau memodifikasi klaster DB. Cadangan otomatis disimpan di Amazon S3. Untuk informasi selengkapnya tentang mempertahankan backup otomatis, lihat Mempertahankan cadangan otomatis

Jika Anda ingin mempertahankan data di luar periode retensi cadangan, Anda dapat mengambil snapshot data di volume klaster Anda. Snapshot klaster DB Aurora tidak kedaluwarsa. Anda dapat membuat klaster DB baru dari snapshot. Untuk informasi selengkapnya, lihat Membuat snapshot klaster DB.

> **Note**
>
> - Untuk klaster DB Amazon Aurora, periode retensi cadangan default adalah satu hari, terlepas dari cara klaster DB dibuat.
> - Anda tidak dapat menonaktifkan cadangan otomatis di Aurora. Periode retensi cadangan untuk Aurora dikelola oleh klaster DB.

Biaya penyimpanan cadangan Anda tergantung pada jumlah cadangan Aurora dan data snapshot yang Anda pertahankan dan berapa lama Anda mempertahankannya. Untuk informasi tentang penyimpanan yang terkait dengan cadangan dan snapshot Aurora, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. Untuk informasi harga tentang penyimpanan cadangan Aurora, lihat harga Amazon untuk RDS Aurora. Setelah klaster Aurora yang terkait dengan snapshot dihapus, penyimpanan snapshot tersebut akan menimbulkan biaya penyimpanan cadangan standar untuk Aurora.

### Penggunaan AWS Backup

Anda dapat menggunakan AWS Backup untuk mengelola cadangan cluster Amazon Aurora DB.

Snapshot dikelola oleh AWS Backup dianggap snapshot cluster DB manual, tetapi tidak dihitung dalam kuota snapshot cluster DB untuk Aurora. Snapshot yang dibuat dengan AWS Backup memiliki nama dengan`awsbackup:job-`*`AWS-Backup-job-number`*. Untuk informasi lebih lanjut tentang AWS Backup, lihat AWS Panduan Pengembang Cadangan .

Anda juga dapat menggunakan AWS Backup untuk mengelola cadangan otomatis cluster Amazon Aurora DB. Jika cluster DB Anda dikaitkan dengan rencana cadangan di AWS Backup, Anda dapat menggunakan rencana cadangan itu untuk point-in-time pemulihan. Pencadangan otomatis (berkelanjutan) yang dikelola oleh AWS Backup memiliki nama dengan`continuous:cluster-`*`AWS-Backup-job-number`*. Untuk informasi selengkapnya, lihat Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup.

## Jendela cadangan

Pencadangan otomatis terjadi setiap hari selama jendela cadangan yang dipilih. Jika pencadangan memerlukan waktu lebih dari yang dialokasikan untuk jendela cadangan, pencadangan akan berlanjut setelah periode berakhir, hingga selesai. Jendela cadangan tidak dapat tumpang tindih dengan jendela pemeliharaan mingguan untuk klaster DB.

Cadangan otomatis Aurora bersifat terus-menerus dan bertahap, namun jendela cadangan digunakan untuk membuat cadangan sistem harian yang dipertahankan dalam periode retensi cadangan. Anda dapat menyalin cadangan untuk mempertahankannya di luar periode retensi.

> **Note**
>
> Saat Anda membuat cluster DB menggunakan AWS Management Console, Anda tidak dapat menentukan jendela cadangan. Namun, Anda dapat menentukan jendela cadangan saat Anda membuat cluster DB menggunakan AWS CLI atau RDSAPI.

Jika Anda tidak menentukan jendela cadangan yang diinginkan saat Anda membuat klaster DB, Aurora akan menetapkan jendela cadangan 30 menit default. Jendela ini dipilih secara acak dari blok waktu 8 jam untuk masing-masing Wilayah AWS. Tabel berikut mencantumkan blok waktu untuk masing-masing Wilayah AWS dari mana jendela cadangan default ditetapkan.

| Nama Wilayah | Wilayah | Blok Waktu |
|---|---|---|
| US East (Northern Virginia) | us-east-1 | 03:00 — 11:00 UTC |
| AS Timur (Ohio) | us-east-2 | 03:00 — 11:00 UTC |
| US West (Northern California) | us-west-1 | 06:00 — 14:00 UTC |
| AS Barat (Oregon) | us-west-2 | 06:00 — 14:00 UTC |
| Afrika (Cape Town) | af-south-1 | 03:00 — 11:00 UTC |
| Asia Pasifik (Hong Kong) | ap-east-1 | 06:00 — 14:00 UTC |
| Asia Pasifik (Hyderabad) | ap-south-2 | 06:30 — 14:30 UTC |
| Asia Pasifik (Jakarta) | ap-southeast-3 | 08:00 — 16:00 UTC |
| Asia Pasifik (Malaysia) | ap-southeast-5 | 09:00 — 17:00 UTC |
| Asia Pasifik (Melbourne) | ap-southeast-4 | 11:00 — 19:00 UTC |

| Nama Wilayah | Wilayah | Blok Waktu |
|---|---|---|
| Asia Pasifik (Mumbai) | ap-south-1 | 16:30 — 00:30 UTC |
| Asia Pacific (Osaka) | ap-northeast-3 | 00:00 — 08:00 UTC |
| Asia Pasifik (Seoul) | ap-northeast-2 | 13:00 — 21:00 UTC |
| Asia Pasifik (Singapura) | ap-southeast-1 | 14:00 — 22:00 UTC |
| Asia Pasifik (Sydney) | ap-southeast-2 | 12:00 — 20:00 UTC |
| Asia Pasifik (Tokyo) | ap-northeast-1 | 13:00 — 21:00 UTC |
| Kanada (Pusat) | ca-central-1 | 03:00 — 11:00 UTC |
| Kanada Barat (Calgary) | ca-west-1 | 18:00 — 02:00 UTC |
| Tiongkok (Beijing) | cn-north-1 | 06:00 — 14:00 UTC |
| Tiongkok (Ningxia) | cn-northwest-1 | 06:00 — 14:00 UTC |
| Eropa (Frankfurt) | eu-central-1 | 20:00 — 04:00 UTC |
| Eropa (Irlandia) | eu-west-1 | 22:00 — 06:00 UTC |
| Eropa (London) | eu-west-2 | 22:00 — 06:00 UTC |
| Eropa (Milan) | eu-south-1 | 02:00 — 10:00 UTC |
| Eropa (Paris) | eu-west-3 | 07:29 — 14:29 UTC |
| Eropa (Spanyol) | eu-south-2 | 02:00 — 10:00 UTC |
| Eropa (Stockholm) | eu-north-1 | 23:00 — 07:00 UTC |
| Eropa (Zürich) | eu-central-2 | 02:00 — 10:00 UTC |
| Israel (Tel Aviv) | il-central-1 | 03:00 — 11:00 UTC |

| Nama Wilayah | Wilayah | Blok Waktu |
|---|---|---|
| Timur Tengah (Bahrain) | me-south-1 | 06:00 — 14:00 UTC |
| Timur Tengah (UAE) | me-central-1 | 05:00 — 13:00 UTC |
| Amerika Selatan (Sao Paulo) | sa-east-1 | 23:00 — 07:00 UTC |
| AWS GovCloud (AS-Timur) | us-gov-east-1 | 17:00 — 01:00 UTC |
| AWS GovCloud (AS-Barat) | us-gov-west-1 | 06:00 — 14:00 UTC |

## Memulihkan data

Anda dapat memulihkan data Anda dengan membuat klaster DB Aurora baru dari data cadangan yang dipertahankan Aurora, dari snapshot klaster DB yang telah Anda simpan, atau dari cadangan otomatis yang dipertahankan. Anda dapat dengan cepat memulihkan salinan baru klaster DB yang dibuat dari data cadangan ke titik waktu mana pun selama periode retensi cadangan Anda. Karena cadangan Aurora bersifat kontinu dan inkremental selama periode retensi cadangan, Anda tidak perlu sering mengambil snapshot data untuk meningkatkan waktu pemulihan.

Waktu pemulihan terbaru untuk klaster DB adalah titik terbaru yang dapat Anda untuk memulihkan klaster DB Anda. Titik ini biasanya 5 menit dari waktu saat ini untuk klaster DB aktif, atau 5 menit dari waktu penghapusan klaster untuk cadangan otomatis yang dipertahankan.

Opsi Waktu pemulihan paling awal menentukan seberapa jauh ke belakang dalam periode retensi cadangan yang dapat Anda tentukan untuk memulihkan volume klaster Anda.

Untuk menentukan waktu restorasi terbaru atau paling awal untuk cluster DB, cari `Earliest restorable time` nilai `Latest restorable time` atau di konsol. RDS Untuk informasi tentang melihat nilai-nilai ini, lihat Melihat cadangan otomatis yang dipertahankan untuk Amazon Aurora.

Anda dapat menentukan kapan pemulihan klaster DB selesai dengan memeriksa nilai `Latest restorable time` dan `Earliest restorable time`. Nilai-nilai ini kembali NULL sampai

operasi pemulihan selesai. Anda tidak dapat meminta operasi pencadangan atau pemulihan jika salah satu `Latest restorable time` atau `Earliest restorable time` kembaliNULL.

Untuk informasi tentang memulihkan klaster DB ke waktu yang ditentukan, lihat Memulihkan klaster DB ke waktu tertentu.

## Kloning basis data untuk Aurora

Anda juga dapat menggunakan kloning basis data untuk mengkloning basis data klaster DB Aurora Anda ke klaster DB baru, alih-alih memulihkan snapshot klaster DB. Basis data klon hanya menggunakan ruang tambahan yang minimal saat pertama kali dibuat. Data disalin hanya seiring perubahannya, misalnya pada basis data sumber atau basis data klon. Anda dapat membuat beberapa klon dari klaster DB yang sama, atau membuat klon tambahan bahkan dari klon lain. Untuk informasi selengkapnya, lihat Mengkloning volume untuk klaster DB Amazon Aurora.

## Backtrack

Aurora My SQL sekarang mendukung “memutar ulang” cluster DB ke waktu tertentu, tanpa memulihkan data dari cadangan. Untuk informasi selengkapnya, lihat Melakukan backtracking klaster DB Aurora.

# Mempertahankan cadangan otomatis

Saat Anda menghapus yang disediakan atau Aurora Serverless v2 DB cluster, Anda dapat mempertahankan backup otomatis. Hal ini memungkinkan Anda memulihkan klaster DB ke titik waktu tertentu dalam periode retensi cadangan, bahkan setelah klaster dihapus.

Cadangan otomatis yang dipertahankan berisi snapshot sistem dan log transaksi dari klaster DB. Cadangan ini juga mencakup properti klaster DB, seperti kelas instans DB, yang diperlukan untuk memulihkannya ke klaster aktif.

Anda dapat memulihkan atau menghapus cadangan otomatis yang dipertahankan menggunakan AWS Management Console, RDSAPI, dan AWS CLI.

> **Note**
>
> Anda tidak dapat menyimpan cadangan otomatis untuk Aurora Serverless v1 klaster DB.

Topik


- Periode penahanan
- Biaya retensi
- Mencegah penghapusan cadangan otomatis
- Batasan
- Melihat cadangan otomatis yang dipertahankan untuk Amazon Aurora
- Menghapus cadangan otomatis yang dipertahankan untuk Amazon Aurora


## Periode penahanan

Snapshot sistem dan log transaksi dalam cadangan otomatis yang dipertahankan akan kedaluwarsa dengan cara yang sama seperti untuk klaster DB sumber. Pengaturan untuk periode retensi klaster sumber juga berlaku untuk cadangan otomatis. Karena tidak ada snapshot atau log baru yang dibuat untuk klaster ini, cadangan otomatis yang dipertahankan pada akhirnya kedaluwarsa sepenuhnya. Setelah periode retensi selesai, Anda terus mempertahankan snapshot klaster DB manual, tetapi semua cadangan otomatis akan kedaluwarsa.

Anda dapat menghapus cadangan otomatis yang dipertahankan menggunakan konsol, AWS CLI atau RDSAPI. Untuk informasi selengkapnya, lihat Menghapus cadangan otomatis yang dipertahankan untuk Amazon Aurora.

Tidak seperti cadangan otomatis yang dipertahankan, snapshot akhir tidak kedaluwarsa. Kami sangat menyarankan agar Anda mengambil snapshot akhir bahkan jika Anda mempertahankan cadangan otomatis karena cadangan otomatis yang dipertahankan pada akhirnya akan kedaluwarsa.

## Biaya retensi

Tidak ada biaya tambahan untuk penyimpanan cadangan hingga 100% dari total penyimpanan basis data Aurora Anda untuk setiap klaster DB Aurora. Selain itu, tidak ada biaya tambahan hingga satu hari ketika Anda mempertahankan cadangan otomatis setelah menghapus klaster DB. Cadangan yang Anda pertahankan selama lebih dari satu hari akan dikenai biaya.

Tidak ada biaya tambahan untuk log transaksi atau metadata instans. Semua aturan penetapan harga lainnya untuk cadangan berlaku untuk klaster yang dapat dipulihkan. Untuk informasi selengkapnya, lihat halaman Harga Amazon Aurora.

## Mencegah penghapusan cadangan otomatis

Amazon RDS menghapus cadangan otomatis dalam beberapa situasi:

- Pada akhir periode retensi.
- Jika Anda menonaktifkan pencadangan otomatis untuk klaster DB.
- Jika Anda menghapus klaster DB.

Jika Anda ingin menyimpan cadangan otomatis untuk jangka waktu yang lebih lama, salin cadangan otomatis tersebut untuk membuat snapshot manual, yang akan dipertahankan hingga Anda menghapusnya. Biaya RDS penyimpanan Amazon mungkin berlaku untuk snapshot manual jika melebihi ruang penyimpanan default Anda.

Untuk informasi selengkapnya tentang menyalin snapshot cluster DB, lihat. Penyalinan snapshot cluster DB

Untuk informasi selengkapnya tentang biaya penyimpanan cadangan, lihat RDSharga Amazon.

## Batasan

Batasan berikut berlaku untuk cadangan otomatis yang dipertahankan:

- Jumlah maksimum backup otomatis yang dipertahankan dalam satu AWS Wilayah adalah 40. Ini tidak termasuk dalam kuota untuk klaster DB. Anda dapat memiliki hingga 40 klaster DB yang berjalan, 40 instans DB yang berjalan, dan 40 cadangan otomatis yang dipertahankan untuk klaster DB secara bersamaan.

  Untuk informasi selengkapnya, lihat Kuota dalam Amazon Aurora.
- Cadangan otomatis yang dipertahankan tidak berisi informasi tentang parameter atau grup opsi.
- Anda dapat memulihkan klaster yang dihapus ke titik waktu yang berada dalam periode retensi pada saat penghapusan.
- Anda tidak dapat memodifikasi cadangan otomatis yang dipertahankan karena cadangan ini terdiri dari cadangan sistem, log transaksi, dan properti klaster DB yang ada saat Anda menghapus klaster sumber.

## Melihat cadangan otomatis yang dipertahankan untuk Amazon Aurora

Untuk melihat cadangan otomatis yang disimpan di RDS konsol, pilih Pencadangan otomatis di panel navigasi, lalu pilih Dipertahankan. Untuk melihat snapshot individual yang terkait dengan cadangan otomatis yang dipertahankan, pilih Snapshot di panel navigasi. Alternatifnya, Anda dapat mendeskripsikan snapshot terpisah yang terkait dengan cadangan otomatis yang dipertahankan. Dari sana, Anda dapat memulihkan instans DB langsung dari salah satu snapshot tersebut.

Untuk menggambarkan cadangan otomatis Anda yang dipertahankan menggunakan AWS CLI, gunakan perintah berikut:

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

Untuk menjelaskan cadangan otomatis Anda yang dipertahankan menggunakan RDSAPI, panggil `DescribeDBClusterAutomatedBackups`tindakan dengan parameter. `DbClusterResourceId`

## Menghapus cadangan otomatis yang dipertahankan untuk Amazon Aurora

Anda dapat menghapus cadangan otomatis yang tersimpan ketika tidak diperlukan lagi. Untuk menghapus cadangan otomatis yang disimpan menggunakan AWS Management Console, AWS CLI, atau Amazon RDS API, gunakan prosedur berikut.

Konsol

Untuk menghapus cadangan otomatis yang dipertahankan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pencadangan otomatis.
3. Pilih tab Dipertahankan.

4. Pilih cadangan otomatis yang dipertahankan yang ingin Anda hapus.
5. Untuk Tindakan, pilih Hapus.
6. Di halaman konfirmasi, masukkan **`delete me`** dan pilih Hapus.

AWS CLI

Anda dapat menghapus cadangan otomatis yang dipertahankan dengan menggunakan AWS CLI perintah delete-db-cluster-automated-backup dengan opsi berikut:

- `--db-cluster-resource-id` – Pengidentifikasi sumber daya untuk klaster DB sumber.

  Anda dapat menemukan pengenal sumber daya untuk cluster DB sumber dari cadangan otomatis yang dipertahankan dengan menjalankan AWS CLI perintah describe-db-cluster-automated -backup.

Example

Contoh ini menghapus cadangan otomatis yang dipertahankan untuk klaster DB sumber yang memiliki ID sumber daya `cluster-123ABCEXAMPLE`.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster-automated-backup \
```

```
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

Untuk Windows:

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

API RDS

Anda dapat menghapus cadangan otomatis yang dipertahankan dengan menggunakan operasi Amazon RDS API Delete DBCluster AutomatedBackup dengan parameter berikut:

- `DbClusterResourceId` – Pengidentifikasi sumber daya untuk klaster DB sumber.

  Anda dapat menemukan pengenal sumber daya untuk instans DB sumber cadangan otomatis yang dipertahankan menggunakan operasi Amazon RDS API Describe. DBCluster AutomatedBackups

# Memahami penggunaan penyimpanan cadangan Amazon Aurora

Amazon Aurora memelihara dua jenis cadangan: cadangan otomatis (berkelanjutan) dan snapshot.

## Penyimpanan cadangan otomatis

Cadangan otomatis (berkelanjutan) untuk klaster secara inkremental menyimpan semua perubahan basis data dalam periode retensi tertentu agar dapat memulihkan ke titik waktu mana pun dalam periode retensi tersebut. Periode retensi dapat berkisar 1–35 hari. Cadangan otomatis bersifat inkremental dan ditagih berdasarkan jumlah penyimpanan yang diperlukan untuk memulihkan ke waktu mana pun dalam periode retensi.

Aurora juga menyediakan jumlah penggunaan cadangan gratis. Jumlah penggunaan gratis ini sama dengan ukuran volume cluster terbaru (seperti yang ditunjukkan oleh CloudWatch metrik `VolumeBytesUsed` Amazon). Jumlah ini dikurangi dari penggunaan cadangan otomatis yang dihitung. Selan itu, tidak ada biaya untuk cadangan otomatis yang periode retensinya hanya 1 hari.

Misalnya, cadangan otomatis Anda memiliki periode retensi 7 hari, dan Anda ingin memulihkan klaster Anda ke statusnya dari empat hari yang lalu. Aurora menggunakan data inkremental yang tersimpan dalam cadangan otomatis untuk membuat kembali status klaster pada waktu yang persis empat hari lalu.

Cadangan otomatis menyimpan semua informasi yang diperlukan agar dapat memulihkan klaster ke waktu mana pun dalam periode retensi. Itu berarti bahwa cadangan otomatis menyimpan semua perubahan selama periode retensi, termasuk penulisan informasi baru atau penghapusan informasi yang ada. Untuk basis data yang memiliki banyak perubahan yang terjadi, ukuran cadangan otomatis akan tumbuh dari waktu ke waktu. Setelah basis data berhenti mengalami perubahan, Anda dapat mendapati bahwa ukuran cadangan otomatis berkurang karena perubahan tersimpan sebelumnya berada di luar periode retensi.

Total penggunaan yang ditagih untuk cadangan otomatis tidak pernah melebihi ukuran volume klaster kumulatif selama periode retensi. Misalnya, jika periode retensi Anda adalah 7 hari, dan volume klaster Anda 100 GB setiap hari, maka penggunaan cadangan otomatis yang ditagih tidak pernah melebihi 700 GB (100 GB * 7).

## Penyimpanan snapshot

Snapshot cluster DB selalu merupakan cadangan penuh yang menangkap ukuran volume cluster saat Anda membuatnya. Baik Anda mengambil snapshot secara manual atau melalui rencana

AWS Pencadangan, Aurora memperlakukannya sebagai snapshot manual. Aurora menyediakan penyimpanan gratis tanpa batas untuk snapshot dalam periode retensi cadangan otomatis. Setelah snapshot manual berada di luar periode ini, ia dikenakan biaya per GB-bulan. Snapshot sistem otomatis tetap gratis kecuali Anda menyalinnya. Karena pencadangan otomatis tidak mencakup salinan snapshot, AWS selalu menagih untuk mereka.

Untuk informasi umum tentang cadangan Aurora, lihat Cadangan. Untuk informasi tentang harga penyimpanan cadangan Aurora, lihat halaman harga Amazon Aurora.

## CloudWatch Metrik Amazon untuk penyimpanan cadangan Aurora

Anda dapat memantau klaster Aurora dan membuat laporan menggunakan CloudWatch metrik Amazon melalui konsol. CloudWatch Anda dapat menggunakan CloudWatch metrik berikut untuk meninjau dan memantau jumlah penyimpanan yang digunakan oleh cadangan Aurora Anda. Metrik ini dihitung secara independen untuk setiap klaster Aurora DB.

- `BackupRetentionPeriodStorageUsed` – Merupakan jumlah penyimpanan cadangan yang digunakan, dalam byte, untuk menyimpan cadangan otomatis pada saat ini.
  - Nilainya tergantung pada ukuran volume klaster dan jumlah perubahan (penulisan dan pembaruan) yang dibuat untuk klaster DB selama periode retensi. Hal ini karena cadangan otomatis harus menyimpan semua perubahan inkremental yang dilakukan pada klaster agar dapat memulihkan ke titik waktu mana pun.
  - Metrik ini tidak mengurangi tingkat gratis penggunaan cadangan yang disediakan Aurora.
  - Metrik ini menghasilkan satu titik data harian untuk penggunaan cadangan otomatis yang dicatat pada hari itu.
- `SnapshotStorageUsed` – Merupakan jumlah penyimpanan cadangan yang digunakan, dalam byte, untuk menyimpan snapshot manual di luar periode retensi cadangan otomatis.
  - Nilainya tergantung pada jumlah snapshot yang dipertahankan di luar periode retensi cadangan otomatis dan ukuran masing-masing snapshot.
  - Ukuran masing-masing snapshot adalah ukuran volume klaster pada saat Anda mengambil snapshot.
  - Snapshot adalah cadangan penuh, bukan inkremental.
  - Metrik ini menghasilkan satu titik data harian untuk setiap snapshot yang diisi. Untuk mengetahui total penggunaan snapshot harian Anda, gunakan jumlah metrik ini selama 1 hari.
- `TotalBackupStorageBilled` – Merupakan metrik untuk semua penggunaan cadangan yang ditagih, dalam byte, untuk klaster yang ditentukan:

`BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier`

- Metrik ini menghasilkan satu titik data harian untuk nilai `BackupRetentionPeriodStorageUsed` dikurangi tingkat gratis penggunaan cadangan yang disediakan Aurora. Tingkat gratis ini sama dengan ukuran volume klaster DB yang terakhir dicatat. Titik data ini merepresentasikan penggunaan sebenarnya yang ditagih untuk cadangan otomatis.
- Metrik ini menghasilkan titik data harian individu untuk semua nilai `SnapshotStorageUsed`.
- Untuk mengetahui total penggunaan cadangan harian yang ditagih, gunakan jumlah metrik ini selama 1 hari. Metrik ini menjumlahkan semua penggunaan snapshot yang ditagih dengan penggunaan cadangan otomatis yang ditagih, untuk menghasilkan total penggunaan cadangan yang ditagih.

Untuk informasi selengkapnya tentang cara menggunakan CloudWatch metrik, lihatKetersediaan metrik Aurora di konsol Amazon RDS.

## Menghitung penggunaan penyimpanan cadangan

Penggunaan untuk cadangan otomatis dihitung dengan melihat semua catatan inkremental yang harus disimpan agar dapat memulihkan ke titik waktu mana pun dalam periode retensi cadangan. Perubahan ini tidak hanya mencakup jumlah operasi tulis, tetapi juga ukuran dan ruang lingkup modifikasi data. Setiap jenis operasi (INSERT, UPDATE, DELETE) membuat catatan perubahan yang harus disimpan untuk point-in-time pemulihan. Oleh karena itu, sementara dua database mungkin memiliki jumlah operasi tulis (IOPS) yang sama, persyaratan penyimpanan cadangannya dapat berbeda secara signifikan berdasarkan volume data yang dimodifikasi dalam setiap transaksi.

Misalnya, Anda memiliki cadangan otomatis dengan periode retensi 7 hari. Ukuran volume klaster Anda tepat sebelum periode retensi adalah 100 GB. Dengan demikian, itulah jumlah paling sedikit yang perlu disimpan Aurora. Kemudian, Anda memiliki aktivitas di bawah selama 7 hari ke depan, dengan ukuran catatan inkremental yang merupakan jumlah penyimpanan yang diperlukan untuk menyimpan catatan perubahan yang berasal dari penulisan dan pembaruan basis data Anda.

| Hari | Ukuran catatan inkremental (GB) |
|---|---|
| 1 | 10 |
| 2 | 15 |

| Hari | Ukuran catatan inkremental (GB) |
|---|---|
| 3 | 25 |
| 4 | 20 |
| 5 | 10 |
| 6 | 25 |
| 7 | 30 |
| Total | 135 |

Data ini menunjukkan bahwa penggunaan cadangan otomatis yang dihitung untuk cadangan Anda adalah sebagai berikut:

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

Kemudian, kurangi tingkat penggunaan gratis dengan penggunaan yang ditagih. Asumsikan bahwa ukuran terbaru volume Anda adalah 200 GB:

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

## FAQs

Kapan saya ditagih untuk snapshot?

Anda ditagih untuk snapshot manual yang berada di luar (lebih lama dari) periode retensi cadangan otomatis.

Apa itu snapshot manual?

Snapshot manual adalah snapshot yang memenuhi ketentuan berikut ini:

- Diminta secara manual oleh Anda
- Diambil oleh layanan backup otomatis seperti AWS Backup
- Disalin dari snapshot sistem otomatis untuk mempertahankannya di luar periode retensi

Apa yang terjadi pada snapshot manual saya jika saya menghapus klaster DB saya?

Snapshot manual tidak kedaluwarsa sampai Anda menghapusnya.

Saat Anda menghapus klaster DB Anda, snapshot manual yang sebelumnya Anda ambil terus ada. Jika snapshot ini sebelumnya tidak ditagih karena berada dalam periode retensi cadangan otomatis, sekarang snapshot tersebut tidak tercakup lagi dan semuanya akan mulai ditagih menurut ukuran penuh untuk penggunaannya.

Bagaimana cara mengurangi biaya penyimpanan cadangan saya?

Ada beberapa cara untuk mengurangi biaya terkait penggunaan cadangan:

- Hapus snapshot manual yang berada di luar periode penyimpanan cadangan otomatis Anda. Ini termasuk snapshot yang Anda ambil, serta snapshot yang mungkin diambil oleh paket AWS Backup Anda. Pastikan untuk memeriksa paket AWS Backup untuk memastikan tidak menyimpan snapshot di luar periode retensi yang tidak Anda harapkan.
- Evaluasi penulisan dan pembaruan ke basis data Anda untuk melihat apakah Anda dapat mengurangi jumlah perubahan yang Anda buat. Karena cadangan otomatis kami menyimpan semua perubahan inkremental dalam periode retensi, mengurangi jumlah pembaruan yang Anda buat juga mengurangi biaya cadangan otomatis Anda.
- Evaluasi apakah mengurangi periode retensi cadangan otomatis Anda layak untuk dilakukan. Mengurangi periode retensi berarti bahwa cadangan menyimpan data inkremental dalam lebih sedikit hari, yang dapat mengurangi biaya cadangan secara keseluruhan. Namun, mengurangi periode retensi ini juga dapat menyebabkan beberapa snapshot mulai ditagih karena snapshot tersebut sekarang berada di luar periode retensi. Pastikan untuk memeriksa semua biaya snapshot tambahan yang mungkin Anda keluarkan sebelum memutuskan apakah ini tindakan yang tepat untuk Anda.

Bagaimana penyimpanan cadangan ditagih?

Penyimpanan cadangan ditagih berdasarkan GB-bulan.

Hal ini berarti bahwa penggunaan penyimpanan cadangan ditagih sebagai penggunaan rata-rata berbobot selama bulan tertentu. Berikut adalah beberapa contoh untuk bulan 30 hari:

- Penggunaan cadangan yang ditagih adalah 100 GB untuk total 30 hari dalam sebulan. Biaya Anda adalah sebagai berikut:

```
(100 GB * 30) / 30 = 100 GB-month
```

- Penggunaan cadangan yang ditagih adalah 100 GB untuk 15 hari pertama dalam sebulan, kemudian 0 GB untuk 15 terakhir. Biaya Anda adalah sebagai berikut:

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- Penggunaan cadangan yang ditagih adalah 50 GB untuk 10 hari pertama dalam sebulan, 100 GB untuk 10 hari ke depan, kemudian 150 GB untuk 10 hari terakhir. Biaya Anda adalah sebagai berikut:

```
(50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
```

Bagaimana pengaturan pelacakan mundur untuk klaster DB saya memengaruhi penggunaan penyimpanan cadangan?

Pengaturan pelacakan mundur untuk klaster Aurora DB tidak memengaruhi volume data cadangan untuk klaster tersebut. Amazon menagih penyimpanan untuk pelacakan mundur data secara terpisah. Untuk informasi harga pelacakan mundur Aurora, lihat halaman harga Amazon Aurora.

Bagaimana biaya penyimpanan berlaku untuk snapshot yang dibagikan?

Jika Anda berbagi snapshot dengan pengguna lain, Anda masih menjadi pemilik snapshot tersebut. Biaya penyimpanan berlaku untuk pemilik snapshot. Jika Anda menghapus snapshot yang dibagikan yang Anda miliki, tidak ada yang dapat mengaksesnya.

Untuk mempertahankan akses ke snapshot yang dibagikan yang dimiliki orang lain, Anda dapat menyalin snapshot tersebut. Dengan begitu, Anda menjadi pemilik snapshot baru. Setiap biaya penyimpanan untuk snapshot yang disalin berlaku untuk akun Anda.

Untuk informasi selengkapnya tentang berbagi snapshot, lihat Berbagi snapshot klaster DB. Untuk informasi selengkapnya tentang menyalin snapshot, lihat Penyalinan snapshot cluster DB.

# Membuat snapshot klaster DB

Amazon RDS membuat snapshot volume penyimpanan cluster DB Anda, mencadangkan seluruh cluster DB dan bukan hanya database individual. Saat Anda membuat snapshot klaster DB, Anda perlu mengidentifikasi klaster DB yang akan dicadangkan, dan kemudian memberikan nama snapshot klaster DB Anda agar Anda dapat memulihkannya di lain waktu. Jumlah waktu yang diperlukan untuk membuat snapshot klaster DB bervariasi sesuai ukuran basis data Anda. Karena snapshot menyertakan seluruh volume penyimpanan, ukuran file, seperti file sementara, juga memengaruhi jumlah waktu yang diperlukan untuk membuat snapshot.

> **Note**
>
> Klaster DB Anda harus dalam status `available` untuk mengambil snapshot klaster DB.

Tidak seperti pencadangan otomatis, snapshot manual tidak bergantung pada periode retensi pencadangan. Snapshots tidak kedaluwarsa.

Untuk cadangan jangka sangat panjang, sebaiknya Anda mengekspor data snapshot ke Amazon S3. Jika versi utama mesin DB Anda tidak didukung lagi, Anda tidak dapat memulihkan ke versi tersebut dari snapshot. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Anda dapat membuat snapshot cluster DB menggunakan AWS Management Console, file AWS CLI, atau file. RDS API

## Konsol

Untuk membuat snapshot klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.

   Daftar Snapshot manual akan muncul.
3. Pilih Ambil snapshot.

   Jendela Ambil snapshot DB akan muncul.
4. Untuk jenis Snapshot, pilih DB cluster.

5. Pilih cluster DB yang ingin Anda ambil snapshot.
6. Masukkan nama Snapshot.
7. Pilih Ambil snapshot.

   Daftar snapshot Manual muncul, dengan status snapshot cluster DB baru ditampilkan sebagai `Creating` Setelah statusnya adalah `Available`, Anda dapat melihat waktu pembuatannya.

## AWS CLI

Saat Anda membuat snapshot cluster DB menggunakan AWS CLI, Anda perlu mengidentifikasi cluster DB mana yang akan Anda cadangkan, dan kemudian beri nama snapshot cluster DB Anda sehingga Anda dapat memulihkannya nanti. Anda dapat melakukan ini dengan menggunakan AWS CLI `create-db-cluster-snapshot`perintah dengan parameter berikut:

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

Dalam contoh ini, Anda membuat snapshot klaster DB yang diberi nama *`mydbclustersnapshot`* untuk kluster DB yang disebut *`mydbcluster`*.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Untuk Windows:

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

## RDS API

Saat Anda membuat snapshot cluster DB menggunakan Amazon RDSAPI, Anda perlu mengidentifikasi cluster DB mana yang akan Anda cadangkan, lalu beri nama snapshot cluster DB

Anda sehingga Anda dapat memulihkannya nanti. Anda dapat melakukan ini dengan menggunakan RDS API `CreateDBClusterSnapshot`perintah Amazon dengan parameter berikut:

- DBClusterIdentifier
- DBClusterSnapshotIdentifier

## Menentukan apakah snapshot klaster DB tersedia

Anda dapat memeriksa apakah snapshot cluster DB tersedia dengan melihat di bawah Snapshots pada tab Maintenance & backup pada halaman detail untuk cluster di AWS Management Console, dengan menggunakan `describe-db-cluster-snapshots`CLIperintah, atau dengan menggunakan tindakan. `DescribeDBClusterSnapshots`API

Anda juga dapat menggunakan `wait db-cluster-snapshot-available`CLIperintah untuk melakukan polling API setiap 30 detik hingga snapshot tersedia.

# Memulihkan dari snapshot klaster DB

Amazon RDS membuat snapshot volume penyimpanan instans basis data Anda, sehingga mencadangkan seluruh klaster DB dan bukan hanya masing-masing basis data. Anda dapat membuat klaster DB baru dengan memulihkannya dari snapshot DB. Anda memberikan nama snapshot klaster DB untuk memulihkannya, kemudian memberikan nama klaster DB baru yang dibuat dari pemulihan tersebut. Anda tidak dapat memulihkan dari snapshot klaster DB ke klaster DB yang sudah ada; klaster DB baru dibuat saat Anda memulihkan.

**Important**

Jika Anda mencoba mengembalikan snapshot ke versi mesin DB yang tidak digunakan lagi, peningkatan langsung ke versi mesin terbaru akan terjadi. Cluster DB yang dipulihkan dari versi engine yang tidak digunakan lagi tidak akan dapat diakses hingga ditingkatkan ke versi utama yang lebih baru.
Selain itu, biaya Extended Support mungkin berlaku jika versi menggunakan Extended Support atau telah mencapai akhir dukungan standar. Untuk informasi selengkapnya, lihat Amazon RDS Extended Support dengan .

Anda dapat menggunakan klaster DB yang dipulihkan segera setelah statusnya `available`.

Anda dapat menggunakan AWS CloudFormation untuk memulihkan cluster DB dari snapshot cluster DB. Untuk informasi selengkapnya, lihat AWS: :RDS:: DBCluster di AWS CloudFormation Panduan Pengguna.

**Note**

Berbagi snapshot cluster DB manual, baik terenkripsi atau tidak terenkripsi, memungkinkan AWS akun resmi untuk secara langsung memulihkan cluster DB dari snapshot alih-alih mengambil salinannya dan memulihkannya. Untuk informasi selengkapnya, lihat Berbagi snapshot klaster DB.

Untuk informasi tentang memulihkan klaster Aurora DB atau klaster global dengan versi RDS Extended Support, lihat. Aurora DB atau cluster global dengan Amazon Extended Support RDS

## Pertimbangan grup parameter

Kami sarankan agar Anda mempertahankan grup parameter DB dan grup parameter klaster DB untuk snapshot klaster DB yang Anda buat, sehingga Anda dapat menautkan klaster DB yang dipulihkan dengan grup parameter yang benar.

Grup parameter DB dan grup parameter klaster DB default dikaitkan dengan klaster yang dipulihkan, kecuali jika Anda memilih yang berbeda. Tidak ada pengaturan parameter kustom yang tersedia di grup parameter default.

Anda dapat menentukan grup parameter saat memulihkan klaster DB.

Untuk informasi selengkapnya tentang grup parameter DB dan grup parameter klaster DB, lihat .

## Pertimbangan grup keamanan

Saat Anda memulihkan klaster DB, cloud privat virtual (VPC) default, grup subnet DB, dan grup keamanan VPC akan dikaitkan dengan instans yang dipulihkan, kecuali jika Anda memilih yang berbeda.

- Jika Anda menggunakan konsol Amazon RDS, Anda dapat menentukan grup keamanan VPC kustom untuk ditautkan dengan klaster atau membuat grup keamanan VPC baru.
- Jika Anda menggunakan AWS CLI, Anda dapat menentukan grup keamanan VPC kustom untuk dikaitkan dengan cluster dengan menyertakan `--vpc-security-group-ids` opsi dalam perintah. `restore-db-cluster-from-snapshot`
- Jika Anda menggunakan API Amazon RDS, Anda dapat menyertakan `VpcSecurityGroupIds.VpcSecurityGroupId.N` di dalam tindakan `RestoreDBClusterFromSnapshot`.

Segera setelah pemulihan selesai dan klaster DB baru Anda tersedia, Anda juga dapat mengubah pengaturan VPC dengan memodifikasi klaster DB. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

## Pertimbangan Amazon Aurora

Dengan Aurora, Anda dapat memulihkan snapshot klaster DB ke klaster DB.

Dengan Aurora MySQL dan Aurora PostgreSQL, Anda juga dapat memulihkan snapshot klaster DB ke klaster DB Aurora Serverless. Untuk informasi selengkapnya, lihat Memulihkan Aurora Serverless v1 klaster DB.

Dengan Aurora MySQL, Anda dapat memulihkan snapshot klaster DB dari klaster tanpa kueri paralel ke klaster dengan kueri paralel. Karena kueri paralel biasanya digunakan dengan tabel yang sangat besar, mekanisme snapshot adalah cara tercepat untuk menyerap data dalam volume besar ke klaster Aurora MySQL yang memiliki kueri paralel aktif. Untuk informasi selengkapnya, lihat Kueri paralel untuk Amazon Aurora My SQL.

## Memulihkan dari snapshot

Anda dapat memulihkan klaster DB dari snapshot klaster DB menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Untuk memulihkan klaster DB dari snapshot klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot klaster DB untuk memulihkan klaster.
4. Untuk Tindakan, pilih Pulihkan snapshot.

   Halaman Pulihkan snapshot ditampilkan.
5. Pilih versi mesin DB yang Anda inginkan untuk memulihkan klaster DB.

   Secara default, snapshot dipulihkan ke versi mesin DB yang sama dengan klaster DB sumber, jika versi tersebut tersedia.
6. Untuk Pengidentifikasi instans DB, masukkan nama untuk instans DB yang dipulihkan. Perhatikan bahwa Amazon RDS memperoleh pengidentifikasi cluster DB dari pengidentifikasi instans DB yang Anda tentukan.
7. Tentukan pengaturan lain, seperti konfigurasi penyimpanan klaster DB.

   Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.
8. Pilih Pulihkan klaster DB.

AWS CLI

Untuk mengembalikan cluster DB dari snapshot cluster DB, gunakan AWS CLI perintah restore-db-cluster-from-snapshot.

Dalam contoh ini, Anda memulihkan dari snapshot klaster DB yang dibuat sebelumnya yang bernama `mydbclustersnapshot`. Anda memulihkan ke klaster DB baru yang bernama `mynewdbcluster`.

Anda dapat menentukan pengaturan lain, seperti versi mesin DB. Jika Anda tidak menentukan versi mesin, klaster DB dipulihkan ke versi mesin default.

Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

Setelah klaster DB dipulihkan, Anda harus menambahkannya ke grup keamanan yang digunakan oleh klaster DB ini untuk membuat snapshot klaster DB jika Anda menginginkan fungsi yang sama dengan yang digunakan pada klaster DB sebelumnya.

> **Important**
>
> Jika Anda menggunakan konsol untuk memulihkan klaster DB, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk memulihkan klaster DB, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat

dalam klaster DB. Jika Anda tidak membuat instans DB primer, titik akhir klaster DB tetap dalam status `creating`.
Panggil create-db-instance AWS CLI perintah untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai `--db-cluster-identifier` nilai pilihan.

API RDS

Untuk memulihkan cluster DB dari snapshot cluster DB, panggil Restore operasi RDS API DBCluster FromSnapshot dengan parameter berikut:

- `DBClusterIdentifier`
- `SnapshotIdentifier`

**Important**

Jika Anda menggunakan konsol untuk memulihkan klaster DB, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan API RDS untuk memulihkan klaster DB, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Jika Anda tidak membuat instans DB primer, titik akhir klaster DB tetap dalam status `creating`.
Panggil operasi RDS API Create DBInstance untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai nilai parameter `DBClusterIdentifier`.

# Penyalinan snapshot cluster DB

Dengan Amazon Aurora, Anda dapat menyalin cadangan otomatis atau snapshot klaster DB manual. Setelah Anda menyalin snapshot, salinannya disebut snapshot manual. Anda dapat membuat beberapa salinan cadangan otomatis atau snapshot manual, tetapi setiap salinan harus memiliki pengidentifikasi unik.

Anda dapat menyalin snapshot dalam hal yang sama Wilayah AWS, Anda dapat menyalin snapshot di seluruh Wilayah AWS, dan Anda dapat menyalin snapshot bersama.

Anda tidak dapat menyalin snapshot klaster DB antar-Wilayah dan akun dalam satu langkah. Lakukan satu langkah untuk setiap tindakan penyalinan ini. Sebagai alternatif untuk menyalin, Anda juga dapat berbagi snapshot manual dengan akun lain AWS . Untuk informasi selengkapnya, lihat Berbagi snapshot klaster DB.

> **Note**
>
> Amazon menagih Anda berdasarkan jumlah data snapshot dan cadangan Amazon Aurora yang Anda simpan dan jangka waktu penyimpanannya. Untuk informasi tentang penyimpanan yang terkait dengan cadangan dan snapshot Aurora, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. Untuk informasi harga terkait penyimpanan Aurora, lihat Harga Amazon RDS for Aurora.

Tinjau batasan dan pertimbangan untuk penyalinan snapshot cluster DB. Untuk menyalin snapshot cluster DB, lihat salah satu topik berikut.

- Menyalin snapshot cluster DB dengan AWS Management Console
- Menyalin snapshot cluster DB yang tidak terenkripsi dengan menggunakan AWS CLI atau Amazon RDS API
- Menyalin snapshot cluster DB terenkripsi dengan menggunakan AWS CLI atau Amazon RDS API
- Menyalin snapshot klaster DB di seluruh akun

## Menyalin snapshot cluster DB dengan AWS Management Console

Gunakan prosedur dalam topik ini untuk menyalin snapshot klaster DB. Jika mesin basis data sumber Anda adalah Aurora, maka snapshot Anda adalah snapshot klaster DB.

Untuk setiap AWS akun, Anda dapat menyalin hingga lima snapshot cluster DB sekaligus dari satu Wilayah AWS ke yang lain. Menyalin snapshot klaster DB terenkripsi maupun tidak terenkripsi didukung. Jika Anda menyalin snapshot cluster DB ke yang lain Wilayah AWS, Anda membuat snapshot cluster DB manual yang dipertahankan di dalamnya. Wilayah AWS Menyalin snapshot cluster DB dari sumber menimbulkan biaya transfer Wilayah AWS data Amazon RDS.

Untuk informasi selengkapnya tentang biaya transfer data, lihat Harga Amazon RDS.

Setelah salinan snapshot cluster DB dibuat di yang baru Wilayah AWS, salinan snapshot cluster DB berperilaku sama seperti semua snapshot cluster DB lainnya di dalamnya. Wilayah AWS

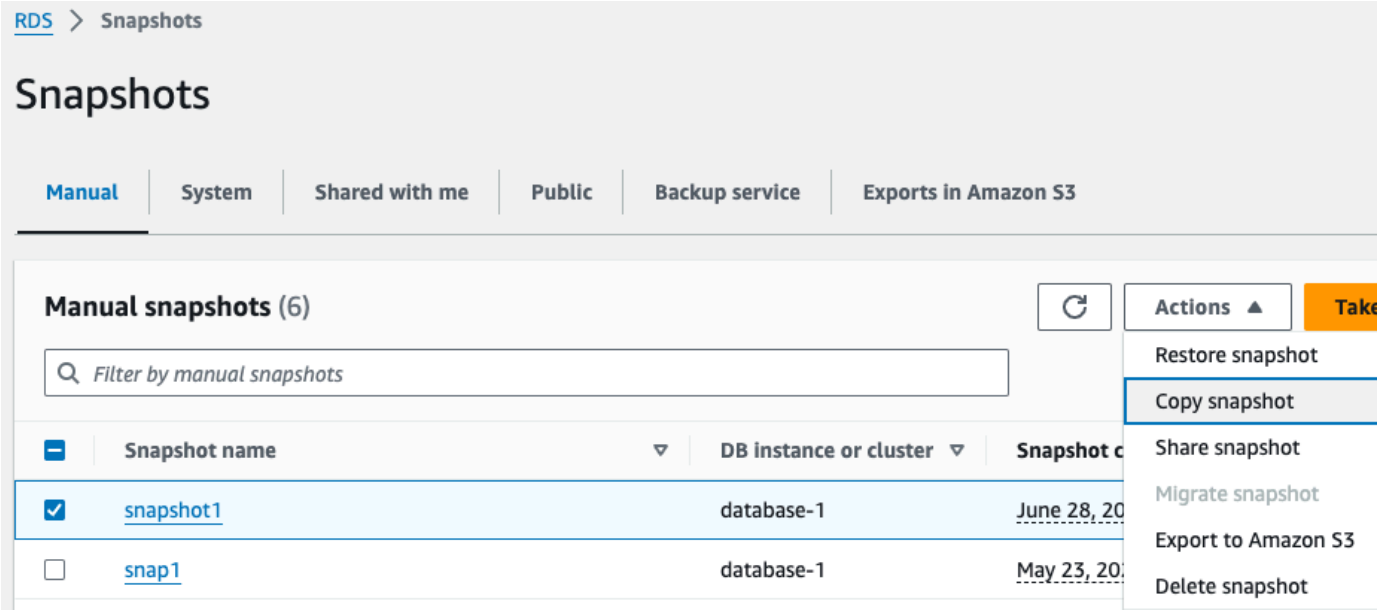
Prosedur ini berfungsi untuk menyalin snapshot klaster DB terenkripsi atau tidak terenkripsi, di Wilayah AWS yang sama atau lintas Wilayah.

Untuk membatalkan operasi penyalinan setelah berlangsung, hapus snapshot klaster DB target saat snapshot klaster DB tersebut dalam status menyalin.

Sebelum menyalin snapshot cluster DB, tinjau dan. Batasan Pertimbangan untuk penyalinan snapshot

Untuk menyalin snapshot klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot cluster DB yang ingin Anda salin.
4. Pilih Tindakan, lalu pilih Salin snapshot.




5. (Opsional) Untuk menyalin snapshot cluster DB ke yang lain Wilayah AWS, pilih itu Wilayah AWS untuk Wilayah Tujuan.
6. Masukkan nama salinan snapshot klaster DB dalam Pengidentifikasi Snapshot DB Baru.
7. Untuk menyalin tanda dan nilai dari snapshot ke salinan snapshot, pilih Salin Tanda.
8. Pilih Salin Snapshot.

## Batasan

Berikut adalah beberapa batasan saat Anda menyalin snapshot:

- Anda tidak dapat menyalin snapshot ke atau dari berikut ini: Wilayah AWS
  - China (Beijing)
  - China (Ningxia)
- Anda dapat menyalin snapshot antara AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat). Namun, Anda tidak dapat menyalin snapshot antara AWS GovCloud (US) Wilayah ini dan komersial Wilayah AWS.
- Jika Anda menghapus snapshot sumber sebelum snapshot target tersedia, penyalinan snapshot mungkin gagal. Verifikasi bahwa snapshot target telah berstatus `AVAILABLE` sebelum Anda menghapus snapshot sumber.
- Anda dapat memiliki maksimum lima permintaan penyalinan snapshot yang sedang berlangsung ke satu Wilayah tujuan per akun.

- Bila Anda meminta beberapa salinan snapshot untuk instans DB sumber yang sama, salinan snapshot tersebut akan berada dalam antrean internal. Salinan yang diminta nanti tidak akan dimulai hingga salinan snapshot sebelumnya selesai. Untuk informasi selengkapnya, lihat Mengapa pembuatan snapshot EC2 AMI atau EBS saya lambat? di pusat AWS pengetahuan.
- Bergantung pada yang Wilayah AWS terlibat dan jumlah data yang akan disalin, salinan snapshot lintas wilayah dapat memakan waktu berjam-jam untuk diselesaikan. Dalam beberapa kasus, mungkin ada sejumlah besar permintaan penyalinan snapshot lintas Wilayah dari Wilayah sumber tertentu. Dalam kasus seperti itu, Amazon RDS mungkin menempatkan permintaan penyalinan lintas Wilayah baru dari Wilayah sumber tersebut ke dalam antrean hingga beberapa penyalinan yang sedang berlangsung selesai. Tidak ada informasi progres yang ditampilkan tentang permintaan penyalinan saat permintaan tersebut ada di dalam antrean. Informasi progres akan ditampilkan saat penyalinan dimulai.
- Aurora tidak mendukung snapshot tambahan. Salinan snapshot cluster Aurora DB selalu disimpan sebagai salinan lengkap. Salinan snapshot lengkap berisi semua data dan metadata yang diperlukan untuk memulihkan klaster DB.

## Pertimbangan untuk penyalinan snapshot

Berikut ini adalah pertimbangan saat menyalin snapshot.

Topik


- Pertimbangan untuk penyalinan snapshot bersama
- Pertimbangan untuk penyalinan snapshot cluster DB terenkripsi
- Pertimbangan untuk penyalinan snapshot Lintas Wilayah
- Pertimbangan untuk kelompok parameter


### Pertimbangan untuk penyalinan snapshot bersama

Anda dapat menyalin snapshot yang dibagikan kepada Anda oleh AWS akun lain. Dalam beberapa kasus, Anda mungkin menyalin snapshot terenkripsi yang telah dibagikan dari akun lain. AWS Dalam kasus ini, Anda harus memiliki akses ke AWS KMS key yang digunakan untuk mengenkripsi snapshot.

Anda hanya dapat menyalin snapshot klaster DB bersama, baik yang dienkripsi atau tidak, di Wilayah AWS yang sama. Untuk informasi selengkapnya, lihat Berbagi snapshot terenkripsi.

## Pertimbangan untuk penyalinan snapshot cluster DB terenkripsi

Anda dapat menyalin snapshot yang telah dienkripsi menggunakan kunci KMS. Jika Anda menyalin snapshot terenkripsi, salinan snapshot tersebut juga harus dienkripsi. Jika Anda menyalin snapshot terenkripsi dalam yang sama Wilayah AWS, Anda dapat mengenkripsi salinan dengan kunci KMS yang sama dengan snapshot asli. Atau Anda dapat menentukan kunci KMS yang berbeda.

Jika Anda menyalin snapshot terenkripsi antar-Wilayah, Anda harus menentukan kunci KMS yang valid di Wilayah AWS tujuan. Kunci ini dapat berupa kunci KMS khusus Wilayah, atau kunci multi-Wilayah. Untuk informasi selengkapnya tentang kunci KMS multi-Wilayah, lihat Menggunakan kunci multi-Wilayah di AWS KMS.

Untuk informasi selengkapnya tentang manajemen AWS KMS kunci untuk Amazon RDS, lihatAWS KMS key manajemen.

Snapshot sumber tetap terenkripsi selama proses penyalinan. Untuk informasi selengkapnya, lihat Batasan klaster DB terenkripsi Amazon Aurora.

> **Note**
>
> Untuk snapshot klaster DB Amazon Aurora, Anda tidak dapat mengenkripsi snapshot klaster DB yang tidak terenkripsi saat Anda menyalin snapshot.

Untuk menyalin snapshot cluster DB terenkripsi, lihat topik berikut.

- Menyalin snapshot cluster DB terenkripsi dengan menggunakan AWS CLI atau Amazon RDS API
- Menyalin snapshot klaster DB di seluruh akun

## Pertimbangan untuk penyalinan snapshot Lintas Wilayah

Anda dapat menyalin snapshot klaster DB di seluruh Wilayah AWS. Namun, ada batasan dan pertimbangan tertentu dalam penyalinan snapshot lintas Wilayah.

Bergantung pada yang Wilayah AWS terlibat dan jumlah data yang akan disalin, salinan snapshot lintas wilayah dapat memakan waktu berjam-jam untuk diselesaikan.

Dalam beberapa kasus, mungkin ada sejumlah besar permintaan penyalinan snapshot lintas Wilayah dari Wilayah AWS sumber tertentu. Dalam kasus seperti itu, Amazon RDS mungkin menempatkan

permintaan salinan Lintas wilayah baru dari sumber tersebut Wilayah AWS ke dalam antrian hingga beberapa salinan yang sedang berlangsung selesai. Tidak ada informasi progres yang ditampilkan tentang permintaan penyalinan saat permintaan tersebut ada di dalam antrean. Informasi progres akan ditampilkan saat penyalinan dimulai.

Penyalinan snapshot Lintas Wilayah membuat salinan lengkap dalam data target, tetapi biaya transfer data bersifat bertahap. Data tambahan mencakup data baru yang telah ditambahkan ke database pelanggan sejak salinan terakhir, serta setiap perubahan yang dilakukan pada data yang ada. Untuk informasi selengkapnya, lihat Membuat salinan cadangan Wilayah AWS di Panduan AWS Backup Pengembang.

> **Note**
>
> Aurora menyalin jumlah minimum data yang diperlukan untuk membuat salinan lengkap snapshot di wilayah tujuan. Biaya transfer data berlaku saat menyalin snapshot antar wilayah.

### Pertimbangan untuk kelompok parameter

Saat Anda menyalin snapshot lintas Wilayah, salinannya tidak menyertakan grup parameter yang digunakan oleh klaster DB asli. Saat Anda memulihkan snapshot untuk membuat cluster DB baru, cluster DB itu mendapatkan grup parameter default untuk Wilayah AWS itu dibuat. Untuk memberi klaster DB baru parameter yang sama seperti aslinya, Anda harus melakukan hal berikut:

1. Di tujuan Wilayah AWS, buat grup parameter cluster DB dengan pengaturan yang sama dengan cluster DB asli. Jika sudah ada di yang baru Wilayah AWS, Anda dapat menggunakan yang itu.
2. Setelah Anda mengembalikan snapshot di tujuan Wilayah AWS, ubah cluster DB baru dan tambahkan grup parameter baru atau yang sudah ada dari langkah sebelumnya.

## Menyalin snapshot cluster DB yang tidak terenkripsi dengan menggunakan AWS CLI atau Amazon RDS API

Gunakan prosedur di bagian berikut untuk menyalin snapshot cluster DB yang tidak terenkripsi dengan menggunakan AWS Management Console, AWS CLI, atau Amazon RDSAPI.

Untuk membatalkan operasi penyalinan setelah dijalankan, hapus snapshot klaster DB target yang diidentifikasi oleh `--target-db-cluster-snapshot-identifier` atau `TargetDBClusterSnapshotIdentifier` saat snapshot klaster DB tersebut berstatus menyalin.

Konsol

Untuk menyalin snapshot cluster DB menggunakan AWS Management Console, lihat Menyalin snapshot cluster DB dengan AWS Management Console.

AWS CLI

Untuk menyalin snapshot cluster DB, gunakan AWS CLI copy-db-cluster-snapshotperintah. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, jalankan perintah di Wilayah AWS ke mana snapshot akan disalin.

Opsi berikut digunakan untuk menyalin snapshot klaster DB yang tidak terenkripsi:

- `--source-db-cluster-snapshot-identifier` – Pengidentifikasi untuk snapshot klaster DB yang akan disalin. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, pengenal ini harus dalam ARN format untuk sumbernya Wilayah AWS.
- `--target-db-cluster-snapshot-identifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB.

Kode berikut membuat salinan snapshot cluster DB `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` bernama `myclustersnapshotcopy` dalam Wilayah AWS di mana perintah dijalankan. Setelah salinan dibuat, semua tanda di snapshot asli akan disalin ke salinan snapshot.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

Untuk Windows:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 ^
```

```
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

RDS API

Untuk menyalin snapshot cluster DB, gunakan operasi opyDBClusterSnapshot Amazon RDS API C. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, melakukan tindakan di Wilayah AWS ke mana snapshot akan disalin.

Parameter berikut digunakan untuk menyalin snapshot klaster DB yang tidak terenkripsi:

- `SourceDBClusterSnapshotIdentifier` – Pengidentifikasi untuk snapshot klaster DB yang akan disalin. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, pengenal ini harus dalam ARN format untuk sumbernya Wilayah AWS.
- `TargetDBClusterSnapshotIdentifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB.

Kode berikut membuat salinan snapshot `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` bernama `myclustersnapshotcopy` di Wilayah AS Barat (California Utara). Setelah salinan dibuat, semua tanda di snapshot asli akan disalin ke salinan snapshot.

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-
snapshot%3Aaurora-cluster1-snapshot-20130805
   &TargetDBSnapshotIdentifier=myclustersnapshotcopy
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
   &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

# Menyalin snapshot cluster DB terenkripsi dengan menggunakan AWS CLI atau Amazon RDS API

Gunakan prosedur di bagian berikut untuk menyalin snapshot cluster DB terenkripsi dengan menggunakan AWS Management Console, AWS CLI, atau Amazon RDSAPI.

Untuk membatalkan operasi penyalinan setelah dijalankan, hapus snapshot klaster DB target yang diidentifikasi oleh `--target-db-cluster-snapshot-identifier` atau `TargetDBClusterSnapshotIdentifier` saat snapshot klaster DB tersebut berstatus menyalin.

Konsol

Untuk menyalin snapshot cluster DB menggunakan AWS Management Console, lihat Menyalin snapshot cluster DB dengan AWS Management Console.

AWS CLI

Untuk menyalin snapshot cluster DB, gunakan AWS CLI copy-db-cluster-snapshotperintah. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, jalankan perintah di Wilayah AWS ke mana snapshot akan disalin.

Opsi berikut digunakan untuk menyalin snapshot klaster DB terenkripsi:

- `--source-db-cluster-snapshot-identifier` – Pengidentifikasi untuk snapshot klaster DB terenkripsi yang akan disalin. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, pengenal ini harus dalam ARN format untuk sumbernya Wilayah AWS.
- `--target-db-cluster-snapshot-identifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB terenkripsi.
- `--kms-key-id`— Pengidentifikasi KMS kunci untuk kunci yang akan digunakan untuk mengenkripsi salinan snapshot cluster DB.

  Anda dapat menggunakan opsi ini secara opsional jika snapshot cluster DB dienkripsi, Anda menyalin snapshot dalam hal yang sama Wilayah AWS, dan Anda ingin menentukan KMS kunci baru untuk mengenkripsi salinan. Jika tidak, salinan snapshot cluster DB dienkripsi dengan KMS kunci yang sama dengan snapshot cluster DB sumber.

  Anda harus menggunakan opsi ini jika snapshot cluster DB dienkripsi dan Anda menyalin snapshot ke yang lain Wilayah AWS. Dalam hal ini, Anda harus menentukan KMS kunci untuk tujuan Wilayah AWS.

Contoh kode berikut ini menyalin snapshot klaster DB terenkripsi dari Wilayah US West (Oregon) ke Wilayah US East (N. Virginia) . Perintah ini dipanggil di Wilayah AS Timur (Virginia Utara).

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --kms-key-id my-us-east-1-key
```

Untuk Windows:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

`--source-region`Parameter diperlukan saat Anda menyalin snapshot cluster DB terenkripsi di antara AWS GovCloud (AS-Timur) dan AWS GovCloud Wilayah (AS-Barat). Untuk`--source-region`, tentukan Wilayah AWS dari instans DB sumber. Bagian Wilayah AWS ditentukan dalam `source-db-cluster-snapshot-identifier` harus cocok dengan Wilayah AWS ditentukan untuk`--source-region`.

Jika `--source-region` tidak ditentukan, tentukan nilai `--pre-signed-url`. Presigned URL adalah URL yang berisi permintaan ditandatangani Signature Version 4 untuk `copy-db-cluster-snapshot` perintah yang dipanggil di sumber Wilayah AWS. Untuk mempelajari lebih lanjut tentang `pre-signed-url` opsi ini, lihat copy-db-cluster-snapshotdi AWS CLI Referensi Perintah.

RDS API

Untuk menyalin snapshot cluster DB, gunakan operasi opyDBClusterSnapshot Amazon RDS API C. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, melakukan tindakan di Wilayah AWS ke mana snapshot akan disalin.

Parameter berikut digunakan untuk menyalin snapshot klaster DB terenkripsi:

- `SourceDBClusterSnapshotIdentifier` – Pengidentifikasi untuk snapshot klaster DB terenkripsi yang akan disalin. Jika Anda menyalin snapshot ke yang lain Wilayah AWS, pengenal ini harus dalam ARN format untuk sumbernya Wilayah AWS.
- `TargetDBClusterSnapshotIdentifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB terenkripsi.
- `KmsKeyId`— Pengidentifikasi KMS kunci untuk kunci yang akan digunakan untuk mengenkripsi salinan snapshot cluster DB.

  Anda dapat menggunakan parameter ini secara opsional jika snapshot cluster DB dienkripsi, Anda menyalin snapshot dalam hal yang sama Wilayah AWS, dan Anda menentukan KMS kunci baru untuk digunakan untuk mengenkripsi salinan. Jika tidak, salinan snapshot cluster DB dienkripsi dengan KMS kunci yang sama dengan snapshot cluster DB sumber.

  Anda harus menggunakan parameter ini jika snapshot cluster DB dienkripsi dan Anda menyalin snapshot ke yang lain Wilayah AWS. Dalam hal ini, Anda harus menentukan KMS kunci untuk tujuan Wilayah AWS.
- `PreSignedUrl`— Jika Anda menyalin snapshot ke yang lain Wilayah AWS, Anda harus menentukan `PreSignedUrl` parameternya. `PreSignedUrl`Nilai harus berupa permintaan URL yang ditandatangani Signature Version 4 untuk `CopyDBClusterSnapshot` tindakan yang akan dipanggil di sumber Wilayah AWS dari mana snapshot cluster DB disalin. Untuk mempelajari selengkapnya tentang menggunakan presignedURL, lihat C opyDBCluster Snapshot.

Contoh kode berikut menyalin snapshot klaster DB terenkripsi dari Wilayah AS Barat (Oregon) ke Wilayah AS Timur (Virginia Utara). Tindakan ini dipanggil di Wilayah AS Timur (Virginia Utara).

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
```

```
        %2526Version%253D2014-10-31
        %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
        %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
        %2526X-Amz-Date%253D20161117T215409Z
        %2526X-Amz-Expires%253D3600
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

`PreSignedUrl`Parameter ini diperlukan saat Anda menyalin snapshot cluster DB terenkripsi antara AWS GovCloud (AS-Timur) dan AWS GovCloud Wilayah (AS-Barat). `PreSignedUrl`Nilai harus berupa permintaan URL yang ditandatangani Signature Version 4 untuk `CopyDBClusterSnapshot` operasi yang akan dipanggil di sumber Wilayah AWS dari mana snapshot cluster DB disalin. Untuk mempelajari selengkapnya tentang menggunakan presignedURL, lihat C opyDBCluster Snapshot di Referensi Amazon RDS API.

Untuk secara otomatis daripada menghasilkan presigned secara manualURL, gunakan AWS CLI copy-db-cluster-snapshotperintah dengan `--source-region` opsi sebagai gantinya.

## Menyalin snapshot klaster DB di seluruh akun

Anda dapat mengaktifkan lainnya AWS akun untuk menyalin snapshot cluster DB yang Anda tentukan dengan menggunakan Amazon RDS API `ModifyDBClusterSnapshotAttribute` dan `CopyDBClusterSnapshot` tindakan. Anda hanya dapat menyalin snapshot cluster DB di seluruh akun dalam hal yang sama Wilayah AWS. Proses penyalinan lintas akun berfungsi sebagai berikut, di mana Akun A membuat snapshot tersedia untuk disalin, dan Akun B menyalinnya.

1. Dengan menggunakan Akun A, panggil `ModifyDBClusterSnapshotAttribute`, dengan menentukan **`restore`** untuk parameter `AttributeName`, dan ID untuk Akun B untuk parameter `ValuesToAdd`.
2. (Jika snapshot dienkripsi) Menggunakan Akun A, perbarui kebijakan kunci untuk KMS kunci tersebut, pertama-tama tambahkan Akun B sebagai a`Principal`, lalu izinkan tindakan. ARN `kms:CreateGrant`
3. (Jika snapshot dienkripsi) Menggunakan Akun B, pilih atau buat pengguna dan lampirkan IAM kebijakan ke pengguna tersebut yang memungkinkannya menyalin snapshot cluster DB terenkripsi menggunakan kunci Anda. KMS
4. Menggunakan Akun B, panggil `CopyDBClusterSnapshot` dan gunakan `SourceDBClusterSnapshotIdentifier` parameter untuk menentukan snapshot cluster DB yang akan disalin, yang harus menyertakan ID untuk Akun A. ARN

Untuk daftar semua AWS akun diizinkan untuk memulihkan snapshot cluster DB, gunakan escribeDBCluster SnapshotAttributes API operasi escribeDBSnapshotAtribut D atau D.

Untuk menghapus izin berbagi untuk AWS akun, gunakan `ModifyDBSnapshotAttribute` atau `ModifyDBClusterSnapshotAttribute` tindakan dengan `AttributeName` set ke `restore` dan ID akun untuk menghapus dalam `ValuesToRemove` parameter.

Menyalin snapshot klaster DB tidak terenkripsi ke akun lain

Gunakan prosedur berikut untuk menyalin snapshot cluster DB yang tidak terenkripsi ke akun lain di akun yang sama Wilayah AWS.

1. Di akun sumber untuk snapshot klaster DB, panggil `ModifyDBClusterSnapshotAttribute`, dengan menentukan **`restore`** untuk parameter `AttributeName`, dan ID akun target untuk parameter `ValuesToAdd`.

   Menjalankan contoh berikut menggunakan akun `987654321` mengizinkan dua AWS pengidentifikasi akun, `123451234512` dan`123456789012`, untuk mengembalikan snapshot cluster DB bernama. `manual-snapshot1`

   ```
   https://rds.us-west-2.amazonaws.com/
    ?Action=ModifyDBClusterSnapshotAttribute
    &AttributeName=restore
    &DBClusterSnapshotIdentifier=manual-snapshot1
    &SignatureMethod=HmacSHA256&SignatureVersion=4
   ```

```
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. Di akun target, panggil `CopyDBClusterSnapshot` dan gunakan `SourceDBClusterSnapshotIdentifier` parameter untuk menentukan snapshot cluster DB yang akan disalin, yang harus menyertakan ID untuk akun sumber. ARN

   Menjalankan contoh berikut menggunakan akun `123451234512` menyalin snapshot klaster DB `aurora-cluster1-snapshot-20130805` dari akun `987654321` dan membuat snapshot klaster DB yang diberi nama `dbclustersnapshot1`.

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

### Menyalin snapshot klaster DB terenkripsi ke akun lain

Gunakan prosedur berikut untuk menyalin snapshot cluster DB terenkripsi ke akun lain di akun yang sama Wilayah AWS.

1. Di akun sumber untuk snapshot klaster DB, panggil `ModifyDBClusterSnapshotAttribute`, dengan menentukan **`restore`** untuk parameter `AttributeName`, dan ID akun target untuk parameter `ValuesToAdd`.

   Menjalankan contoh berikut menggunakan akun `987654321` mengizinkan dua AWS pengidentifikasi akun, `123451234512` dan`123456789012`, untuk mengembalikan snapshot cluster DB bernama. `manual-snapshot1`

   ```
   https://rds.us-west-2.amazonaws.com/
    ?Action=ModifyDBClusterSnapshotAttribute
    &AttributeName=restore
    &DBClusterSnapshotIdentifier=manual-snapshot1
    &SignatureMethod=HmacSHA256&SignatureVersion=4
    &ValuesToAdd.member.1=123451234512
    &ValuesToAdd.member.2=123456789012
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
    &X-Amz-Date=20150922T220515Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
   ```

2. Di akun sumber untuk snapshot cluster DB, buat KMS kunci khusus yang sama Wilayah AWS sebagai snapshot cluster DB terenkripsi. Saat membuat kunci yang dikelola pelanggan, Anda memberikan akses ke sana untuk target Akun AWS Untuk informasi selengkapnya, lihat Buat kunci yang dikelola pelanggan dan berikan akses ke sana.

3. Salin dan bagikan snapshot ke target Akun AWS Untuk informasi selengkapnya, lihat Salin dan bagikan snapshot dari akun sumber.

4. Di akun target, panggil `CopyDBClusterSnapshot` dan gunakan `SourceDBClusterSnapshotIdentifier` parameter untuk menentukan snapshot cluster DB yang akan disalin, yang harus menyertakan ID untuk akun sumber. ARN

   Menjalankan contoh berikut menggunakan akun `123451234512` akan menyalin snapshot klaster DB `aurora-cluster1-snapshot-20130805` dari akun `987654321` dan membuat snapshot klaster DB bernama `dbclustersnapshot1`.

   ```
   https://rds.us-west-2.amazonaws.com/
      ?Action=CopyDBClusterSnapshot
      &CopyTags=true
      &SignatureMethod=HmacSHA256
   ```

```
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

# Berbagi snapshot klaster DB

Menggunakan AmazonRDS, Anda dapat membagikan snapshot cluster DB manual dengan cara berikut:

- Berbagi snapshot klaster DB manual, baik yang dienkripsi maupun tidak, sehingga memungkinkan akun AWS terotorisasi menyalin snapshot tersebut.
- Dengan berbagi snapshot klaster DB manual, baik yang dienkripsi maupun tidak, akun AWS terotorisasi akan dapat memulihkan klaster DB secara langsung dari snapshot ini, bukan membuat salinannya dan memulihkan dari salinan tersebut.

> **Note**
>
> Untuk berbagi snapshot klaster DB otomatis, buat klaster DB manual dengan menyalin snapshot otomatis, lalu membagikannya. Proses ini juga berlaku untuk sumber daya yang dihasilkan oleh AWS Backup.

Untuk informasi selengkapnya tentang cara menyalin snapshot, lihat Penyalinan snapshot cluster DB. Untuk informasi selengkapnya tentang memulihkan instans DB dari snapshot klaster DB, lihat Memulihkan dari snapshot klaster DB.

Untuk informasi selengkapnya tentang memulihkan klaster DB dari snapshot klaster DB, lihat Gambaran umum pencadangan dan pemulihan klaster DB Aurora.

Anda dapat berbagi snapshot manual dengan hingga 20 lainnya Akun AWS.

Batasan berikut berlaku saat berbagi foto manual dengan yang lain Akun AWS:

- Saat memulihkan kluster DB dari snapshot bersama menggunakan AWS Command Line Interface (AWS CLI) atau Amazon RDSAPI, Anda harus menentukan Amazon Resource Name (ARN) dari snapshot bersama sebagai pengenal snapshot.

Pelajari cara berbagi snapshot, snapshot publik, dan snapshot terenkripsi di bagian berikut. Anda juga dapat mempelajari cara berhenti berbagi snapshot.

Topik

- Berbagi snapshot

- Berbagi snapshot publik
- Berbagi snapshot terenkripsi
- Menghentikan berbagi snapshot

## Berbagi snapshot

Anda dapat membagikan snapshot cluster DB menggunakan AWS Management Console, file AWS CLI, atau file. RDS API

Konsol

Dengan menggunakan RDS konsol Amazon, Anda dapat membagikan snapshot cluster DB manual hingga 20 Akun AWS. Anda juga dapat menggunakan konsol untuk berhenti berbagi snapshot manual dengan satu atau beberapa akun.

Untuk membagikan snapshot cluster DB manual dengan menggunakan konsol Amazon RDS

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot manual yang ingin Anda bagikan.
4. Untuk Tindakan, pilih Bagikan snapshot.
5. Pilih salah satu opsi berikut untuk Visibilitas snapshot DB.

    - Jika sumbernya tidak terenkripsi, pilih Publik untuk mengizinkan semua Akun AWS memulihkan cluster DB dari snapshot cluster DB manual Anda, atau pilih Private untuk mengizinkan hanya Akun AWS yang Anda tentukan untuk memulihkan cluster DB dari snapshot cluster DB manual Anda.

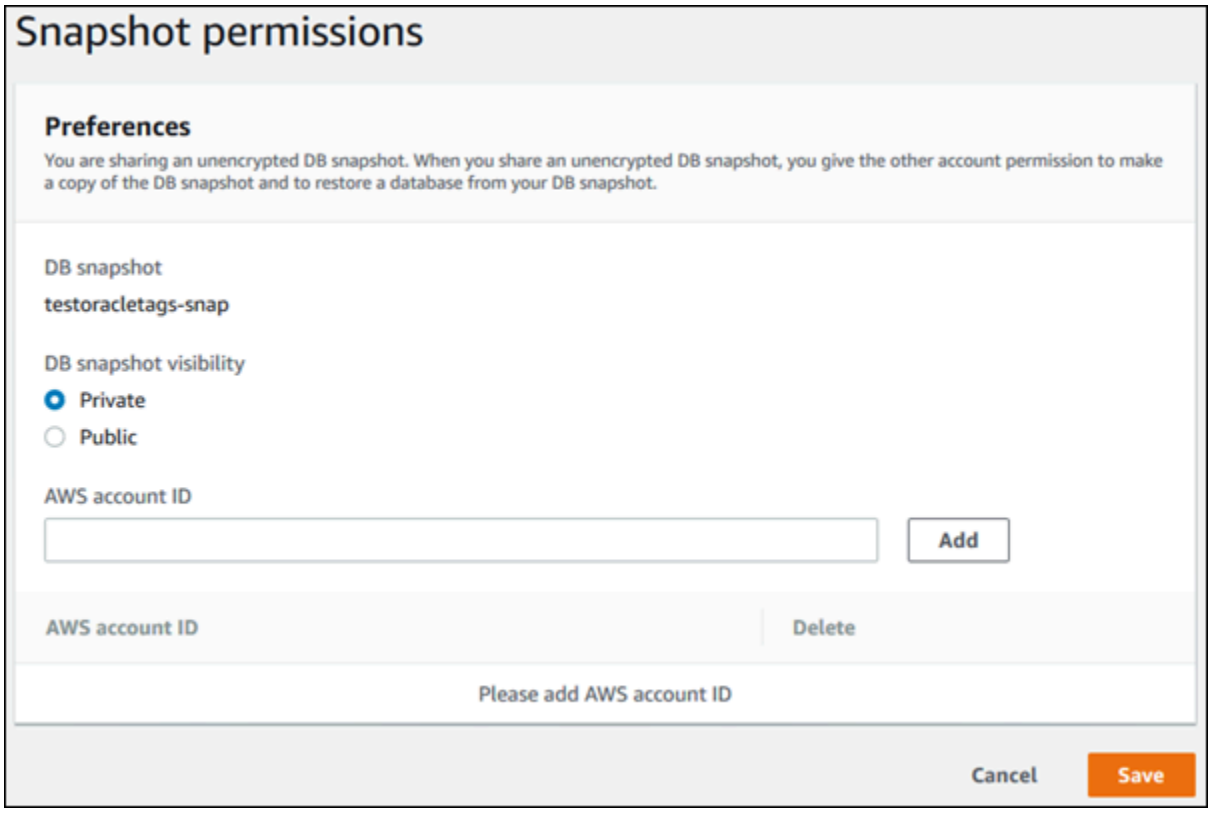
    > **Warning**
    >
    > Jika Anda menyetel visibilitas snapshot DB ke Publik, semua Akun AWS dapat memulihkan cluster DB dari snapshot cluster DB manual Anda dan memiliki akses ke data Anda. Jangan berbagi snapshot klaster DB manual apa pun yang berisi informasi pribadi sebagai Publik.
    > Untuk informasi selengkapnya, lihat Berbagi snapshot publik.

- Jika sumbernya dienkripsi, Visibilitas snapshot DB ditetapkan sebagai Privat karena snapshot terenkripsi tidak dapat dibagikan sebagai publik.

> **Note**
>
> Snapshot yang telah dienkripsi dengan default tidak AWS KMS key dapat dibagikan. Untuk informasi tentang cara mengatasi masalah ini, lihatBerbagi snapshot terenkripsi.

6. Untuk ID AWS Akun, masukkan Akun AWS pengenal untuk akun yang ingin Anda izinkan untuk memulihkan kluster DB dari snapshot manual Anda, lalu pilih Tambah. Ulangi untuk menyertakan Akun AWS pengidentifikasi tambahan, hingga 20 Akun AWS.

   Jika Anda membuat kesalahan saat menambahkan Akun AWS pengenal ke daftar akun yang diizinkan, Anda dapat menghapusnya dari daftar dengan memilih Hapus di sebelah kanan Akun AWS pengidentifikasi yang salah.




7. Setelah Anda menambahkan pengenal untuk semua Akun AWS yang ingin Anda izinkan untuk memulihkan snapshot manual, pilih Simpan untuk menyimpan perubahan Anda.

AWS CLI

Untuk berbagi snapshot klaster DB, gunakan perintah `aws rds modify-db-cluster-snapshot-attribute`. Gunakan `--values-to-add` parameter untuk menambahkan daftar IDs untuk Akun AWS yang berwenang untuk mengembalikan snapshot manual.

Example berbagi snapshot dengan satu akun

Contoh berikut memungkinkan Akun AWS identifier `123456789012` untuk mengembalikan snapshot cluster DB bernama. `cluster-3-snapshot`

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

Untuk Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example berbagi snapshot dengan beberapa akun

Contoh berikut memungkinkan dua Akun AWS pengidentifikasi, `111122223333` dan`444455556666`, untuk mengembalikan snapshot cluster DB bernama. `manual-cluster-snapshot1`

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

Untuk Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
```

```
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> Saat menggunakan prompt perintah Windows, Anda harus menghindari tanda kutip ganda (“) dalam JSON kode dengan mengawali mereka dengan garis miring terbalik (\).

Untuk membuat daftar yang Akun AWS diaktifkan untuk memulihkan snapshot, gunakan `describe-db-cluster-snapshot-attributes` AWS CLI perintah.

RDS API

Anda juga dapat berbagi snapshot cluster DB manual dengan yang lain Akun AWS dengan menggunakan Amazon RDSAPI. Untuk melakukannya, panggil operasi `ModifyDBClusterSnapshotAttribute`. Tentukan `restore` untuk`AttributeName`, dan gunakan `ValuesToAdd` parameter untuk menambahkan daftar IDs untuk Akun AWS yang berwenang untuk mengembalikan snapshot manual.

Untuk membuat snapshot manual publik dan dapat dipulihkan oleh semua orang Akun AWS, gunakan nilainya. `all` Namun, berhati-hatilah untuk tidak menambahkan `all` nilai untuk setiap snapshot manual yang berisi informasi pribadi yang Anda tidak ingin tersedia untuk semua Akun AWS. Selain itu, jangan tentukan `all` untuk snapshot terenkripsi karena menjadikan snapshot tersebut berstatus publik tidak didukung.

Untuk mencantumkan semua yang Akun AWS diizinkan untuk memulihkan snapshot, gunakan `DescribeDBClusterSnapshotAttributes`APIoperasi.

## Berbagi snapshot publik

Anda dapat membagikan snapshot manual yang tidak terenkripsi sebagai publik, yang membuat snapshot tersedia untuk semua. Akun AWS Pastikan saat berbagi snapshot sebagai publik bahwa tidak ada informasi pribadi yang dimasukkan ke dalam snapshot publik.

Ketika snapshot dibagikan secara publik, ia memberikan semua Akun AWS izin untuk menyalin snapshot dan membuat cluster DB darinya.

Anda tidak ditagih untuk penyimpanan cadangan snapshot publik yang dimiliki oleh akun lain. Anda hanya ditagih untuk snapshot yang Anda miliki.

Jika Anda menyalin snapshot publik, Anda memiliki salinannya. Anda ditagih untuk penyimpanan cadangan salinan snapshot Anda. Jika Anda membuat klaster DB dari snapshot publik, Anda ditagih untuk klaster DB tersebut. Untuk informasi harga Amazon Aurora, lihat halaman harga Aurora.

Anda hanya dapat menghapus snapshot publik yang Anda miliki. Untuk menghapus snapshot bersama atau publik, pastikan untuk masuk ke snapshot Akun AWS yang memiliki snapshot tersebut.

## Melihat snapshot publik yang dimiliki oleh orang lain Akun AWS

Anda dapat melihat snapshot publik yang dimiliki oleh akun lain secara khusus Wilayah AWS di tab Publik halaman Snapshots di konsol AmazonRDS. Snapshot Anda (yang dimiliki oleh akun Anda) tidak muncul di tab ini.

Untuk melihat snapshot publik

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih tab Publik.

   Snapshot publik muncul. Anda dapat melihat akun mana yang memiliki snapshot publik di kolom Pemilik.

   > **Note**
   >
   > Anda mungkin harus mengubah preferensi halaman, dengan memilih ikon roda gigi di kanan atas daftar Snapshot publik, untuk melihat kolom ini.

## Melihat snapshot publik Anda sendiri

Anda dapat menggunakan AWS CLI perintah berikut (hanya Unix) untuk melihat snapshot publik yang dimiliki oleh Anda Akun AWS secara khusus. Wilayah AWS

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

Output yang ditampilkan mirip dengan contoh berikut jika Anda memiliki snapshot publik:

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
```

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

### Berbagi snapshot publik dari versi mesin DB yang tidak digunakan lagi

Memulihkan atau menyalin snapshot publik dari versi mesin DB yang tidak digunakan lagi tidak didukung. Untuk membuat snapshot publik yang tidak didukung yang ada tersedia untuk dipulihkan atau disalin, lakukan langkah-langkah berikut:

1. Tandai snapshot sebagai pribadi.
2. Pulihkan snapshot yang telah disalin.
3. Tingkatkan cluster DB yang dipulihkan ke versi mesin yang didukung.
4. Buat snapshot baru.
5. Bagikan kembali snapshot secara publik.

## Berbagi snapshot terenkripsi

Anda dapat membagikan snapshot cluster DB yang telah dienkripsi “saat istirahat” menggunakan algoritma enkripsi AES -256, seperti yang dijelaskan dalam. Mengenkripsi sumber daya Amazon Aurora

Pembatasan berikut berlaku untuk berbagi snapshot terenkripsi:

- Anda tidak dapat berbagi snapshot terenkripsi sebagai publik.
- Anda tidak dapat membagikan snapshot yang telah dienkripsi menggunakan kunci default KMS Akun AWS yang membagikan snapshot.

  Untuk informasi lebih lanjut tentang AWS KMS manajemen kunci untuk AmazonRDS, lihatAWS KMS key manajemen.

Untuk mengatasi masalah KMS kunci default, lakukan tugas-tugas berikut:

1. Buat kunci yang dikelola pelanggan dan berikan akses ke sana.
2. Salin dan bagikan snapshot dari akun sumber.
3. Salin snapshot bersama di akun target.

## Buat kunci yang dikelola pelanggan dan berikan akses ke sana

Pertama, Anda membuat KMS kunci khusus dalam hal yang sama Wilayah AWS sebagai snapshot cluster DB terenkripsi. Saat membuat kunci yang dikelola pelanggan, Anda memberikan akses ke sana untuk yang lain Akun AWS.

Untuk membuat kunci yang dikelola pelanggan dan memberikan akses ke sana

1. Masuk ke AWS Management Console dari sumbernya Akun AWS.
2. Buka AWS KMS konsol di https://console.aws.amazon.com/kms.
3. Untuk mengubah Wilayah AWS, gunakan pemilih Wilayah di sudut kanan atas halaman.
4. Di panel navigasi, pilih Kunci yang dikelola pelanggan.
5. Pilih Buat kunci.
6. Pada halaman tombol Konfigurasi:
    a. Untuk Key type, pilih Symmetric.
    b. Untuk penggunaan Kunci, pilih Enkripsi dan dekripsi.
    c. Perluas Opsi lanjutan.
    d. Untuk asal bahan utama, pilih KMS.
    e. Untuk Regionalitas, pilih kunci Wilayah Tunggal.
    f. Pilih Berikutnya.
7. Pada halaman Tambahkan label:
    a. Untuk Alias. masukkan nama tampilan untuk KMS kunci Anda, misalnya**`share-snapshot`**.
    b. (Opsional) Masukkan deskripsi untuk KMS kunci Anda.
    c. (Opsional) Tambahkan tag ke KMS kunci Anda.
    d. Pilih Berikutnya.
8. Pada halaman Tentukan izin administratif kunci, pilih Berikutnya.
9. Pada halaman Tentukan izin penggunaan kunci:
    a. Untuk  Lainnya Akun AWS, pilih  Tambahkan yang lain Akun AWS.
    b. Masukkan ID dari Akun AWS yang ingin Anda berikan akses.

        Anda dapat memberikan akses ke beberapa Akun AWS.
    c. Pilih Berikutnya.

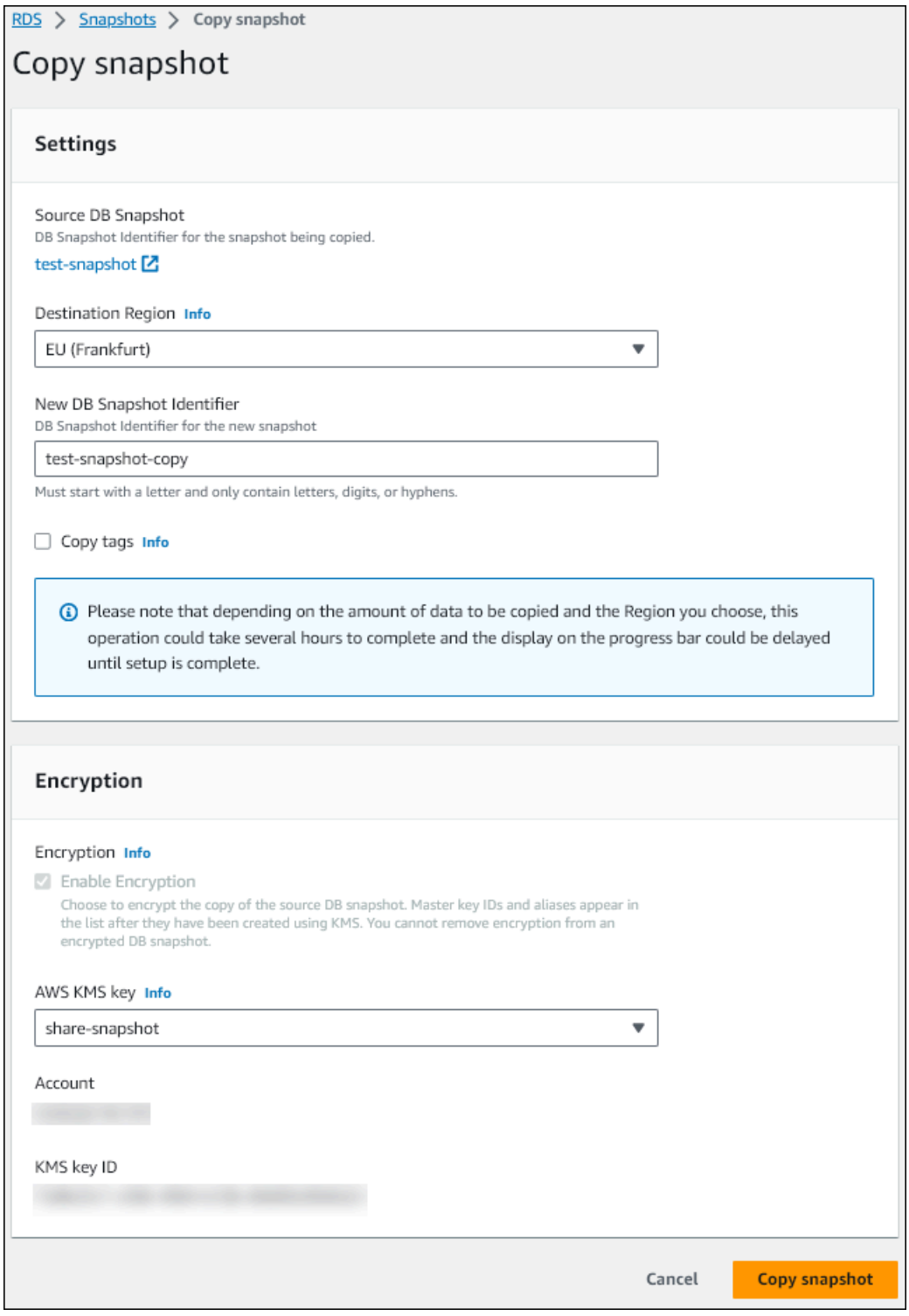
10. Tinjau KMS kunci Anda, lalu pilih Selesai.

## Salin dan bagikan snapshot dari akun sumber

Selanjutnya Anda menyalin snapshot cluster DB sumber ke snapshot baru menggunakan kunci yang dikelola pelanggan. Kemudian Anda membagikannya dengan target Akun AWS.

Untuk menyalin dan membagikan snapshot

1. Masuk ke AWS Management Console dari sumbernya Akun AWS.
2. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/
3. Di panel navigasi, pilih Snapshot.
4. Pilih snapshot cluster DB yang ingin Anda salin.
5. Untuk Tindakan, pilih Salin snapshot.
6. Pada halaman Salin snapshot:
    a. Untuk Wilayah Tujuan, pilih Wilayah AWS di mana Anda membuat kunci yang dikelola pelanggan dalam prosedur sebelumnya.
    b. Masukkan nama salinan snapshot klaster DB dalam Pengidentifikasi Snapshot DB Baru.
    c. Untuk AWS KMS key, pilih kunci yang dikelola pelanggan yang Anda buat.




d. Pilih Salin snapshot.

7. Ketika salinan snapshot tersedia, pilih.
8. Untuk Tindakan, pilih Bagikan snapshot.
9. Pada halaman izin Snapshot:

a. Masukkan Akun AWS ID yang Anda gunakan untuk membagikan salinan snapshot, lalu pilih Tambah.

b. Pilih Simpan.

Snapshot dibagikan.

### Salin snapshot bersama di akun target

Sekarang Anda dapat menyalin snapshot bersama di target Akun AWS.

Untuk menyalin snapshot bersama

1. Masuk ke AWS Management Console dari target Akun AWS.
2. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/
3. Di panel navigasi, pilih Snapshot.
4. Pilih tab Dibagikan dengan saya.
5. Pilih snapshot bersama.
6. Untuk Tindakan, pilih Salin snapshot.
7. Pilih pengaturan Anda untuk menyalin snapshot seperti pada prosedur sebelumnya, tetapi gunakan AWS KMS key yang termasuk dalam akun target.

   Pilih Salin snapshot.

## Menghentikan berbagi snapshot

Untuk berhenti membagikan snapshot cluster DB, Anda menghapus izin dari target Akun AWS.

Konsol

Untuk berhenti berbagi snapshot cluster DB manual dengan Akun AWS

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot manual yang pembagiannya ingin Anda hentikan.

4. Pilih Tindakan, lalu pilih Bagikan snapshot.
5. Untuk menghapus izin untuk Akun AWS, pilih Hapus untuk pengenal AWS akun untuk akun tersebut dari daftar akun resmi.
6. Pilih Simpan untuk menyimpan perubahan Anda.

CLI

Untuk menghapus Akun AWS pengenal dari daftar, gunakan `--values-to-remove` parameter.

Example menghentikan berbagi snapshot

Contoh berikut mencegah Akun AWS ID 444455556666 memulihkan snapshot.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

Untuk Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS API

Untuk menghapus izin berbagi untuk sebuah Akun AWS, gunakan `ModifyDBClusterSnapshotAttribute`operasi dengan `AttributeName` set to `restore` dan `ValuesToRemove` parameter. Untuk menandai snapshot manual sebagai privat, hapus nilai `all` dari daftar nilai untuk atribut `restore`.

# Mengekspor data klaster DB ke Amazon S3

Anda dapat mengekspor data dari klaster DB Amazon Aurora langsung ke bucket Amazon S3. Proses ekspor berjalan di latar belakang dan tidak memengaruhi performa klaster DB aktif Anda.

Secara default, semua data dalam klaster DB diekspor. Namun, Anda dapat memilih untuk mengekspor set basis data, skema, atau tabel tertentu.

Amazon Aurora mengklona klaster DB, mengekstrak data dari klona, dan menyimpan data dalam bucket Amazon S3. Data disimpan dalam format Apache Parquet yang dikompresi dan konsisten. Setiap file Parquet biasanya berukuran 1–10 MB.

Kinerja lebih cepat yang bisa Anda dapatkan dengan mengekspor data snapshot untuk Aurora My SQL versi 2 dan versi 3 tidak berlaku untuk mengekspor data cluster DB. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Anda dikenakan biaya untuk mengekspor seluruh klaster DB, baik Anda mengekspor semua atau sebagian data. Untuk informasi selengkapnya, lihat Halaman Harga Amazon Aurora.

Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi lebih lanjut tentang menggunakan Athena untuk membaca data Parket, lihat Parket di Panduan Pengguna SerDe Amazon Athena. Untuk informasi selengkapnya tentang penggunaan Redshift Spectrum untuk membaca data Parket, lihat COPYdari format data kolumnar di Panduan Pengembang Database Amazon Redshift.

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin basis data dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah pengeksporan data klaster DB ke S3, lihat Daerah yang Didukung dan mesin Aurora DB untuk mengekspor data cluster ke Amazon S3.

Anda menggunakan proses berikut untuk mengekspor data klaster DB ke bucket Amazon S3. Untuk detail selengkapnya, lihat bagian berikut.

Ikhtisar pengeksporan data klaster DB

1. Identifikasi klaster DB yang datanya ingin Anda ekspor.
2. Atur akses ke bucket Amazon S3.

   Bucket adalah kontainer untuk objek atau file Amazon S3. Untuk memberikan informasi untuk mengakses bucket, lakukan langkah-langkah berikut:

a. Identifikasi bucket S3 tempat data klaster DB akan diekspor. Bucket S3 harus berada di Wilayah AWS yang sama dengan klaster DB. Untuk informasi selengkapnya, lihat Mengidentifikasi bucket Amazon S3 untuk ekspor.

b. Buat peran AWS Identity and Access Management (IAM) yang memberikan akses tugas ekspor klaster DB ke bucket S3. Untuk informasi selengkapnya, lihat Menyediakan akses ke bucket Amazon S3 menggunakan peran IAM.

3. Buat enkripsi simetris AWS KMS key untuk enkripsi sisi server. KMSKuncinya digunakan oleh tugas ekspor cluster untuk mengatur enkripsi AWS KMS sisi server saat menulis data ekspor ke S3.

   Kebijakan KMS utama harus menyertakan izin `kms:CreateGrant` dan `kms:DescribeKey` izin. Untuk informasi selengkapnya tentang penggunaan KMS kunci di Amazon Aurora, lihat. AWS KMS key manajemen

   Jika Anda memiliki pernyataan penolakan dalam kebijakan KMS utama Anda, pastikan untuk secara eksplisit mengecualikan prinsip AWS layanan. `export.rds.amazonaws.com`

   Anda dapat menggunakan KMS kunci dalam AWS akun Anda, atau Anda dapat menggunakan KMS kunci lintas akun. Untuk informasi selengkapnya, lihat Menggunakan cross-account AWS KMS key.

4. Ekspor cluster DB ke Amazon S3 menggunakan konsol atau perintah. `start-export-task` CLI Untuk informasi selengkapnya, lihat Membuat tugas ekspor cluster DB.

5. Untuk mengakses data Anda yang diekspor di bucket Amazon S3 lihat Mengunggah, mengunduh, dan mengelola objek dalam Panduan Pengguna Amazon Simple Storage Service.

Pelajari cara mengatur, mengekspor, memantau, membatalkan, dan memecahkan masalah tugas ekspor klaster DB di bagian berikut.

Topik

- Pertimbangan untuk ekspor cluster DB
- Menyiapkan akses ke bucket Amazon S3
- Membuat tugas ekspor cluster DB
- Memantau tugas ekspor klaster DB
- Membatalkan tugas ekspor klaster DB
- Memecahkan masalah ekspor cluster DB

# Pertimbangan untuk ekspor cluster DB

Gunakan bagian berikut untuk mempelajari tentang batasan, konvensi penamaan file, serta konversi dan penyimpanan data saat mengekspor data cluster DB ke Amazon S3.

Topik


- Batasan
- Konvensi penamaan file
- Konversi data dan format penyimpanan


## Batasan

Mengekspor data klaster DB ke Amazon S3 memiliki batasan sebagai berikut:

- Anda tidak dapat menjalankan beberapa tugas ekspor untuk klaster DB yang sama secara bersamaan. Batasan ini berlaku untuk ekspor penuh dan sebagian.
- Anda dapat memiliki hingga lima tugas ekspor snapshot DB bersamaan yang sedang berlangsung per Akun AWS.
- Aurora Serverless v1 Cluster DB tidak mendukung ekspor ke S3.
- Aurora My dan SQL Aurora Postgre SQL mendukung ekspor ke S3 hanya untuk mode mesin yang disediakan.
- Ekspor ke S3 tidak mendukung awalan S3 yang berisi titik dua (:).
- Karakter berikut di jalur file S3 akan diubah menjadi garis bawah (_) selama ekspor berlangsung:

```
\ ` " (space)
```

- Jika basis data, skema, atau tabel memiliki karakter dalam namanya selain yang berikut ini, maka ekspor parsial tidak didukung. Namun, Anda dapat mengekspor seluruh klaster DB.
  - Huruf latin (A–Z)
  - Digit (0–9)
  - Simbol dolar ($)
  - Garis bawah (_)
- Spasi ( ) dan karakter-karakter tertentu tidak didukung dalam nama kolom tabel basis data. Tabel yang nama kolomnya berisi karakter berikut akan dilewati selama ekspor berlangsung:

```
, ; { } ( ) \n \t = (space)
```

- Tabel yang namanya berisi garis miring (/) akan dilewati selama ekspor berlangsung.
- Aurora Postgre tabel SQL sementara dan tidak tercatat dilewati selama ekspor.
- Jika data berisi objek besar, seperti BLOB atauCLOB, yang mendekati atau lebih besar dari 500 MB, maka ekspor gagal.
- Jika suatu tabel berisi baris besar yang berukuran mendekati atau lebih dari 2 GB, maka tabel tersebut akan dilewati selama ekspor berlangsung.
- Untuk ekspor sebagian, `ExportOnly` daftar memiliki ukuran maksimum 200 KB.
- Sebaiknya Anda menggunakan nama unik untuk setiap tugas ekspor. Jika tidak menggunakan nama tugas yang unik, Anda mungkin menerima pesan kesalahan berikut:

  ExportTaskAlreadyExistsFault: Terjadi kesalahan (ExportTaskAlreadyExists) saat memanggil StartExportTask operasi: Tugas ekspor dengan ID *xxxxx* sudah ada.
- Karena beberapa tabel mungkin dilewati, sebaiknya Anda memverifikasi jumlah baris dan tabel dalam data setelah ekspor.

## Konvensi penamaan file

Data yang diekspor untuk tabel tertentu disimpan dalam format *base_prefix*/*files*, dengan prefiks dasar sebagai berikut:

```
export_identifier/database_name/schema_name.table_name/
```

Contohnya:

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

File keluaran menggunakan konvensi penamaan berikut, di mana *partition_index* adalah alfanumerik:

```
partition_index/part-00000-random_uuid.format-based_extension
```

Sebagai contoh:

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
```

```
    a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

Konvensi penamaan file dapat berubah sewaktu-waktu. Oleh karena itu, ketika membaca tabel target, sebaiknya baca segala sesuatu di dalam prefiks dasar untuk tabel.

### Konversi data dan format penyimpanan

Saat Anda mengekspor cluster DB ke bucket Amazon S3, Amazon Aurora mengonversi, mengekspor, dan menyimpan data dalam format Parket. Untuk informasi selengkapnya, lihat Konversi data saat mengekspor ke bucket Amazon S3.

## Menyiapkan akses ke bucket Amazon S3

Anda mengidentifikasi bucket Amazon S3, lalu memberikan izin tugas ekspor klaster DB untuk mengaksesnya.

Topik

- Mengidentifikasi bucket Amazon S3 untuk ekspor
- Menyediakan akses ke bucket Amazon S3 menggunakan peran IAM
- Menggunakan bucket Amazon S3 lintas akun

### Mengidentifikasi bucket Amazon S3 untuk ekspor

Identifikasi bucket Amazon S3 untuk mengekspor data klaster DB tujuan. Gunakan bucket S3 yang ada atau buat bucket S3 baru.

> **Note**
>
> Bucket S3 harus berada di AWS Region yang sama dengan cluster DB.

Untuk informasi selengkapnya tentang cara bekerja dengan bucket Amazon S3, lihat informasi berikut dalam Panduan Pengguna Amazon Simple Storage Service:

- Bagaimana cara melihat properti untuk bucket S3?
- Bagaimana cara mengaktifkan enkripsi default untuk bucket Amazon S3?

- Bagaimana cara membuat bucket S3?

## Menyediakan akses ke bucket Amazon S3 menggunakan peran IAM

Sebelum Anda mengekspor data klaster DB ke Amazon S3, berikan izin akses tulis tugas ekspor ke bucket Amazon S3.

Untuk memberikan izin ini, buat IAM kebijakan yang menyediakan akses ke bucket, lalu buat IAM peran dan lampirkan kebijakan ke peran tersebut. Nanti, Anda dapat menetapkan IAM peran ke tugas ekspor cluster DB Anda.

> **Important**
>
> Jika berencana menggunakan klaster AWS Management Console to export DB, Anda dapat memilih untuk membuat IAM kebijakan dan peran secara otomatis saat mengekspor klaster DB. Untuk petunjuk, silakan lihat Membuat tugas ekspor cluster DB.

Untuk memberikan akses tugas ke Amazon S3

1. Buat IAM kebijakan. Kebijakan ini memberikan bucket dan izin objek yang memungkinkan tugas ekspor klaster DB Anda mengakses Amazon S3.

   Dalam kebijakan tersebut, sertakan tindakan yang diperlukan berikut untuk mengizinkan transfer file dari Amazon Aurora ke bucket S3:

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   Dalam kebijakan tersebut, sertakan sumber daya berikut untuk mengidentifikasi bucket S3 dan objek dalam bucket. Daftar sumber daya berikut menunjukkan format Amazon Resource Name (ARN) untuk mengakses Amazon S3.

   - `arn:aws:s3:::amzn-s3-demo-bucket`

- `arn:aws:s3:::amzn-s3-demo-bucket/*`

Untuk informasi selengkapnya tentang membuat IAM kebijakan untuk Amazon Aurora, lihat. Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM Lihat juga Tutorial: Buat dan lampirkan kebijakan terkelola pelanggan pertama Anda di Panduan IAM Pengguna.

AWS CLI Perintah berikut membuat IAM kebijakan bernama `ExportPolicy` dengan opsi ini. Ini memberikan akses ke ember bernama *amzn-s3-demo-bucket*.

> **Note**
>
> Setelah Anda membuat kebijakan, perhatikan kebijakan tersebut. ARN Anda memerlukan langkah selanjutnya saat Anda melampirkan kebijakan ke suatu IAM peran. ARN

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. Buat IAM peran, sehingga Aurora dapat mengambil IAM peran ini atas nama Anda untuk mengakses bucket Amazon S3 Anda. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin kepada IAM pengguna di Panduan IAM Pengguna.

Contoh berikut menunjukkan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-export-role`.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Lampirkan IAM kebijakan yang Anda buat ke IAM peran yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat sebelumnya ke peran bernama`rds-s3-export-role`. Ganti *`your-policy-arn`* dengan kebijakan ARN yang Anda catat di langkah sebelumnya.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

### Menggunakan bucket Amazon S3 lintas akun

Anda dapat menggunakan bucket S3 di seluruh AWS akun. Untuk informasi selengkapnya, lihat Menggunakan bucket Amazon S3 lintas akun.

## Membuat tugas ekspor cluster DB

Buat tugas ekspor untuk mengekspor data dari cluster Aurora DB Anda ke bucket Amazon S3. Anda dapat memiliki hingga lima tugas ekspor klaster DB bersamaan yang sedang berlangsung per Akun AWS.

> **Note**
>
> Mengekspor data klaster DB dapat memakan waktu, tergantung jenis dan ukuran basis data Anda. Tugas ekspor mengklona dan menskalakan seluruh basis data terlebih dahulu sebelum mengekstrak data ke Amazon S3. Progres tugas selama fase ini ditampilkan sebagai Memulai. Saat tugas beralih menjadi mengekspor data ke S3, progres akan ditampilkan sebagai Sedang berlangsung.
> Waktu yang diperlukan untuk menyelesaikan ekspor tergantung pada data yang disimpan di basis data. Misalnya, tabel dengan kunci primer numerik atau kolom indeks terdistribusi dengan baik mengekspor paling cepat. Tabel yang tidak berisi kolom yang sesuai untuk partisi dan tabel dengan hanya satu indeks pada kolom berbasis string membutuhkan waktu lebih lama karena pengeksporan menggunakan proses single-threaded yang lebih lambat.

Anda dapat mengekspor data cluster DB ke Amazon S3 menggunakan AWS Management Console, file AWS CLI, atau file. RDS API

Jika Anda menggunakan fungsi Lambda untuk mengekspor klaster DB, tambahkan tindakan `kms:DescribeKey` terhadap kebijakan fungsi Lambda. Untuk informasi selengkapnya, lihat izin AWS Lambda.

Konsol

Opsi konsol Ekspor ke Amazon S3 hanya muncul untuk klaster DB yang dapat diekspor ke Amazon S3. Klaster DB mungkin tidak tersedia untuk diekspor karena alasan berikut:

- Mesin DB tidak didukung untuk ekspor S3.
- Versi klaster DB tidak didukung untuk ekspor S3.
- Ekspor S3 tidak didukung di AWS Wilayah tempat cluster DB dibuat.

Untuk mengekspor data klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB yang datanya ingin Anda ekspor.
4. Untuk Tindakan, pilih Ekspor ke Amazon S3.

Jendela Ekspor ke Amazon S3 akan muncul.

5. Untuk Pengidentifikasi ekspor, masukkan nama untuk mengidentifikasi tugas ekspor. Nilai ini juga akan digunakan untuk nama file yang dibuat di bucket S3.
6. Pilih data yang akan diekspor:

   - Pilih Semua untuk mengekspor semua data dalam klaster DB.
   - Pilih Parsial untuk mengekspor bagian tertentu dari klaster DB. Untuk mengidentifikasi bagian klaster mana yang akan diekspor, masukkan satu atau beberapa basis data, skema, atau tabel untuk Pengidentifikasi, dipisahkan dengan spasi.

     Gunakan format berikut:

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
     [.tablen]
     ```

     Contohnya:

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```
7. Untuk Bucket S3, pilih bucket yang akan dijadikan tujuan ekspor.

   Untuk menetapkan data yang diekspor ke jalur folder dalam bucket S3, masukkan jalur opsional untuk Prefiks S3.
8. Untuk IAMperan, pilih peran yang memberi Anda akses tulis ke bucket S3 pilihan Anda, atau buat peran baru.

   - Jika Anda membuat peran dengan mengikuti langkah-langkah di Menyediakan akses ke bucket Amazon S3 menggunakan peran IAM, pilih peran tersebut.
   - Jika Anda tidak membuat peran yang memberi Anda akses tulis ke bucket S3 yang Anda pilih, pilih Buat peran baru untuk membuat peran secara otomatis. Selanjutnya, masukkan nama untuk peran dalam nama IAM peran.
9. Untuk KMSkunci, masukkan kunci ARN untuk digunakan untuk mengenkripsi data yang diekspor.
10. Pilih Ekspor ke Amazon S3.

AWS CLI

Untuk mengekspor data cluster DB ke Amazon S3 menggunakan AWS CLI, gunakan start-export-taskperintah dengan opsi yang diperlukan berikut:

- `--export-task-identifier`
- `--source-arn`— Nama Sumber Daya Amazon (ARN) dari cluster DB
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

Dalam contoh berikut, tugas ekspor diberi nama *`my-cluster-export`*, yang mengekspor data ke bucket S3 bernama *`amzn-s3-demo-destination-bucket`*.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Untuk Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Berikut adalah contoh output.

```
{
    "ExportTaskIdentifier": "my-cluster-export",
```

```
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
}
```

Untuk menyediakan jalur folder di bucket S3 untuk ekspor cluster DB, sertakan `--s3-prefix` opsi dalam start-export-taskperintah.

RDS API

Untuk mengekspor data cluster DB ke Amazon S3 menggunakan Amazon RDSAPI, gunakan StartExportTaskoperasi dengan parameter yang diperlukan berikut:

- `ExportTaskIdentifier`
- `SourceArn`— ARN dari cluster DB
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

## Memantau tugas ekspor klaster DB

Anda dapat memantau ekspor cluster DB menggunakan AWS Management Console, AWS CLI, atau file. RDS API

Konsol

Untuk memantau ekspor klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor klaster DB ditunjukkan di kolom Tipe sumber. Status ekspor ditampilkan di kolom Status.
3. Untuk melihat informasi detail tentang ekspor klaster khusus, pilih tugas ekspor.

AWS CLI

Untuk memantau tugas ekspor cluster DB menggunakan AWS CLI, gunakan describe-export-tasksperintah.

Contoh berikut menunjukkan cara menampilkan informasi saat ini tentang semua ekspor klaster DB Anda.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
```

```
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket amzn-s3-demo-bucket2 isn't located in the
 current AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```

Untuk menampilkan informasi tentang ekspor khusus, sertakan opsi `--export-task-identifier` dengan perintah `describe-export-tasks`. Sertakan opsi `--Filters` untuk memfilter output. Untuk opsi lainnya, lihat describe-export-tasksperintah.

RDS API

Untuk menampilkan informasi tentang ekspor cluster DB menggunakan Amazon RDSAPI, gunakan DescribeExportTasksoperasi.

Untuk melacak penyelesaian alur kerja ekspor atau memulai alur kerja lainnya, Anda dapat berlangganan topik Amazon Simple Notification Service. Untuk informasi lebih lanjut tentang AmazonSNS, lihatBekerja dengan pemberitahuan RDS acara Amazon.

## Membatalkan tugas ekspor klaster DB

Anda dapat membatalkan tugas ekspor klaster DB menggunakan AWS Management Console, file AWS CLI, atau file RDSAPI.

> **Note**
>
> Membatalkan tugas ekspor tidak menghapus data apa pun yang diekspor ke Amazon S3. Untuk mengetahui informasi tentang cara menghapus data menggunakan konsol,

lihat Bagaimana cara menghapus objek dari bucket S3? Untuk menghapus data menggunakanCLI, gunakan perintah delete-object.

Konsol

Untuk membatalkan tugas ekspor klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor klaster DB ditunjukkan di kolom Tipe sumber. Status ekspor ditampilkan di kolom Status.
3. Pilih tugas ekspor yang ingin Anda batalkan.
4. Pilih Batalkan.
5. Pilih Batalkan tugas ekspor di halaman konfirmasi.

AWS CLI

Untuk membatalkan tugas ekspor menggunakan AWS CLI, gunakan cancel-export-taskperintah. Perintah tersebut memerlukan opsi `--export-task-identifier`.

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

RDS API

Untuk membatalkan tugas ekspor menggunakan Amazon RDSAPI, gunakan CancelExportTaskoperasi dengan `ExportTaskIdentifier` parameter.

## Memecahkan masalah ekspor cluster DB

Gunakan bagian berikut untuk membantu memecahkan masalah pesan kegagalan dan kesalahan SQL izin Postgre untuk tugas ekspor klaster DB ke Amazon S3.

### Pesan kegagalan untuk tugas ekspor Amazon S3

Tabel berikut menjelaskan pesan yang akan ditampilkan jika tugas ekspor Amazon S3 gagal.

| Pesan kegagalan | Deskripsi |
| --- | --- |
| Gagal menemukan atau mengakses klaster DB sumber: [nama klaster] | Klaster DB sumber tidak dapat diklona. |
| Terjadi kesalahan internal yang tidak diketahui. | Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan. |
| Terjadi kesalahan internal yang tidak diketahui saat menulis metadata tugas ekspor ke bucket S3 [nama bucket]. | Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan. |
| RDSEkspor gagal menulis metadata tugas ekspor karena tidak dapat mengambil IAM peran [peranARN]. | Tugas ekspor mengasumsikan IAM peran Anda untuk memvalidasi apakah diperbolehkan menulis metadata ke bucket S3 Anda. Jika tugas tidak dapat mengambil IAM peran Anda, itu gagal. |
| RDSEkspor gagal menulis metadata tugas ekspor ke bucket S3 [nama bucket] menggunakan IAM role [roleARN] dengan KMS kunci [key ID]. Kode kesalahan: [kode kesalahan] | Satu atau beberapa izin tidak ada, sehingga tugas ekspor tidak dapat mengakses bucket S3. Pesan kegagalan ini muncul saat menerima salah satu kode kesalahan berikut:<br>• `AWSSecurityTokenServiceException` dengan kode kesalahan `AccessDenied`<br>• `AmazonS3Exception` dengan kode kesalahan `NoSuchBucket` , `AccessDenied` , `KMS.KMSIn` |

| Pesan kegagalan | Deskripsi |
| --- | --- |
| | `validStateException` , `403 Forbidden` , atau `KMS.DisabledException`<br><br>Kode kesalahan ini menunjukkan bahwa pengaturan salah dikonfigurasi untuk IAM peran, bucket S3, atau KMS kunci. |
| IAMRole [roleARN] tidak diizinkan untuk memanggil [S3 action] pada bucket S3 [nama bucket]. Tinjau izin Anda dan coba lagi ekspor. | IAMKebijakan ini salah dikonfigurasi. Izin untuk tindakan S3 tertentu pada bucket S3 tidak ada, sehingga menyebabkan tugas ekspor gagal. |
| KMSpemeriksaan kunci gagal. Periksa kredensil pada KMS kunci Anda dan coba lagi. | Pemeriksaan KMS kredensi kunci gagal. |
| Pemeriksaan kredensial S3 gagal. Periksa izin pada bucket dan IAM kebijakan S3 Anda. | Pemeriksaan kredensial S3 gagal. |
| Bucket S3 [nama bucket] tidak valid. Bucket tersebut tidak berada di Wilayah AWS saat ini atau tidak ada. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor. | Bucket S3 tidak valid. |
| Bucket S3 [nama bucket] tidak berada di Wilayah AWS saat ini. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor. | Ember S3 salah Wilayah AWS. |

## Memecahkan masalah kesalahan izin Postgre SQL

Saat mengekspor SQL database Postgre ke Amazon S3, Anda mungkin melihat `PERMISSIONS_DO_NOT_EXIST` kesalahan yang menyatakan bahwa tabel tertentu dilewati. Kesalahan ini biasanya terjadi saat superuser, yang Anda tetapkan saat membuat klaster DB, tidak memiliki izin untuk mengakses tabel tersebut.

Untuk memperbaiki kesalahan ini, jalankan perintah berikut:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

Untuk informasi selengkapnya tentang hak pengguna super, lihat Hak akses akun pengguna master.

# Mengekspor data snapshot klaster DB ke Amazon S3

Anda dapat mengekspor data snapshot klaster DB ke bucket Amazon S3. Proses ekspor berjalan di latar belakang dan tidak memengaruhi performa klaster DB aktif Anda.

Saat Anda mengekspor snapshot klaster DB, Amazon Aurora akan mengekstrak data dari snapshot tersebut dan menyimpannya di bucket Amazon S3. Anda dapat mengekspor snapshot manual dan snapshot sistem otomatis. Secara default, semua data dalam snapshot akan diekspor. Namun, Anda dapat memilih untuk mengekspor set basis data, skema, atau tabel tertentu.

> **Note**
>
> Mengekspor data dari snapshot cluster DB memerlukan pemulihan snapshot. Waktu pemulihan dipengaruhi oleh berbagai faktor, seperti jumlah lalu lintas jaringan dan Wilayah AWS menerima relatif terhadap bandwidth yang tersedia. Ketika ada peningkatan lalu lintas yang tiba-tiba, Anda mungkin mengalami waktu penyelesaian yang lebih lama dari yang diharapkan.
> Alternatif untuk mengurangi waktu ekspor S3 untuk database Aurora adalah ekspor cluster DB langsung ke S3. Ekspor cluster DB memiliki waktu mulai yang lebih pendek daripada ekspor snapshot DB, karena tidak perlu mengembalikan snapshot. Untuk informasi selengkapnya, lihat Mengekspor data klaster DB ke Amazon S3.

Data disimpan dalam format Apache Parquet yang dikompresi dan konsisten. Setiap file Parquet biasanya berukuran 1–10 MB.

Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi lebih lanjut tentang menggunakan Athena untuk membaca data Parket, lihat Parket di Panduan Pengguna SerDe Amazon Athena. Untuk informasi selengkapnya tentang penggunaan Redshift Spectrum untuk membaca data Parket, lihat COPYdari format data kolumnar di Panduan Pengembang Database Amazon Redshift.

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin database dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah untuk mengekspor data snapshot cluster DB ke S3, lihat. Daerah yang Didukung dan mesin Aurora DB untuk mengekspor data snapshot ke Amazon S3

Anda menggunakan proses berikut untuk mengekspor data snapshot DB ke bucket Amazon S3. Untuk detail selengkapnya, lihat bagian berikut.

Ringkasan pengeksporan data snapshot

1. Identifikasi snapshot yang akan diekspor.

   Gunakan snapshot otomatis atau manual yang ada, atau buat snapshot manual instans DB.

2. Siapkan akses ke bucket Amazon S3.

   Bucket adalah kontainer untuk objek atau file Amazon S3. Untuk memberikan informasi agar dapat mengakses bucket, lakukan langkah-langkah berikut:

   a. Identifikasi bucket S3 tempat snapshot akan diekspor. Bucket S3 harus sama AWS Wilayah sebagai snapshot. Untuk informasi selengkapnya, lihat Mengidentifikasi bucket Amazon S3 untuk ekspor.

   b. Buat sebuah AWS Identity and Access Management (IAM) peran yang memberikan akses tugas ekspor snapshot ke bucket S3. Untuk informasi selengkapnya, lihat Memberikan akses ke bucket Amazon S3 menggunakan peran IAM.

3. Buat enkripsi simetris AWS KMS key untuk enkripsi sisi server. KMSKuncinya digunakan oleh tugas ekspor snapshot untuk mengatur AWS KMS enkripsi sisi server saat menulis data ekspor ke S3.

   Kebijakan KMS utama harus menyertakan izin `kms:CreateGrant` dan `kms:DescribeKey` izin. Untuk informasi selengkapnya tentang penggunaan KMS kunci di Amazon Aurora, lihat. AWS KMS key manajemen

   Jika Anda memiliki pernyataan penolakan dalam kebijakan KMS utama Anda, pastikan untuk secara eksplisit mengecualikan AWS kepala layanan`export.rds.amazonaws.com`.

   Anda dapat menggunakan KMS kunci di dalam AWS akun, atau Anda dapat menggunakan KMS kunci lintas akun. Untuk informasi selengkapnya, lihat Menggunakan cross-account AWS KMS key.

4. Ekspor snapshot ke Amazon S3 menggunakan konsol atau `start-export-task` CLI perintah. Untuk informasi selengkapnya, lihat Membuat tugas ekspor snapshot.

5. Untuk mengakses data Anda yang diekspor di bucket Amazon S3 lihat Mengunggah, mengunduh, dan mengelola objek dalam Panduan Pengguna Amazon Simple Storage Service.

Pelajari cara mengatur, mengekspor, memantau, membatalkan, dan memecahkan masalah tugas ekspor snapshot cluster DB di bagian berikut.

Topik


- Pertimbangan untuk ekspor snapshot cluster DB
- Menyiapkan akses ke bucket Amazon S3
- Membuat tugas ekspor snapshot
- Memantau ekspor snapshot
- Membatalkan tugas ekspor snapshot
- Ekspor kinerja di Aurora My SQL
- Memecahkan masalah ekspor snapshot


# Pertimbangan untuk ekspor snapshot cluster DB

## Batasan

Batasan berikut berlaku untuk ekspor data snapshot DB ke Amazon S3:

- Anda tidak dapat menjalankan beberapa tugas ekspor untuk snapshot klaster DB yang sama secara bersamaan. Batasan ini berlaku untuk ekspor penuh dan sebagian.
- Anda dapat memiliki hingga lima tugas ekspor snapshot DB bersamaan yang sedang berlangsung per. Akun AWS
- Anda tidak dapat mengekspor data snapshot dari Aurora Serverless v1 Cluster DB ke S3.
- Ekspor ke S3 tidak mendukung awalan S3 yang berisi titik dua (:).
- Karakter berikut di jalur file S3 akan diubah menjadi garis bawah (_) selama ekspor berlangsung:

```
\ ` " (space)
```

- Jika basis data, skema, atau tabel memiliki karakter dalam namanya selain yang berikut ini, maka ekspor parsial tidak didukung. Namun, Anda dapat mengekspor seluruh snapshot DB.
  - Huruf latin (A-Z)
  - Digit (0–9)
  - Simbol dolar ($)
  - Garis bawah (_)

- Spasi ( ) dan karakter-karakter tertentu tidak didukung dalam nama kolom tabel basis data. Tabel yang nama kolomnya berisi karakter berikut akan dilewati selama ekspor berlangsung:

  ```
  , ; { } ( ) \n \t = (space)
  ```

- Tabel yang namanya berisi garis miring (/) akan dilewati selama ekspor berlangsung.
- Tabel Aurora PostgreSQL yang bersifat sementara dan tidak tercatat akan dilewati selama ekspor berlangsung.
- Jika data berisi objek besar, seperti BLOB atau CLOB, yang berukuran mendekati atau lebih dari 500 MB, maka ekspornya akan gagal.
- Jika suatu tabel berisi baris besar yang berukuran mendekati atau lebih dari 2 GB, maka tabel tersebut akan dilewati selama ekspor berlangsung.
- Untuk ekspor sebagian, `ExportOnly` daftar memiliki ukuran maksimum 200 KB.
- Sebaiknya Anda menggunakan nama unik untuk setiap tugas ekspor. Jika tidak menggunakan nama tugas yang unik, Anda mungkin menerima pesan kesalahan berikut:

  ExportTaskAlreadyExistsFault: Terjadi kesalahan (ExportTaskAlreadyExists) saat memanggil StartExportTask operasi: Tugas ekspor dengan ID *xxxxx* sudah ada.
- Anda dapat menghapus snapshot saat sedang mengekspor datanya ke S3, tetapi Anda masih dikenai biaya penyimpanan untuk snapshot tersebut hingga tugas ekspor selesai.
- Anda tidak dapat memulihkan data snapshot yang diekspor dari S3 ke klaster DB baru.

## Konvensi penamaan file

Data yang diekspor untuk tabel tertentu disimpan dalam format *base_prefix*/*files*, dengan prefiks dasar sebagai berikut:

```
export_identifier/database_name/schema_name.table_name/
```

Contohnya:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

Ada dua konvensi cara penamaan file.

- Konvensi saat ini:

```
batch_index/part-partition_index-random_uuid.format-based_extension
```

Indeks batch adalah nomor urut yang mewakili batch data yang dibaca dari tabel. Jika kami tidak dapat mempartisi tabel Anda menjadi bagian-bagian kecil untuk diekspor secara paralel, akan ada beberapa indeks batch. Hal yang sama akan terjadi jika tabel Anda dipartisi menjadi beberapa tabel. Beberapa indeks batch akan tersedia, dengan satu untuk setiap partisi tabel dari tabel utama Anda.

Jika kami dapat mempartisi tabel Anda menjadi bagian-bagian kecil yang akan dibaca secara paralel, hanya akan ada folder 1 indeks batch.

Di dalam folder indeks batch, akan ada satu atau beberapa file Parquet yang berisi data tabel Anda. Prefiks file Parket adalah `part-`*`partition_index`*. Jika tabel Anda dipartisi, akan ada beberapa file yang diawali dengan indeks partisi `00000`.

Mungkin ada kesenjangan dalam urutan indeks partisi. Hal ini terjadi karena setiap partisi diperoleh dari kueri dengan rentang di tabel Anda. Jika tidak ada data dalam rentang partisi tersebut, maka nomor urut itu akan dilewati.

Misalnya, anggap kolom `id` adalah kunci primer tabel, dan nilai minimum dan maksimumnya adalah `100` dan `1000`. Saat kami mencoba mengekspor tabel ini dengan sembilan partisi, kami membacanya dengan kueri paralel seperti berikut:

```
SELECT * FROM table WHERE id <= 100 AND id < 200
 SELECT * FROM table WHERE id <= 200 AND id < 300
```

Partisi ini akan menghasilkan sembilan file, dari `part-00000-`*`random_uuid`*`.gz.parquet` hingga `part-00008-`*`random_uuid`*`.gz.parquet`. Namun, jika tidak ada baris dengan IDs antara `200` dan`350`, salah satu partisi selesai kosong, dan tidak ada file yang dibuat untuk itu. Dalam contoh sebelumnya, `part-00001-`*`random_uuid`*`.gz.parquet` tidak dibuat.

- Konvensi yang lebih lama:

```
part-partition_index-random_uuid.format-based_extension
```

Konvensi ini sama seperti konvensi saat ini, tetapi tanpa prefiks *`batch_index`*, contohnya:

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
```

```
part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

Konvensi penamaan file dapat berubah sewaktu-waktu. Oleh karena itu, saat membaca tabel target, sebaiknya baca segala sesuatu di dalam prefiks dasar untuk tabel tersebut.

### Konversi data saat mengekspor ke bucket Amazon S3

Saat Anda mengekspor snapshot DB ke bucket Amazon S3 Amazon Aurora akan mengonversi data ke, mengekspor data dalam, dan menyimpan data dalam format Parquet. Untuk informasi selengkapnya tentang Parquet, lihat situs web Apache Parquet.

Parquet menyimpan semua data sebagai salah satu jenis primitif berikut:

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – Array byte dengan panjang variabel, juga dikenal sebagai biner
- FIXED_LEN_BYTE_ARRAY – Array byte dengan panjang tetap yang digunakan saat nilai memiliki ukuran konstan

Jenis data Parquet berjumlah sedikit untuk mengurangi kerumitan membaca dan menulis format. Parquet menyediakan jenis logis untuk memperluas jenis primitif. Jenis logis diimplementasikan sebagai anotasi dengan data di kolom metadata `LogicalType`. Anotasi jenis logis menjelaskan cara menginterpretasikan jenis primitif.

Saat jenis logis `STRING` menganotasi jenis `BYTE_ARRAY`, hal ini menunjukkan bahwa array byte harus diinterpretasikan sebagai string karakter yang dienkode UTF-8. Setelah tugas ekspor selesai, Amazon Aurora akan memberi tahu Anda jika terjadi konversi string. Data dasar yang diekspor selalu sama seperti data dari sumbernya. Namun, karena perbedaan encoding dalam UTF-8, beberapa karakter mungkin terlihat berbeda dari sumber saat dibaca di alat seperti Athena.

Untuk informasi selengkapnya, lihat Parquet logical type definitions dalam dokumentasi Parquet.

Topik


- Pemetaan jenis data MySQL ke Parquet
- Pemetaan jenis data PostgreSQL ke Parquet


## Pemetaan jenis data MySQL ke Parquet

Tabel berikut menunjukkan pemetaan dari jenis data MySQL ke jenis data Parquet saat data dikonversi dan diekspor ke Amazon S3.

| Jenis data sumber | Jenis primitif Parquet | Anotasi jenis logis | Catatan konversi |
|---|---|---|---|
| Jenis data numerik | | | |
| BIGINT | INT64 | | |
| BIGINT UNSIGNED | FIXED_LEN _BYTE_ARRAY(9) | DECIMAL(20,0) | Parquet hanya mendukung jenis yang ditandatangani, sehingga pemetaann ya memerlukan tambahan byte (8 plus 1) untuk menyimpan jenis BIGINT_UN SIGNED. |
| BIT | BYTE_ARRAY | | |
| DECIMAL | INT32 | DECIMAL(p,s) | Jika nilai sumber kurang dari $2^{31}$, itu disimpan sebagai INT32. |
| | INT64 | DECIMAL(p,s) | Jika nilai sumber adalah $2^{31}$ atau lebih besar, tetapi kurang dari $2^{63}$, itu disimpan sebagaiINT64. |

| Jenis data sumber | Jenis primitif Parquet | Anotasi jenis logis | Catatan konversi |
|---|---|---|---|
| | FIXED_LEN _BYTE_ARRAY(N) | DECIMAL(p,s) | Jika nilai sumber adalah $2^{63}$ atau lebih besar, maka nilai tersebut akan disimpan sebagai FIXED_LEN _BYTE_ARRAY(N). |
| | BYTE_ARRAY | STRING | Parquet tidak mendukung presisi Desimal yang lebih besar dari 38. Nilai desimal dikonversi ke string dalam tipe BYTE_ARRAY dan dikodekan sebagai. UTF8 |
| DOUBLE | DOUBLE | | |
| FLOAT | DOUBLE | | |
| INT | INT32 | | |
| INT UNSIGNED | INT64 | | |
| MEDIUMINT | INT32 | | |
| MEDIUMINT UNSIGNED | INT64 | | |
| NUMERIC | INT32 | DECIMAL(p,s) | Jika nilai sumber kurang dari $2^{31}$, itu disimpan sebagaiIN T32. |

| Jenis data sumber | Jenis primitif Parquet | Anotasi jenis logis | Catatan konversi |
|---|---|---|---|
| | INT64 | DECIMAL(p,s) | Jika nilai sumber adalah $2^{31}$ atau lebih besar, tetapi kurang dari $2^{63}$, itu disimpan sebagaiINT64. |
| | FIXED_LEN _ARRAY(N) | DECIMAL(p,s) | Jika nilai sumber adalah $2^{63}$ atau lebih besar, maka nilai tersebut akan disimpan sebagai FIXED_LEN _BYTE_ARRAY(N). |
| | BYTE_ARRAY | STRING | Parquet tidak mendukung presisi Numerik yang lebih besar dari 38. Nilai Numerik ini dikonvers i ke string dalam tipe BYTE_ARRAY dan dikodekan sebagai. UTF8 |
| SMALLINT | INT32 | | |
| SMALLINT UNSIGNED | INT32 | | |
| TINYINT | INT32 | | |
| TINYINT UNSIGNED | INT32 | INT (16, benar) | |
| Jenis data string | | | |
| BINARY | BYTE_ARRAY | | |

| Jenis data sumber | Jenis primitif Parquet | Anotasi jenis logis | Catatan konversi |
|---|---|---|---|
| BLOB | BYTE_ARRAY | | |
| CHAR | BYTE_ARRAY | | |
| ENUM | BYTE_ARRAY | STRING | |
| LINESTRING | BYTE_ARRAY | | |
| LONGBLOB | BYTE_ARRAY | | |
| LONGTEXT | BYTE_ARRAY | STRING | |
| MEDIUMBLOB | BYTE_ARRAY | | |
| MEDIUMTEXT | BYTE_ARRAY | STRING | |
| MULTILINESTRING | BYTE_ARRAY | | |
| SET | BYTE_ARRAY | STRING | |
| TEXT | BYTE_ARRAY | STRING | |
| TINYBLOB | BYTE_ARRAY | | |
| TINYTEXT | BYTE_ARRAY | STRING | |
| VARBINARY | BYTE_ARRAY | | |
| VARCHAR | BYTE_ARRAY | STRING | |
| Jenis data tanggal dan waktu | | | |
| DATE | BYTE_ARRAY | STRING | Tanggal dikonversi ke string dalam tipe BYTE_ARRAY dan dikodekan sebagai. UTF8 |

| Jenis data sumber | Jenis primitif Parquet | Anotasi jenis logis | Catatan konversi |
|---|---|---|---|
| DATETIME | INT64 | TIMESTAMP _MICROS | |
| TIME | BYTE_ARRAY | STRING | Tipe TIME dikonversi ke string dalam BYTE_ARRAY dan dikodekan sebagai. UTF8 |
| TIMESTAMP | INT64 | TIMESTAMP _MICROS | |
| YEAR | INT32 | | |
| Jenis data geometris | | | |
| GEOMETRY | BYTE_ARRAY | | |
| GEOMETRYCOLLECTION | BYTE_ARRAY | | |
| MULTIPOINT | BYTE_ARRAY | | |
| MULTIPOLYGON | BYTE_ARRAY | | |
| POINT | BYTE_ARRAY | | |
| POLYGON | BYTE_ARRAY | | |
| Jenis data JSON | | | |
| JSON | BYTE_ARRAY | STRING | |

Pemetaan jenis data PostgreSQL ke Parquet

Tabel berikut menunjukkan pemetaan dari dan jenis data PostgreSQL ke jenis data Parquet saat data dikonversi dan diekspor ke Amazon S3.

| Jenis data PostgreSQL | Jenis primitif Parquet | Anotasi jenis logis | Catatan pemetaan |
|---|---|---|---|
| Jenis data numerik | | | |
| BIGINT | INT64 | | |
| BIGSERIAL | INT64 | | |
| DECIMAL | BYTE_ARRAY | STRING | Tipe DECIMAL dikonversi ke string dalam tipe BYTE_ARRAY dan dikodekan sebagai. UTF8<br><br>Konversi ini dimaksudkan untuk menghindari kerumitan akibat presisi data dan nilai data yang bukan berupa angka (NaN). |
| DOUBLE PRECISION | DOUBLE | | |
| INTEGER | INT32 | | |
| MONEY | BYTE_ARRAY | STRING | |
| REAL | FLOAT | | |
| SERIAL | INT32 | | |
| SMALLINT | INT32 | INT (16, benar) | |
| SMALLSERIAL | INT32 | INT (16, benar) | |
| Jenis data string dan terkait | | | |

| Jenis data PostgreSQL | Jenis primitif Parquet | Anotasi jenis logis | Catatan pemetaan |
|---|---|---|---|
| ARRAY | BYTE_ARRAY | STRING | Array dikonversi ke string dan dikodekan sebagai BINARY ()UTF8.<br><br>Konversi ini dimaksudkan untuk menghindari kerumitan akibat presisi data, nilai data yang bukan berupa angka (NaN), dan nilai data waktu. |
| BIT | BYTE_ARRAY | STRING | |
| BIT VARYING | BYTE_ARRAY | STRING | |
| BYTEA | BINARY | | |
| CHAR | BYTE_ARRAY | STRING | |
| CHAR(N) | BYTE_ARRAY | STRING | |
| ENUM | BYTE_ARRAY | STRING | |
| NAME | BYTE_ARRAY | STRING | |
| TEXT | BYTE_ARRAY | STRING | |
| TEXT SEARCH | BYTE_ARRAY | STRING | |
| VARCHAR(N) | BYTE_ARRAY | STRING | |
| XML | BYTE_ARRAY | STRING | |

| Jenis data PostgreSQL | Jenis primitif Parquet | Anotasi jenis logis | Catatan pemetaan |
|---|---|---|---|
| Jenis data tanggal dan waktu | | | |
| DATE | BYTE_ARRAY | STRING | |
| INTERVAL | BYTE_ARRAY | STRING | |
| TIME | BYTE_ARRAY | STRING | |
| TIME WITH TIME ZONE | BYTE_ARRAY | STRING | |
| TIMESTAMP | BYTE_ARRAY | STRING | |
| TIMESTAMP WITH TIME ZONE | BYTE_ARRAY | STRING | |
| Jenis data geometris | | | |
| BOX | BYTE_ARRAY | STRING | |
| CIRCLE | BYTE_ARRAY | STRING | |
| LINE | BYTE_ARRAY | STRING | |
| LINESEGMENT | BYTE_ARRAY | STRING | |
| PATH | BYTE_ARRAY | STRING | |
| POINT | BYTE_ARRAY | STRING | |
| POLYGON | BYTE_ARRAY | STRING | |
| Jenis data JSON | | | |
| JSON | BYTE_ARRAY | STRING | |
| JSONB | BYTE_ARRAY | STRING | |
| Jenis data lainnya | | | |

| Jenis data PostgreSQL | Jenis primitif Parquet | Anotasi jenis logis | Catatan pemetaan |
|---|---|---|---|
| BOOLEAN | BOOLEAN | | |
| CIDR | BYTE_ARRAY | STRING | Jenis data jaringan |
| COMPOSITE | BYTE_ARRAY | STRING | |
| DOMAIN | BYTE_ARRAY | STRING | |
| INET | BYTE_ARRAY | STRING | Jenis data jaringan |
| MACADDR | BYTE_ARRAY | STRING | |
| OBJECT IDENTIFIER | N/A | | |
| PG_LSN | BYTE_ARRAY | STRING | |
| RANGE | BYTE_ARRAY | STRING | |
| UUID | BYTE_ARRAY | STRING | |

## Menyiapkan akses ke bucket Amazon S3

Identifikasi bucket Amazon S3, lalu berikan izin snapshot untuk mengaksesnya.

Topik

- Mengidentifikasi bucket Amazon S3 untuk ekspor
- Memberikan akses ke bucket Amazon S3 menggunakan peran IAM
- Menggunakan bucket Amazon S3 lintas akun
- Menggunakan cross-account AWS KMS key

### Mengidentifikasi bucket Amazon S3 untuk ekspor

Identifikasi bucket Amazon S3 untuk mengekspor snapshot DB. Gunakan bucket S3 yang ada atau buat bucket S3 baru.

> **Note**
>
> Bucket S3 yang akan diekspor harus berada di AWS Wilayah yang sama dengan snapshot.

Untuk informasi selengkapnya tentang cara bekerja dengan bucket Amazon S3, lihat informasi berikut dalam Panduan Pengguna Amazon Simple Storage Service:

- Bagaimana cara melihat properti untuk bucket S3?
- Bagaimana cara mengaktifkan enkripsi default untuk bucket Amazon S3?
- Bagaimana cara membuat bucket S3?

### Memberikan akses ke bucket Amazon S3 menggunakan peran IAM

Sebelum Anda mengekspor data snapshot DB ke Amazon S3, beri tugas ekspor snapshot izin akses tulis ke bucket Amazon S3.

Untuk memberikan izin ini, buat kebijakan IAM yang memberikan akses ke bucket, lalu buat peran IAM dan lampirkan kebijakan pada peran tersebut. Setelah itu, Anda dapat menetapkan peran IAM ke tugas ekspor snapshot Anda.

> **Important**
>
> Jika Anda berencana untuk menggunakan AWS Management Console untuk mengekspor snapshot Anda, Anda dapat memilih untuk membuat kebijakan IAM dan peran secara otomatis ketika Anda mengekspor snapshot. Untuk petunjuk, lihat Membuat tugas ekspor snapshot.

Untuk memberi tugas snapshot DB akses ke Amazon S3

1. Buat kebijakan IAM. Kebijakan ini memberi bucket dan objek izin yang memungkinkan tugas ekspor snapshot Anda mengakses Amazon S3.

   Dalam kebijakan tersebut, sertakan tindakan yang diperlukan berikut untuk memungkinkan transfer file dari Amazon Aurora ke bucket S3:

   - `s3:PutObject*`

- `s3:GetObject*`
- `s3:ListBucket`
- `s3:DeleteObject*`
- `s3:GetBucketLocation`

Dalam kebijakan tersebut, sertakan sumber daya berikut untuk mengidentifikasi bucket S3 dan objek dalam bucket. Daftar sumber daya berikut menunjukkan format Amazon Resource Name (ARN) untuk mengakses Amazon S3.

- `arn:aws:s3:::amzn-s3-demo-bucket`
- `arn:aws:s3:::amzn-s3-demo-bucket/*`

Untuk informasi selengkapnya tentang cara membuat kebijakan IAM untuk Amazon Aurora, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda dalam Panduan Pengguna IAM.

AWS CLI Perintah berikut membuat kebijakan IAM bernama `ExportPolicy` dengan opsi ini. Perintah ini akan memberikan akses ke bucket bernama *amzn-s3-demo-bucket*.

> **Note**
>
> Setelah Anda membuat kebijakan, catat ARN kebijakan tersebut. Anda memerlukan ARN ini untuk langkah berikutnya saat Anda melampirkan kebijakan tersebut pada peran IAM.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
```

```
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. Buat peran IAM agar Aurora dapat mengambil peran IAM ini atas nama Anda untuk mengakses bucket Amazon S3 Anda. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin kepada pengguna IAM dalam Panduan Pengguna IAM.

   Contoh berikut menunjukkan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-export-role`.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Lampirkan kebijakan IAM yang Anda buat pada peran IAM yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat sebelumnya ke peran bernama`rds-s3-export-role`. Ganti *`your-policy-arn`* dengan ARN kebijakan yang Anda catat di langkah sebelumnya.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

## Menggunakan bucket Amazon S3 lintas akun

Anda dapat menggunakan bucket Amazon S3 di seluruh akun. AWS Untuk menggunakan bucket lintas akun, tambahkan kebijakan bucket untuk mengizinkan akses ke peran IAM yang Anda gunakan untuk ekspor S3. Untuk informasi selengkapnya, lihat Contoh 2: Pemilik bucket yang memberikan izin bucket lintas akun.

- Lampirkan kebijakan bucket pada bucket Anda, seperti yang ditunjukkan dalam contoh berikut.

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
            ]
        }
    ]
}
```

## Menggunakan cross-account AWS KMS key

Anda dapat menggunakan akun silang AWS KMS key untuk mengenkripsi ekspor Amazon S3. Pertama-tama, tambahkan kebijakan kunci ke akun lokal, lalu tambahkan kebijakan IAM di

akun eksternal. Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS.

Untuk menggunakan kunci KMS lintas akun

1. Tambahkan kebijakan kunci ke akun lokal.

   Contoh berikut memberi `ExampleRole` dan `ExampleUser` di akun eksternal 444455556666 izin di akun lokal 123456789012.

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

2. Tambahkan kebijakan IAM ke akun eksternal tersebut.

   Contoh kebijakan IAM berikut memungkinkan pengguna utama menggunakan kunci KMS di akun 123456789012 untuk operasi kriptografi. Untuk memberikan izin ini kepada `ExampleRole` dan `ExampleUser` di akun 444455556666, lampirkan kebijakan pada keduanya di akun tersebut.

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
```

```
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

## Membuat tugas ekspor snapshot

Buat tugas ekspor snapshot untuk mengekspor data dari snapshot Anda ke bucket Amazon S3. Anda dapat memiliki hingga lima tugas ekspor snapshot DB bersamaan yang sedang berlangsung per. Akun AWS

> **Note**
>
> Mengekspor RDS snapshot dapat memakan waktu cukup lama tergantung pada jenis dan ukuran database Anda. Tugas ekspor akan terlebih dahulu memulihkan dan menskalakan seluruh basis data sebelum mengekstrak data ke Amazon S3. Dalam fase ini, progres tugas tersebut akan ditampilkan sebagai Memulai. Saat tugas beralih menjadi mengekspor data ke S3, progres akan ditampilkan sebagai Sedang berlangsung.
> Waktu yang diperlukan untuk menyelesaikan ekspor tergantung pada data yang disimpan di basis data. Misalnya, tabel yang berisi kolom indeks atau kunci primer numerik yang terdistribusi dengan baik akan diekspor paling cepat. Tabel yang tidak berisi kolom yang sesuai untuk partisi dan tabel yang hanya berisi satu indeks pada kolom berbasis string memerlukan waktu lebih lama. Waktu ekspor yang lebih lama ini terjadi karena ekspor menggunakan proses alur tunggal yang lebih lambat.

Anda dapat mengekspor snapshot DB ke Amazon S3 menggunakan AWS Management Console, file, atau AWS CLI file. RDS API

Jika Anda menggunakan fungsi Lambda untuk mengekspor snapshot, tambahkan tindakan `kms:DescribeKey` ke kebijakan fungsi Lambda. Untuk informasi selengkapnya, lihat izin AWS Lambda.

Konsol

Opsi konsol Ekspor ke Amazon S3 hanya muncul untuk snapshot yang dapat diekspor ke Amazon S3. Snapshot mungkin tidak tersedia untuk diekspor karena alasan berikut:

- Mesin DB tidak didukung untuk ekspor S3.
- Versi instans DB tidak didukung untuk ekspor S3.
- Ekspor S3 tidak didukung di AWS Wilayah tempat snapshot dibuat.

Untuk mengekspor snapshot DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Dari tabnya, pilih jenis snapshot yang ingin Anda ekspor.
4. Dalam daftar snapshot, pilih snapshot yang ingin Anda ekspor.
5. Untuk Tindakan, pilih Ekspor ke Amazon S3.

   Jendela Ekspor ke Amazon S3 akan muncul.
6. Untuk Pengidentifikasi ekspor, masukkan nama untuk mengidentifikasi tugas ekspor. Nilai ini juga akan digunakan untuk nama file yang dibuat di bucket S3.
7. Pilih data yang akan diekspor:

   - Pilih Semua untuk mengekspor semua data dalam snapshot.
   - Pilih Sebagian untuk mengekspor bagian tertentu dari snapshot. Untuk mengidentifikasi bagian snapshot yang akan diekspor, masukkan satu atau beberapa basis data, skema, atau tabel untuk Pengidentifikasi, dipisahkan dengan spasi.

     Gunakan format berikut:

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
     [.tablen]
     ```

     Contohnya:

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```

8. Untuk Bucket S3, pilih bucket yang akan dijadikan tujuan ekspor.

   Untuk menetapkan data yang diekspor ke jalur folder dalam bucket S3, masukkan jalur opsional untuk Prefiks S3.

9. Untuk IAMperan, pilih peran yang memberi Anda akses tulis ke bucket S3 pilihan Anda, atau buat peran baru.

   - Jika Anda membuat peran dengan mengikuti langkah-langkah di Memberikan akses ke bucket Amazon S3 menggunakan peran IAM, pilih peran tersebut.
   - Jika Anda tidak membuat peran yang memberi Anda akses tulis ke bucket S3 yang Anda pilih, pilih Buat peran baru untuk membuat peran secara otomatis. Selanjutnya, masukkan nama untuk peran dalam nama IAM peran.

10. Untuk AWS KMS key, masukkan kunci ARN untuk digunakan untuk mengenkripsi data yang diekspor.
11. Pilih Ekspor ke Amazon S3.

AWS CLI

Untuk mengekspor snapshot DB ke Amazon S3 menggunakan AWS CLI, gunakan perintah dengan start-export-taskopsi yang diperlukan berikut:

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

Dalam contoh berikut, tugas ekspor snapshot diberi nama *`my-snapshot-export`*, yang mengekspor snapshot ke bucket S3 bernama *`amzn-s3-demo-destination-bucket`*.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
```

```
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Untuk Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Berikut adalah contoh output.

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

Untuk menyediakan jalur folder di bucket S3 untuk ekspor snapshot, sertakan `--s3-prefix` opsi dalam perintah. start-export-task

RDS API

Untuk mengekspor snapshot DB ke Amazon S3 menggunakan RDS API Amazon, gunakan operasi dengan StartExportTaskparameter yang diperlukan berikut:

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

# Memantau ekspor snapshot

Anda dapat memantau ekspor snapshot DB menggunakan AWS Management Console, AWS CLI, atau file. RDS API

Konsol

Untuk memantau ekspor snapshot DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor snapshot DB ditunjukkan di kolom Jenis sumber. Status ekspor ditampilkan di kolom Status.
3. Untuk melihat informasi mendetail tentang ekspor snapshot tertentu, pilih tugas ekspornya.

AWS CLI

Untuk memantau ekspor snapshot DB menggunakan AWS CLI, gunakan perintah. describe-export-tasks

Contoh berikut menunjukkan cara menampilkan informasi saat ini tentang semua ekspor snapshot Anda.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
```

```
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
```

```
}
```

Untuk menampilkan informasi tentang ekspor snapshot tertentu, sertakan opsi `--export-task-identifier` dengan perintah `describe-export-tasks`. Sertakan opsi `--Filters` untuk memfilter output. Untuk opsi lainnya, lihat describe-export-tasksperintah.

RDS API

Untuk menampilkan informasi tentang ekspor snapshot DB menggunakan Amazon RDSAPI, gunakan operasi. DescribeExportTasks

Untuk melacak penyelesaian alur kerja ekspor atau memulai alur kerja lainnya, Anda dapat berlangganan topik Amazon Simple Notification Service. Untuk informasi lebih lanjut tentang AmazonSNS, lihatBekerja dengan pemberitahuan RDS acara Amazon.

## Membatalkan tugas ekspor snapshot

Anda dapat membatalkan tugas ekspor snapshot DB menggunakan AWS Management Console, file AWS CLI, atau file. RDS API

> **Note**
>
> Membatalkan tugas ekspor snapshot tidak akan menghapus data apa pun yang telah diekspor ke Amazon S3. Untuk informasi tentang cara menghapus data menggunakan konsol, lihat Bagaimana cara menghapus objek dari bucket S3? Untuk menghapus data menggunakanCLI, gunakan perintah delete-object.

Konsol

Untuk membatalkan tugas ekspor snapshot

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor snapshot DB ditunjukkan di kolom Jenis sumber. Status ekspor ditampilkan di kolom Status.
3. Pilih tugas ekspor snapshot yang ingin Anda batalkan.

4. Pilih Batalkan.
5. Pilih Batalkan tugas ekspor di halaman konfirmasi.

AWS CLI

Untuk membatalkan tugas ekspor snapshot menggunakan AWS CLI, gunakan cancel-export-taskperintah. Perintah tersebut memerlukan opsi `--export-task-identifier`.

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

Untuk membatalkan tugas ekspor snapshot menggunakan Amazon RDSAPI, gunakan CancelExportTaskoperasi dengan `ExportTaskIdentifier` parameter.

## Ekspor kinerja di Aurora My SQL

Aurora My SQL versi 2 dan versi 3 DB snapshot cluster menggunakan mekanisme ekspor canggih untuk meningkatkan kinerja dan mengurangi waktu ekspor. Mekanisme ini mencakup pengoptimalan seperti beberapa thread ekspor dan kueri SQL paralel Aurora My untuk memanfaatkan arsitektur penyimpanan bersama Aurora. Pengoptimalan diterapkan secara adaptif, tergantung pada ukuran dan struktur set data.

Anda tidak perlu mengaktifkan kueri paralel untuk menggunakan proses ekspor yang lebih cepat, tetapi prosesnya memiliki batasan yang sama dengan kueri paralel. Selain itu, nilai data tertentu tidak

didukung, seperti tanggal yang hari dalam sebulannya adalah `0` atau tahunnya adalah `0000`. Untuk informasi selengkapnya, lihat Kueri paralel untuk Amazon Aurora My SQL.

Saat pengoptimalan kinerja diterapkan, Anda mungkin juga melihat file Parket yang jauh lebih besar (~ 200 GB) untuk ekspor Aurora My SQL versi 2 dan 3.

Jika proses ekspor yang lebih cepat tidak dapat digunakan, misalnya karena jenis atau nilai data yang tidak kompatibel, Aurora akan otomatis beralih ke mode ekspor alur tunggal tanpa kueri paralel. Bergantung pada proses yang digunakan dan jumlah data yang akan diekspor, performa ekspor dapat bervariasi.

## Memecahkan masalah ekspor snapshot

Gunakan bagian berikut untuk membantu memecahkan masalah pesan kegagalan dan kesalahan SQL izin Postgre untuk tugas ekspor klaster DB ke Amazon S3.

### Pesan kegagalan untuk tugas ekspor Amazon S3

Tabel berikut menjelaskan pesan yang akan ditampilkan jika tugas ekspor Amazon S3 gagal.

| Pesan kegagalan | Deskripsi |
| --- | --- |
| Terjadi kesalahan internal yang tidak diketahui. | Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan. |
| Terjadi kesalahan internal yang tidak diketahui saat menulis metadata tugas ekspor ke bucket S3 [nama bucket]. | Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan. |
| RDSEkspor gagal menulis metadata tugas ekspor karena tidak dapat mengambil IAM peran [peranARN]. | Tugas ekspor mengasumsikan IAM peran Anda untuk memvalidasi apakah diperbolehkan menulis metadata ke bucket S3 Anda. Jika tugas tidak dapat mengambil IAM peran Anda, itu gagal. |
| RDSEkspor gagal menulis metadata tugas ekspor ke bucket S3 [nama bucket] menggunakan IAM role [roleARN] dengan KMS kunci [key ID]. Kode kesalahan: [kode kesalahan] | Satu atau beberapa izin tidak ada, sehingga tugas ekspor tidak dapat mengakses bucket S3. Pesan kegagalan ini muncul saat menerima salah satu kode kesalahan berikut: |

| Pesan kegagalan | Deskripsi |
| --- | --- |
| | • `AWSSecurityTokenServiceException` dengan kode kesalahan `AccessDenied`<br>• `AmazonS3Exception` dengan kode kesalahan `NoSuchBucket`, `AccessDenied`, `KMS.KMSInvalidStateException`, `403 Forbidden`, atau `KMS.DisabledException`<br><br>Kode kesalahan ini menunjukkan bahwa pengaturan salah dikonfigurasi untuk IAM peran, bucket S3, atau KMS kunci. |
| IAMRole [roleARN] tidak diizinkan untuk memanggil [S3 action] pada bucket S3 [nama bucket]. Tinjau izin Anda dan coba lagi ekspor. | IAMKebijakan ini salah dikonfigurasi. Izin untuk tindakan S3 tertentu pada bucket S3 tidak ada, sehingga menyebabkan tugas ekspor gagal. |
| KMSpemeriksaan kunci gagal. Periksa kredensil pada KMS kunci Anda dan coba lagi. | Pemeriksaan KMS kredensi kunci gagal. |
| Pemeriksaan kredensial S3 gagal. Periksa izin pada bucket dan IAM kebijakan S3 Anda. | Pemeriksaan kredensial S3 gagal. |
| Bucket S3 [nama bucket] tidak valid. Bucket tersebut tidak berada di Wilayah AWS saat ini atau tidak ada. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor. | Bucket S3 tidak valid. |
| Bucket S3 [nama bucket] tidak berada di Wilayah AWS saat ini. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor. | Ember S3 salah Wilayah AWS. |

## Memecahkan masalah kesalahan izin Postgre SQL

Saat mengekspor SQL database Postgre ke Amazon S3, Anda mungkin melihat `PERMISSIONS_DO_NOT_EXIST` kesalahan yang menyatakan bahwa tabel tertentu dilewati. Kesalahan ini biasanya terjadi saat pengguna super, yang Anda tetapkan saat membuat instans DB, tidak memiliki izin untuk mengakses tabel tersebut.

Untuk memperbaiki kesalahan ini, jalankan perintah berikut:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

Untuk informasi selengkapnya tentang hak pengguna super, lihat Hak akses akun pengguna master.

# Memulihkan klaster DB ke waktu tertentu

Anda dapat memulihkan klaster DB ke titik waktu tertentu, sehingga membuat klaster DB baru.

Saat Anda memulihkan klaster DB ke titik waktu tertentu, Anda dapat memilih grup keamanan cloud privat virtual (VPC) default. Atau Anda dapat menerapkan grup keamanan VPC kustom ke klaster DB Anda.

Klaster DB yang dipulihkan secara otomatis dikaitkan dengan klaster DB dan grup parameter DB default. Namun, Anda dapat menerapkan grup parameter kustom dengan menentukannya selama pemulihan.

Amazon Aurora mengunggah catatan log untuk klaster DB ke Amazon S3 terus menerus. Untuk melihat waktu restorable terbaru untuk cluster DB, gunakan AWS CLI describe-db-clustersperintah dan lihat nilai yang dikembalikan di `LatestRestorableTime` bidang untuk cluster DB.

Anda dapat memulihkan ke titik waktu mana pun dalam periode retensi cadangan Anda.
Untuk melihat waktu restorable paling awal untuk cluster DB, gunakan AWS CLI describe-db-clustersperintah dan lihat nilai yang dikembalikan di `EarliestRestorableTime` lapangan untuk cluster DB.

Periode retensi cadangan dari klaster DB yang dipulihkan sama dengan klaster DB sumber.

> **Note**
>
> Informasi dalam topik ini berlaku untuk Amazon Aurora. Untuk informasi tentang cara memulihkan instans DB Amazon RDS, lihat Memulihkan instans DB ke waktu yang ditentukan.
> Untuk informasi tentang pencadangan dan pemulihan klaster DB Aurora, lihat Gambaran umum pencadangan dan pemulihan klaster DB Aurora.
> Untuk Aurora MySQL, Anda dapat memulihkan klaster DB yang disediakan ke klaster DB Aurora Serverless. Untuk informasi selengkapnya, lihat Memulihkan Aurora Serverless v1 klaster DB.
> Anda juga dapat menggunakan AWS Backup untuk mengelola cadangan cluster Amazon Aurora DB. Jika cluster DB Anda dikaitkan dengan rencana cadangan di AWS Backup, paket cadangan itu digunakan untuk point-in-time pemulihan. Untuk informasi, lihat Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup.

Untuk informasi tentang memulihkan klaster Aurora DB atau klaster global dengan versi RDS Extended Support, lihat. Aurora DB atau cluster global dengan Amazon Extended Support RDS

Kembalikan cluster DB ke waktu tertentu dari pencadangan otomatis, cadangan otomatis yang dipertahankan, atau penggunaan AWS Backup.

Topik

- Memulihkan cluster DB ke titik waktu
- Memulihkan klaster DB ke waktu tertentu dari cadangan otomatis yang dipertahankan
- Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup

## Memulihkan cluster DB ke titik waktu

Anda dapat memulihkan cluster DB ke titik waktu menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Memulihkan klaster DB ke waktu tertentu

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pencadangan otomatis.

   Pencadangan otomatis ditampilkan di tab Wilayah Saat Ini.

3. Pilih klaster DB yang ingin Anda pulihkan.
4. Untuk Tindakan, pilih Pulihkan ke titik waktu.

   Jendela Pulihkan ke titik waktu akan muncul.
5. Pilih Waktu pemulihan terbaru untuk memulihkan ke waktu terbaru yang dimungkinkan atau pilih Kustom untuk memilih waktu.

Jika Anda memilih Kustom, masukkan tanggal dan waktu untuk mengembalikan klaster.

> **Note**
>
> Waktu ditampilkan dalam zona waktu lokal Anda, yang ditunjukkan dengan offset dari Coordinated Universal Time (UTC). Misalnya, UTC-5 adalah Waktu Standar Timur/Waktu Musim Panas Tengah.

6. Untuk Pengidentifikasi klaster DB, masukkan nama klaster DB target yang dipulihkan. Nama harus unik.
7. Pilih opsi lain sesuai kebutuhan, seperti kelas instans DB dan konfigurasi penyimpanan klaster DB.

   Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.
8. Pilih Pulihkan ke titik waktu.

AWS CLI

Untuk mengembalikan cluster DB ke waktu tertentu, gunakan AWS CLI perintah restore-db-cluster-to-point-in-time untuk membuat cluster DB baru.

Anda dapat menentukan pengaturan lain. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Pemberian tag sumber daya didukung untuk operasi ini. Saat Anda menggunakan opsi `--tags`, tag klaster DB sumber diabaikan dan tag yang disediakan digunakan. Jika tidak, tag terbaru dari klaster sumber digunakan.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
       --source-db-cluster-identifier mysourcedbcluster \
       --db-cluster-identifier mytargetdbcluster \
       --restore-to-time 2017-10-14T23:45:00.000Z
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
       --source-db-cluster-identifier mysourcedbcluster ^
       --db-cluster-identifier mytargetdbcluster ^
       --restore-to-time 2017-10-14T23:45:00.000Z
```

**Important**

Jika Anda menggunakan konsol untuk memulihkan klaster DB ke waktu tertentu, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk mengembalikan cluster DB ke waktu tertentu, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai `--db-cluster-identifier` nilai pilihan.

### API RDS

Untuk memulihkan klaster DB ke waktu yang ditentukan, panggil operasi API Amazon RDS `RestoreDBClusterToPointInTime` dengan parameter berikut ini:

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

**Important**

Jika Anda menggunakan konsol untuk memulihkan klaster DB ke waktu tertentu, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan API RDS untuk memulihkan klaster DB ke waktu tertentu, pastikan Anda secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
Untuk membuat instance utama untuk cluster DB Anda, panggil operasi RDS API Create DBInstance. Sertakan nama klaster DB sebagai nilai parameter `DBClusterIdentifier`.

# Memulihkan klaster DB ke waktu tertentu dari cadangan otomatis yang dipertahankan

Anda dapat memulihkan klaster DB dari cadangan otomatis yang dipertahankan setelah Anda menghapus klaster DB sumber, jika cadangan tersebut berada dalam periode retensi klaster sumber. Prosesnya mirip dengan pemulihan klaster DB dari cadangan otomatis.

> **Note**
>
> Anda tidak dapat memulihkan klaster DB Aurora Serverless v1 menggunakan prosedur ini, karena pencadangan otomatis untuk klaster Aurora Serverless v1 tidak dipertahankan.

Konsol

Memulihkan klaster DB ke waktu tertentu

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pencadangan otomatis.
3. Pilih tab Dipertahankan.

4. Pilih klaster DB yang ingin Anda pulihkan.
5. Untuk Tindakan, pilih Pulihkan ke titik waktu.

   Jendela Pulihkan ke titik waktu akan muncul.
6. Pilih Waktu pemulihan terbaru untuk memulihkan ke waktu terbaru yang dimungkinkan atau pilih Kustom untuk memilih waktu.

   Jika Anda memilih Kustom, masukkan tanggal dan waktu untuk mengembalikan klaster.

> **Note**
>
> Waktu ditampilkan di zona waktu lokal Anda, yang ditunjukkan dengan offset dari Coordinated Universal Time ()UTC. Misalnya, UTC -5 adalah Waktu Standar Timur/ Waktu Siang Tengah.

7. Untuk Pengidentifikasi klaster DB, masukkan nama klaster DB target yang dipulihkan. Nama harus unik.
8. Pilih opsi lain sesuai kebutuhan, seperti kelas instans DB.

   Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.
9. Pilih Pulihkan ke titik waktu.

AWS CLI

Untuk mengembalikan cluster DB ke waktu tertentu, gunakan AWS CLI perintah restore-db-cluster-to-point-in-time untuk membuat cluster DB baru.

Anda dapat menentukan pengaturan lain. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Pemberian tag sumber daya didukung untuk operasi ini. Saat Anda menggunakan opsi `--tags`, tag klaster DB sumber diabaikan dan tag yang disediakan digunakan. Jika tidak, tag terbaru dari klaster sumber digunakan.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
```

```
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

> **Important**
>
> Jika Anda menggunakan konsol untuk memulihkan cluster DB ke waktu yang ditentukan, Amazon RDS secara otomatis membuat instance utama (penulis) untuk cluster DB Anda. Jika Anda menggunakan AWS CLI untuk mengembalikan cluster DB ke waktu tertentu, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
> Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai nilai opsi `--db-cluster-identifier`.

RDS API

Untuk memulihkan cluster DB ke waktu yang ditentukan, panggil RDS API `RestoreDBClusterToPointInTime`operasi Amazon dengan parameter berikut:

- `SourceDbClusterResourceId`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> Jika Anda menggunakan konsol untuk memulihkan cluster DB ke waktu yang ditentukan, Amazon RDS secara otomatis membuat instance utama (penulis) untuk cluster DB Anda. Jika Anda menggunakan RDS API untuk mengembalikan cluster DB ke waktu tertentu, pastikan untuk secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
> Untuk membuat instance utama untuk cluster DB Anda, panggil RDS API operasi C reateDBInstance. Sertakan nama klaster DB sebagai nilai parameter `DBClusterIdentifier`.

# Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup

Anda dapat menggunakan AWS Backup untuk mengelola cadangan otomatis Anda, dan kemudian mengembalikannya ke waktu yang ditentukan. Untuk melakukan ini, Anda membuat rencana cadangan AWS Backup dan menetapkan cluster DB Anda sebagai sumber daya. Kemudian Anda mengaktifkan pencadangan berkelanjutan untuk PITR dalam aturan pencadangan. Untuk informasi selengkapnya tentang rencana pencadangan dan aturan pencadangan, lihat Panduan Developer tentang Pencadangan AWS .

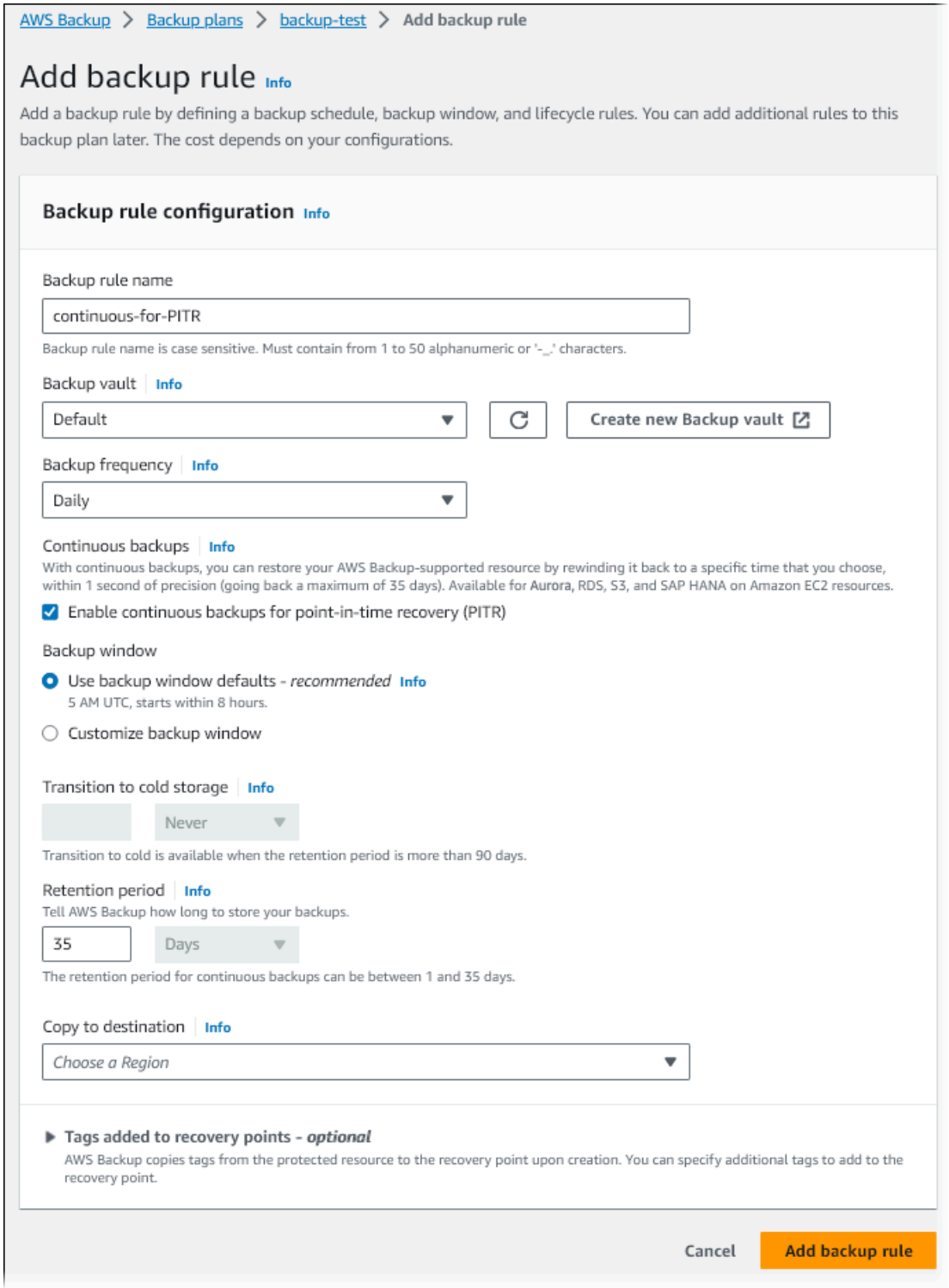
## Mengaktifkan pencadangan berkelanjutan di AWS Backup

Anda mengaktifkan pencadangan berkelanjutan dalam aturan pencadangan.

Untuk mengaktifkan pencadangan berkelanjutan untuk PITR

1. Masuk ke AWS Management Console, dan buka AWS Backup konsol di https://console.aws.amazon.com/backup.
2. Di panel navigasi, pilih Rencana cadangan.
3. Di bawah Nama rencana pencadangan, pilih rencana pencadangan yang Anda gunakan untuk membuat cadangan klaster DB Anda.
4. Di bawah bagian Aturan pencadangan, pilih Tambahkan aturan pencadangan.

   Halaman Tambahkan aturan pencadangan ditampilkan.
5. Pilih kotak centang Aktifkan cadangan berkelanjutan untuk point-in-time pemulihan (PITR).




6. Pilih pengaturan lain sesuai kebutuhan, lalu pilih Tambahkan aturan pencadangan.

## Memulihkan dari cadangan berkelanjutan di AWS Backup

Anda memulihkan ke waktu tertentu dari brankas cadangan.

Konsol

Anda dapat menggunakan AWS Management Console untuk mengembalikan cluster DB ke waktu yang ditentukan.

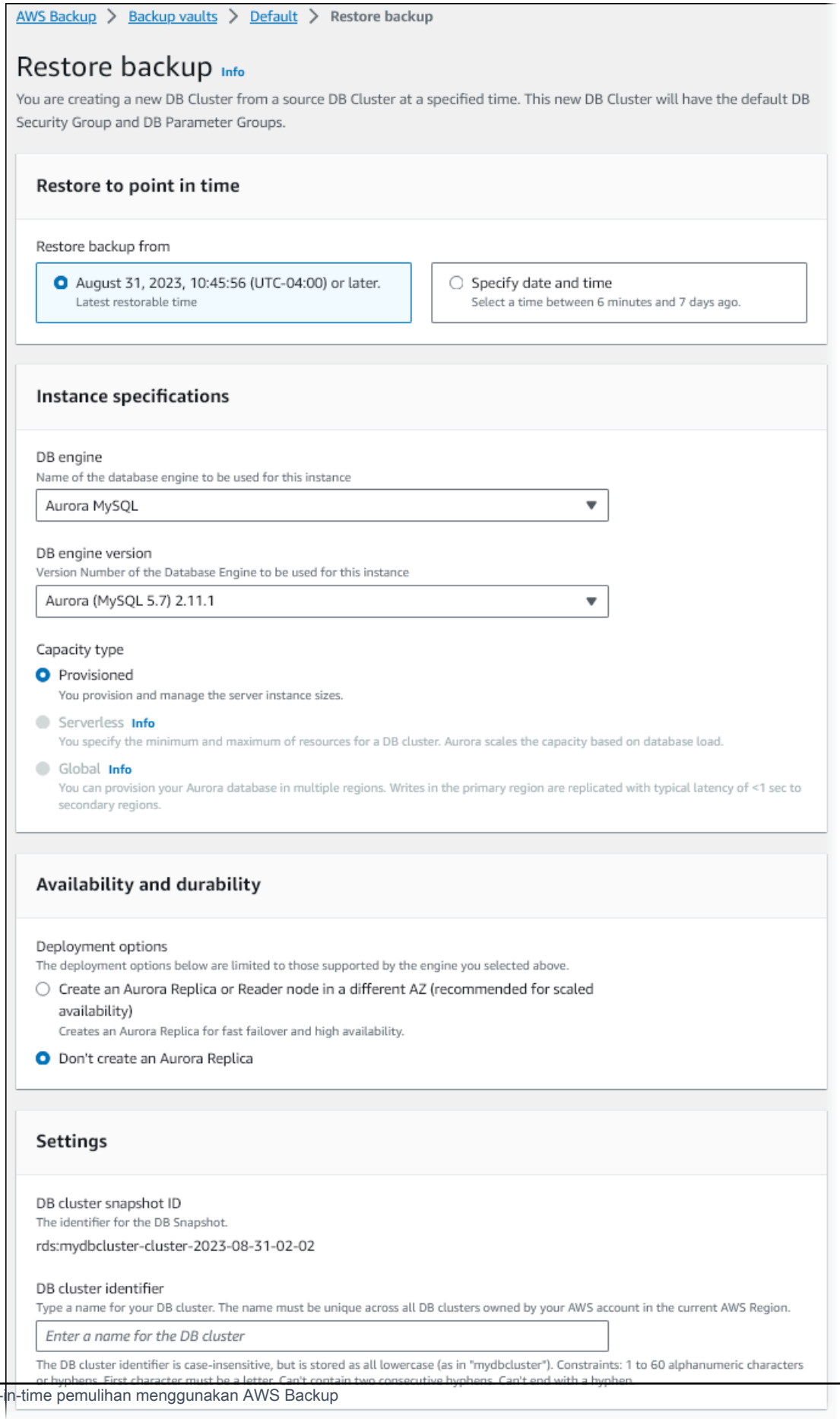
Untuk memulihkan dari cadangan berkelanjutan di AWS Backup

1. Masuk ke AWS Management Console, dan buka AWS Backup konsol di https://console.aws.amazon.com/backup.
2. Di panel navigasi, pilih Brankas cadangan.
3. Pilih brankas cadangan yang berisi cadangan berkelanjutan Anda, misalnya Default.

   Halaman detail brankas cadangan ditampilkan.
4. Di bagian Titik pemulihan, pilih titik pemulihan untuk cadangan otomatis.

   Terdapat jenis cadangan Berkelanjutan dan sebuah nama dengan `continuous:cluster-`*`AWS-Backup-job-number`*.
5. Untuk Tindakan, pilih Pulihkan.

   Halaman Pulihkan cadangan ditampilkan.


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. Untuk Pulihkan ke titik waktu, pilih Tentukan tanggal dan waktu untuk memulihkan ke titik waktu tertentu.
7. Pilih pengaturan lain sesuai kebutuhan untuk memulihkan klaster DB, lalu pilih Pulihkan cadangan.

   Halaman Pekerjaan ditampilkan, menampilkan panel Pekerjaan pemulihan. Pesan di bagian atas halaman memberikan informasi tentang pekerjaan pemulihan.

Setelah klaster DB dipulihkan, Anda harus menambahkan instans DB primer (penulis) ke dalamnya. Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai nilai parameter `--db-cluster-identifier`.

CLI

Anda menggunakan start-restore-job AWS CLI perintah untuk mengembalikan cluster DB ke waktu yang ditentukan. Parameter-parameter berikut diperlukan:

- `--recovery-point-arn`— Nama Sumber Daya Amazon (ARN) untuk titik pemulihan untuk memulihkan.
- `--resource-type` – Gunakan `Aurora`.
- `--iam-role-arn`— ARN Untuk IAM peran yang Anda gunakan untuk AWS Backup operasi.
- `--metadata` – Metadata yang Anda gunakan untuk memulihkan klaster DB. Parameter berikut diperlukan:
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime` atau `UseLatestRestorableTime`

Contoh berikut ini menunjukkan cara memulihkan klaster DB ke waktu tertentu.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

Contoh berikut ini menunjukkan cara memulihkan klaster DB ke waktu pemulihan terbaru.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

Setelah klaster DB dipulihkan, Anda harus menambahkan instans DB primer (penulis) ke dalamnya. Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai nilai parameter `--db-cluster-identifier`.

# Menghapus snapshot klaster DB

Anda dapat menghapus snapshot cluster DB yang dikelola oleh Amazon RDS saat Anda tidak lagi membutuhkannya.

> **Note**
>
> Untuk menghapus cadangan yang dikelola oleh AWS Backup, gunakan konsol. AWS Backup Untuk informasi selengkapnya tentang AWS Backup, lihat Panduan Developer AWS Backup .

## Menghapus snapshot klaster DB

Anda dapat menghapus snapshot cluster DB menggunakan konsol, file AWS CLI, atau file. RDS API

Untuk menghapus snapshot bersama atau publik, Anda harus masuk ke AWS akun yang memiliki snapshot tersebut.

Konsol

Untuk menghapus snapshot klaster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot klaster DB yang ingin Anda hapus.
4. Untuk Tindakan, pilih Hapus snapshot.
5. Pilih Hapus di halaman konfirmasi.

AWS CLI

Anda dapat menghapus snapshot cluster DB dengan menggunakan AWS CLI perintah delete-db-cluster-snapshot.

Opsi berikut digunakan untuk menghapus snapshot klaster DB.

- `--db-cluster-snapshot-identifier` – Pengidentifikasi untuk snapshot klaster DB.

Example

Kode berikut menghapus snapshot klaster DB `mydbclustersnapshot`.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Untuk Windows:

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

Anda dapat menghapus snapshot cluster DB dengan menggunakan RDS API operasi Amazon DeleteDBCluster Snapshot.

Parameter berikut digunakan untuk menghapus snapshot klaster DB.

- `DBClusterSnapshotIdentifier` – Pengidentifikasi untuk snapshot klaster DB.

# Tutorial: Memulihkan klaster DB Amazon Aurora dari snapshot klaster DB

Skenario umum saat bekerja dengan Amazon Aurora adalah memiliki instans DB yang jarang Anda gunakan, tetapi tidak selalu Anda perlukan. Misalnya, Anda mungkin menggunakan klaster DB untuk menyimpan data untuk laporan yang hanya dijalankan setiap tiga bulan. Salah satu cara untuk menghemat biaya pada skenario tersebut adalah dengan mengambil snapshot klaster DB setelah laporan selesai. Kemudian, Anda menghapus klaster DB, dan memulihkannya saat Anda perlu mengunggah data baru dan menjalankan laporan selama kuartal berikutnya.

Saat memulihkan klaster DB, masukkan nama snapshot klaster DB yang akan digunakan untuk memulihkan. Kemudian, masukkan nama untuk klaster DB baru yang dibuat dari operasi pemulihan. Untuk informasi lebih mendetail tentang cara memulihkan klaster DB dari snapshot, lihat Memulihkan dari snapshot klaster DB.

Dalam tutorial ini, kami juga meningkatkan cluster DB yang dipulihkan dari Aurora My SQL versi 2 (kompatibel dengan My SQL 5.7) ke Aurora My SQL versi 3 (kompatibel dengan My 8.0). SQL

Mengembalikan cluster DB ke waktu tertentu dari snapshot cluster DB menggunakan RDS konsol Amazon atau AWS CLI.

Untuk informasi tentang AWS KMS manajemen kunci untuk AmazonRDS, lihatAWS KMS key manajemen.

Topik

- Tutorial: Memulihkan cluster DB dari snapshot cluster DB menggunakan konsol Amazon RDS
- Tutorial: Memulihkan cluster DB dari snapshot cluster DB menggunakan AWS CLI

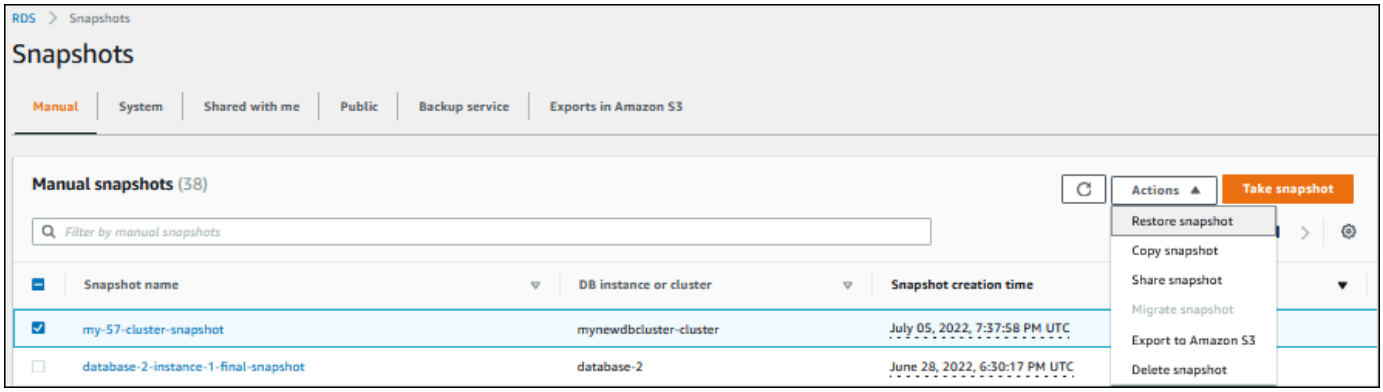
## Tutorial: Memulihkan cluster DB dari snapshot cluster DB menggunakan konsol Amazon RDS

Dalam tutorial ini, Anda memulihkan cluster DB dari snapshot cluster DB menggunakan konsol Amazon RDS. Saat Anda memulihkan cluster DB dari snapshot menggunakan AWS Management Console, instans DB utama (penulis) juga dibuat.

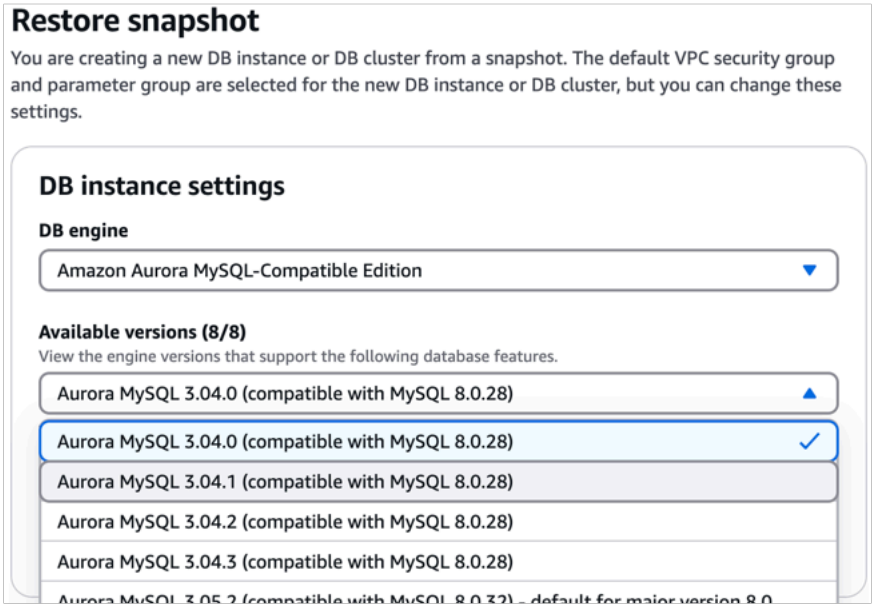
> **Note**
>
> Saat instans DB primer sedang dibuat, instans ini muncul sebagai instans pembaca, tetapi setelah pembuatan, instans ini menjadi instans penulis.

Untuk memulihkan klaster DB dari snapshot klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot klaster DB untuk memulihkan klaster.
4. Untuk Tindakan, pilih Pulihkan snapshot.




   Halaman Pulihkan snapshot muncul.
5. Pada Pengaturan instans DB, lakukan tindakan berikut:
   a. Gunakan pengaturan default untuk mesin DB.
   b. Untuk versi yang tersedia, pilih versi MySQL-8.0 yang kompatibel, seperti Aurora MySQL 3.04.0 (kompatibel dengan MySQL 8.0.28).



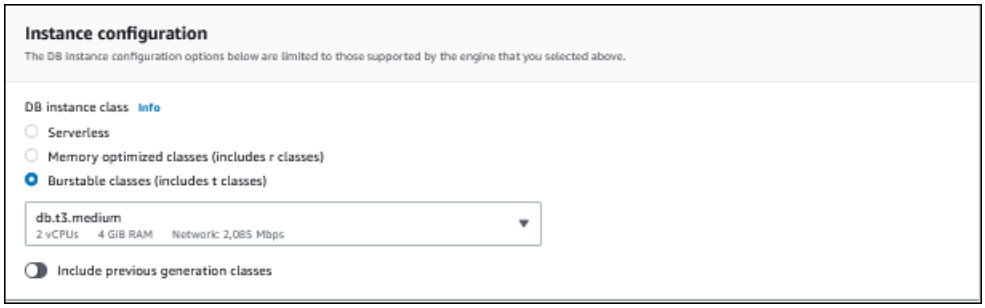

6. Di bagian Pengaturan, untuk pengidentifikasi instans DB masukkan nama unik yang ingin Anda gunakan untuk instans DB yang dipulihkan, misalnya **`my-80`**.

> **Note**
>
> Untuk membuat pengidentifikasi cluster DB, Amazon RDS menambahkan `-cluster` ke pengenal instans DB yang Anda tentukan.

7. Di bagian Konektivitas, gunakan pengaturan default untuk berikut ini:
   - Cloud privat virtual (VPC)
   - Grup subnet DB
   - Akses publik
   - Grup keamanan VPC (firewall)
8. Pilih Kelas instans DB.

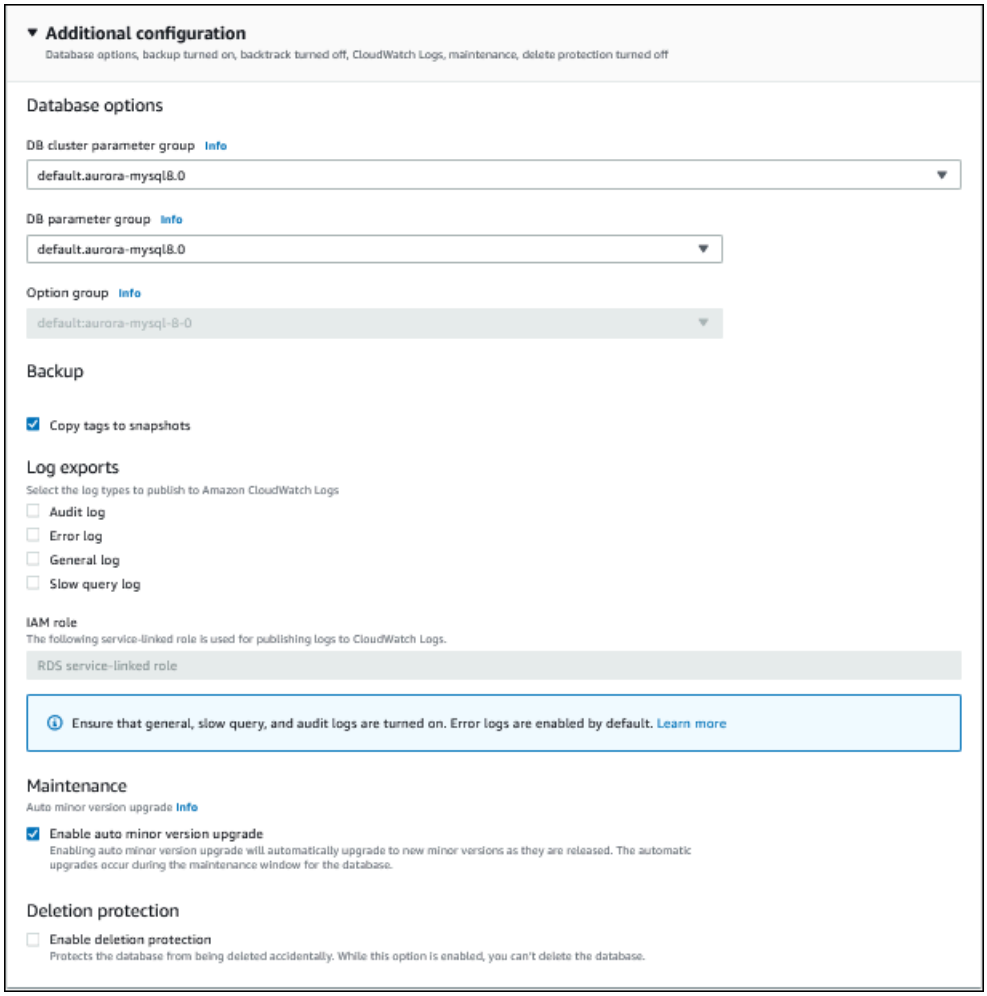
   Untuk tutorial ini, pilih Kelas runtutan (termasuk kelas t), lalu pilih db.t3.medium.

> **Note**
>
> Sebaiknya Anda menggunakan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.




9. Untuk Autentikasi basis data, gunakan pengaturan default.
10. Untuk Enkripsi, gunakan pengaturan default.

    Jika klaster DB sumber untuk snapshot dienkripsi, instans DB yang dipulihkan juga dienkripsi. Anda tidak dapat membuatnya tidak terenkripsi.
11. Luaskan Konfigurasi tambahan di bagian bawah halaman.




12. Buat pilihan berikut:

    a. Untuk tutorial ini, gunakan nilai default untuk grup parameter klaster DB.

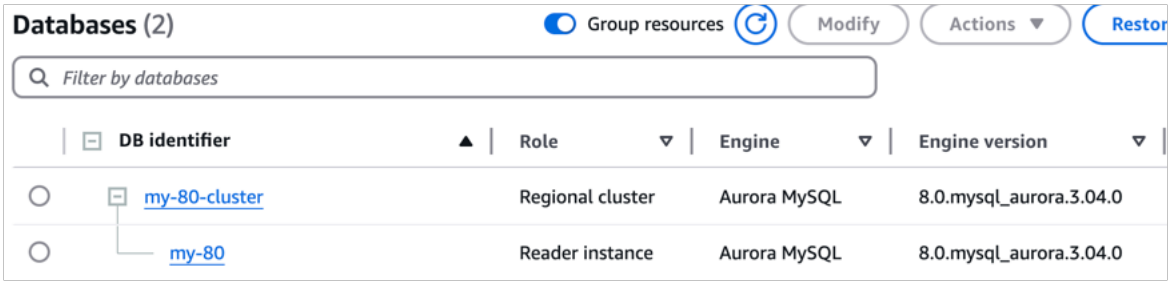
    b. Untuk tutorial ini, gunakan nilai default untuk grup parameter DB.

    c. Untuk Ekspor log, pilih semua kotak centang.

    d. Untuk Perlindungan penghapusan, pilih kotak centang Aktifkan perlindungan penghapusan.

13. Pilih Pulihkan instans DB.

Halaman Basis Data menampilkan klaster DB yang dipulihkan, dengan status `Creating`.




Saat instans DB primer sedang dibuat, instans ini muncul sebagai instans pembaca, tetapi setelah pembuatan, instans ini menjadi instans penulis.

## Tutorial: Memulihkan cluster DB dari snapshot cluster DB menggunakan AWS CLI

Dalam tutorial ini, Anda mengembalikan cluster DB dari snapshot cluster DB menggunakan file. AWS CLI Memulihkan cluster DB dari snapshot menggunakan AWS CLI memiliki dua langkah:

1. Memulihkan klaster DBmenggunakan perintah restore-db-cluster-from-snapshot
2. Membuat instans DB primer (penulis)menggunakan create-db-instanceperintah

### Memulihkan klaster DB

Gunakan perintah `restore-db-cluster-from-snapshot`. Opsi berikut diperlukan:

- `--db-cluster-identifier` – Nama klaster DB yang dipulihkan.
- `--snapshot-identifier` – Nama snapshot DB tempat asal pemulihan.
- `--engine` – Mesin basis data klaster DB yang dipulihkan. Ini harus kompatibel dengan mesin basis data klaster DB sumber.

  Pilihannya adalah sebagai berikut:

  - `aurora-mysql`— Aurora My SQL 5.7 dan 8.0 kompatibel.
  - `aurora-postgresql`— Aurora SQL Postgre kompatibel.

  Dalam contoh ini, kami menggunakan `aurora-mysql`.
- `--engine-version` – Versi klaster DB yang dipulihkan. Dalam contoh ini, kami menggunakan versi My SQL -8.0 yang kompatibel.

Contoh berikut mengembalikan klaster DB yang kompatibel dengan Aurora SQL My 8.0 yang `my-new-80-cluster` dinamai dari snapshot cluster DB bernama. `my-57-cluster-snapshot`

Untuk memulihkan klaster DB

- Gunakan salah satu perintah berikut ini.

  Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

Output-nya menyerupai berikut.

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
```

```
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```


## Membuat instans DB primer (penulis)

Untuk membuat instans DB primer (penulis), gunakan perintah `create-db-instance`. Opsi berikut diperlukan:

- `--db-cluster-identifier` – Nama klaster DB yang dipulihkan.

- `--db-instance-identifier` – Nama instans DB primer.
- `--db-instance-class` – Kelas instans dari instans DB primer. Dalam contoh ini, kami menggunakan `db.t3.medium`.

> **Note**
>
> Sebaiknya Anda menggunakan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail lebih lanjut tentang kelas instans T, lihat Jenis kelas instans DB.

- `--engine` – Mesin basis data instans DB primer. Ini harus berupa mesin basis data yang sama dengan yang digunakan klaster DB yang dipulihkan.

  Pilihannya adalah sebagai berikut:

  - `aurora-mysql`— Aurora My SQL 5.7 dan 8.0 kompatibel.
  - `aurora-postgresql`— Aurora SQL Postgre kompatibel.

  Dalam contoh ini, kami menggunakan `aurora-mysql`.

Contoh berikut membuat instance DB utama (penulis) bernama `my-new-80-cluster-instance` dalam cluster DB Aurora My SQL 8.0 yang kompatibel dengan nama yang dipulihkan. `my-new-80-cluster`

Untuk membuat instans DB primer

- Gunakan salah satu perintah berikut ini.

  Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
    --engine aurora-mysql
```

  Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
```

```
    --db-instance-identifier my-new-80-cluster-instance ^
    --db-instance-class db.t3.medium ^
    --engine aurora-mysql
```

Output-nya menyerupai berikut.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1a"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
```

```
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1b"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1c"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
        "PendingModifiedValues": {},
        "MultiAZ": false,
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:aurora-mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": false,
        "StorageType": "aurora",
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
```

```
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
        "TagList": []
    }
}
```

# Memantau metrik di klaster Amazon Aurora

Amazon Aurora menggunakan klaster server basis data yang direplikasi. Pemantauan klaster Aurora biasanya memerlukan pemeriksaan kondisi pada beberapa instans DB. Instans tersebut mungkin memiliki peran khusus, menangani sebagian besar operasi tulis, operasi hanya baca, atau gabungan keduanya. Anda juga memantau kondisi keseluruhan klaster dengan mengukur jeda replikasi. Ini adalah jumlah waktu untuk perubahan yang diterapkan oleh satu instans DB agar tersedia bagi instans lainnya.

Topik


- Rencana pemantauan
- Acuan dasar performa
- Pedoman performa
- Alat pemantauan untuk Aurora
- Melihat status cluster
-
- Melihat metrik di konsol Amazon RDS
- Melihat metrik gabungan dengan dasbor Performance Insights
- Memantau Amazon Aurora dengan Amazon CloudWatch
- Memantau database Amazon Aurora dengan Database CloudWatch Insights
- Memantau muatan DB dengan Wawasan Performa di Amazon Aurora
- Menganalisis anomali kinerja Aurora dengan Amazon Guru untuk Amazon DevOps RDS
- Memantau metrik OS dengan Pemantauan yang Ditingkatkan
- Referensi metrik untuk Aurora


## Rencana pemantauan

Sebelum Anda mulai memantau Amazon Aurora, buat rencana pemantauan. Rencana ini sepatutnya menjawab pertanyaan-pertanyaan berikut:

- Apa saja sasaran pemantauan Anda?
- Sumber daya manakah yang akan Anda pantau?

- Seberapa seringkah Anda akan memantau sumber daya ini?
- Apa sajakah alat pemantauan yang akan Anda gunakan?
- Siapakah yang akan melakukan tugas pemantauan?
- Siapakah yang harus diberi tahu apabila ada kesalahan?

## Acuan dasar performa

Untuk mencapai semua sasaran pemantauan, Anda perlu menetapkan acuan dasar. Untuk melakukan ini, ukur kinerja dalam kondisi beban yang berbeda pada berbagai waktu di lingkungan Aurora Anda. Anda dapat memantau metrik-metrik seperti:

- Throughput jaringan
- Koneksi klien
- I/O untuk operasi baca, tulis, atau metadata
- Saldo kredit lonjakan untuk instans DB Anda

Kami menyarankan Anda menyimpan data kinerja historis untuk Aurora. Dengan menggunakan data yang disimpan, Anda dapat membandingkan performa saat ini dengan tren sebelumnya. Anda juga dapat membedakan pola performa normal dari anomali, dan merancang teknik untuk mengatasi masalah.

## Pedoman performa

Secara umum, nilai-nilai yang diperkenankan untuk metrik performa bergantung pada performa aplikasi Anda secara relatif terhadap acuan Anda. Selidiki variansi yang konsisten atau sedang tren dari acuan Anda. Metrik-metrik berikut sering menjadi sumber masalah performa:

- Tinggi CPU atau RAM konsumsi — Nilai tinggi untuk CPU atau RAM konsumsi mungkin sesuai, jika sesuai dengan tujuan Anda untuk aplikasi Anda (seperti throughput atau konkurensi) dan diharapkan.
- Konsumsi ruang disk – Selidiki konsumsi ruang disk jika ruang yang digunakan selalu berada pada atau di atas 85 persen dari total ruang disk. Periksa apakah ada kemungkinan untuk menghapus data dari instans atau mengarsipkan data ke sistem yang berbeda guna mengosongkan sebagian ruang.

- Lalu lintas jaringan – Untuk lalu lintas jaringan, bicaralah kepada administrator sistem Anda untuk memahami throughput yang diharapkan bagi jaringan domain dan koneksi internet Anda. Selidiki lalu lintas jaringan jika throughput selalu di bawah yang diharapkan.
- Koneksi basis data – Jika Anda melihat jumlah koneksi pengguna yang tinggi dan juga penurunan performa dan waktu respons instans, pertimbangkan untuk membatasi koneksi basis data. Jumlah koneksi pengguna terbaik untuk instans basis data bervariasi berdasarkan kelas instans dan kerumitan operasi yang dilakukan. Untuk menentukan jumlah koneksi basis data, kaitkan instans basis data Anda dengan grup parameter dengan parameter `User Connections` diatur ke nilai selain 0 (tidak terbatas). Anda dapat menggunakan grup parameter yang ada atau membuat grup baru. Untuk informasi selengkapnya, lihat .
- IOPSmetrik — Nilai yang diharapkan untuk IOPS metrik bergantung pada spesifikasi disk dan konfigurasi server, jadi gunakan baseline Anda untuk mengetahui apa yang khas. Selidiki apakah nilai-nilai selalu berbeda dengan acuan dasar Anda. Untuk IOPS kinerja terbaik, pastikan set kerja tipikal Anda sesuai dengan memori untuk meminimalkan operasi baca dan tulis.

Ketika performa berada di luar garis dasar yang telah ditetapkan, Anda mungkin perlu membuat perubahan untuk mengoptimalkan ketersediaan basis data bagi beban kerja Anda. Misalnya, Anda mungkin perlu mengubah kelas instans dari instans DB Anda. Atau, Anda mungkin perlu mengubah jumlah instans DB dan replika baca yang tersedia untuk klien.

# Alat pemantauan untuk Aurora

Pemantauan adalah bagian penting dalam menjaga keandalan, ketersediaan, dan kinerja Aurora dan yang lain AWS solusi. AWS menyediakan berbagai alat pemantauan untuk menonton Aurora, melaporkan ketika ada sesuatu yang salah, dan mengambil tindakan otomatis bila perlu.

Topik


- Alat-alat pemantauan otomatis
- Alat pemantauan manual


## Alat-alat pemantauan otomatis

Kami menyarankan agar Anda mengautomasi tugas pemantauan sebanyak mungkin.

Topik


- status dan rekomendasi klaster Amazon Aurora
- CloudWatch Metrik Amazon untuk Aurora
- Amazon RDS Performance Insights dan pemantauan sistem operasi
- Layanan terintegrasi


### status dan rekomendasi klaster Amazon Aurora

Anda dapat menggunakan alat otomatis berikut untuk menonton Aurora dan melaporkan ketika ada sesuatu yang salah:

- Status klaster Aurora Amazon Aurora — Lihat detail tentang status klaster saat ini dengan menggunakan RDS konsol Amazon, AWS CLI, atau RDSAPI.
- Rekomendasi RDS Aurora Amazon — Menanggapi rekomendasi otomatis untuk sumber daya database, seperti instans DB, cluster DB, dan grup parameter cluster DB. Untuk informasi selengkapnya, lihat .

### CloudWatch Metrik Amazon untuk Aurora

Amazon Aurora terintegrasi dengan Amazon CloudWatch untuk kemampuan pemantauan tambahan.

- Amazon CloudWatch — Layanan ini memantau Anda AWS sumber daya dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat menggunakan CloudWatch fitur Amazon berikut dengan Amazon Aurora:
  - CloudWatch Metrik Amazon — Aurora secara otomatis mengirimkan metrik ke setiap menit CloudWatch untuk setiap basis data aktif. Anda tidak mendapatkan biaya tambahan untuk RDS metrik Amazon di CloudWatch. Lihat informasi yang lebih lengkap di CloudWatch Metrik Amazon untuk Amazon Aurora
  - CloudWatch Alarm Amazon — Anda dapat menonton satu metrik Amazon Aurora selama periode waktu tertentu. Anda lalu dapat melakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas yang Anda tetapkan.

## Amazon RDS Performance Insights dan pemantauan sistem operasi

Anda dapat menggunakan alat otomatis berikut untuk memantau kinerja Aurora:

- Amazon RDS Performance Insights — Menilai beban pada database Anda, dan menentukan kapan dan di mana harus mengambil tindakan. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Amazon RDS Enhanced Monitoring — Lihat metrik secara real time untuk sistem operasi. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan.

## Layanan terintegrasi

Berikut ini AWS layanan terintegrasi dengan Aurora:

- Amazon EventBridge adalah layanan bus acara tanpa server yang memudahkan untuk menghubungkan aplikasi Anda dengan data dari berbagai sumber. Untuk informasi selengkapnya, lihat .
- Amazon CloudWatch Logs memungkinkan Anda memantau, menyimpan, dan mengakses file log Anda dari instans Aurora Amazon, CloudTrail, dan sumber lainnya. Untuk informasi selengkapnya, lihat Memantau log Amazon Aurora Amazon.
- AWS CloudTrailmenangkap API panggilan dan acara terkait yang dibuat oleh atau atas nama Anda Akun AWS dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Untuk informasi selengkapnya, lihat Memantau panggilan Amazon Aurora API AWS CloudTrail.
- Aliran Aktivitas Database Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

- DevOpsGuru for RDS adalah kemampuan Amazon DevOps Guru yang menerapkan pembelajaran mesin ke metrik Performance Insights untuk database Amazon Aurora. Untuk informasi selengkapnya, lihat Menganalisis anomali kinerja Aurora dengan Amazon Guru untuk Amazon DevOps RDS.

## Alat pemantauan manual

Anda perlu memonitor secara manual item yang tidak CloudWatch tercakup oleh alarm. AmazonRDS, CloudWatch, AWS Trusted Advisor dan lainnya AWS dasbor konsol memberikan at-a-glance tampilan status AWS lingkungan. Kami menyarankan agar Anda juga memeriksa file log pada instans DB Anda.

- Dari RDS konsol Amazon, Anda dapat memantau item berikut untuk sumber daya Anda:
  - Jumlah koneksi dengan instans DB
  - Jumlah operasi baca dan tulis ke instans DB
  - Jumlah penyimpanan yang saat ini digunakan oleh instans DB
  - Jumlah memori dan CPU digunakan untuk instance DB
  - Jumlah lalu lintas jaringan ke dan dari instans DB
- Dari Trusted Advisor dasbor, Anda dapat meninjau pengoptimalan biaya, keamanan, toleransi kesalahan, dan pemeriksaan peningkatan kinerja berikut:
  - Instans Amazon RDS Idle DB
  - Risiko Akses Grup RDS Keamanan Amazon
  - RDSCadangan Amazon
  - Amazon RDS Multi-AZ
  - Aksesibilitas Instans DB Aurora

  Untuk informasi selengkapnya tentang pemeriksaan ini, lihat Praktik terbaik Trusted Advisor (pemeriksaan).
- CloudWatch halaman rumah menunjukkan:
  - Alarm dan status saat ini
  - Grafik alarm dan sumber daya
  - Status kesehatan layanan

Selain itu, Anda dapat menggunakan CloudWatch untuk melakukan hal berikut:

- Membuat dasbor yang disesuaikan untuk memantau layanan yang penting bagi Anda.
- Data metrik grafik untuk memecahkan masalah dan mengungkap tren.
- Cari dan telusuri semua AWS metrik sumber daya.
- Membuat dan mengedit alarm agar diberi tahu tentang masalah.

# Melihat status cluster

Menggunakan konsol Amazon RDS, Anda dapat dengan cepat mengakses status cluster DB Anda.

Topik


- Melihat klaster DB Amazon Aurora
- Melihat status klaster DB
- Melihat status instans DB di klaster Aurora

# Melihat klaster DB Amazon Aurora

Anda memiliki beberapa opsi untuk melihat informasi tentang klaster DB Amazon Aurora dan instans DB dalam klaster DB Anda.

- Anda dapat melihat klaster DB dan instans DB di konsol Amazon RDS dengan memilih Basis data dari panel navigasi.
- Anda bisa mendapatkan cluster DB dan informasi instans DB menggunakan AWS Command Line Interface (AWS CLI).
- Anda bisa mendapatkan informasi klaster DB dan instans DB menggunakan API Amazon RDS.

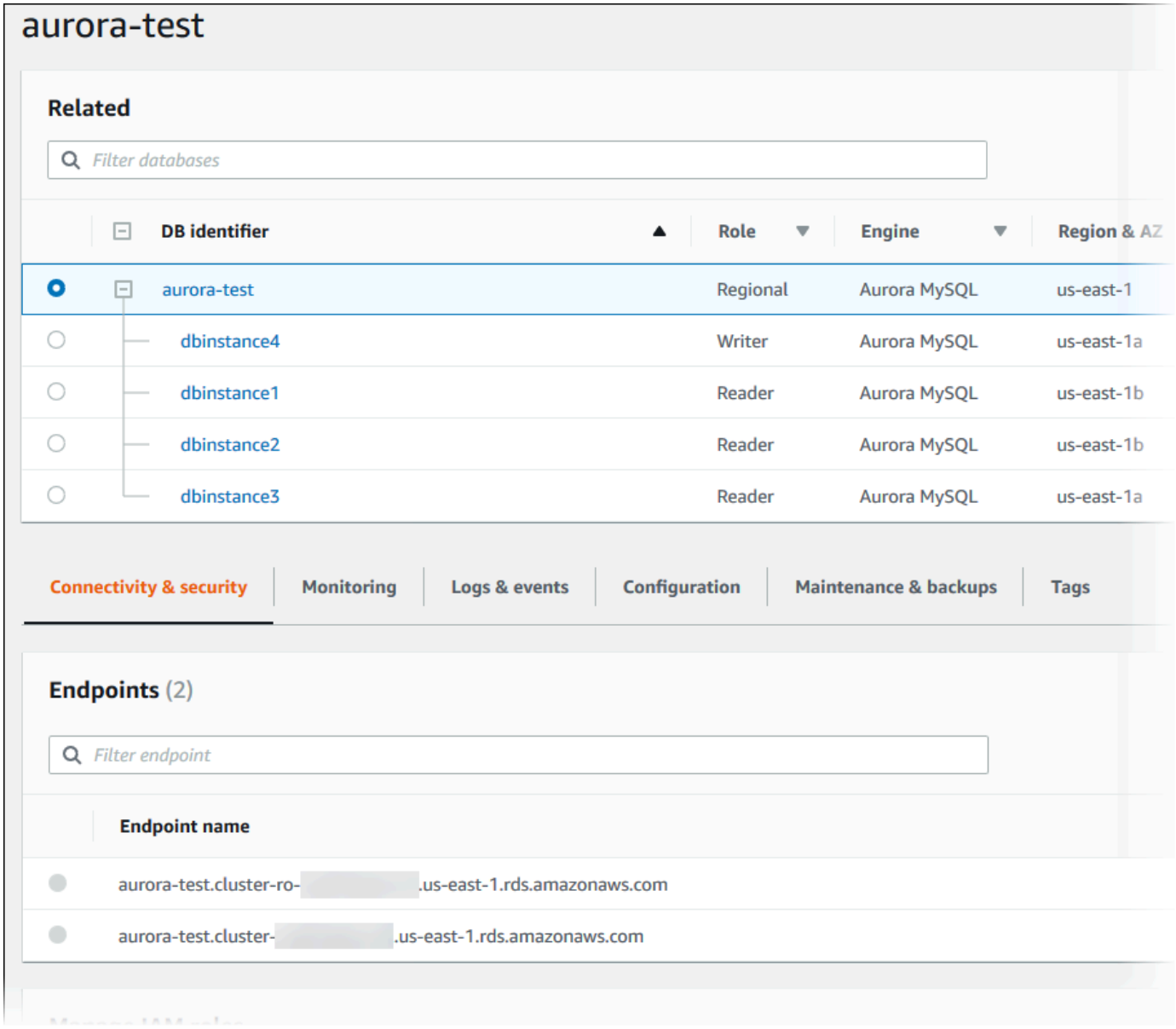
Konsol

Di konsol Amazon RDS, Anda dapat melihat detail tentang klaster DB dengan memilih Basis data dari panel navigasi konsol. Anda juga dapat melihat detail tentang instans DB yang merupakan anggota klaster DB Amazon Aurora DB.

Untuk melihat atau memodifikasi klaster DB di konsol Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama klaster DB Aurora yang ingin Anda lihat dari daftar.

   Misalnya, gambar berikut menunjukkan halaman detail untuk klaster DB bernama `aurora-test`. Klaster DB ini menampilkan empat instans DB dalam daftar pengidentifikasi DB. Instans DB penulis, yakni `dbinstance4`, merupakan instans DB utama untuk klaster DB.



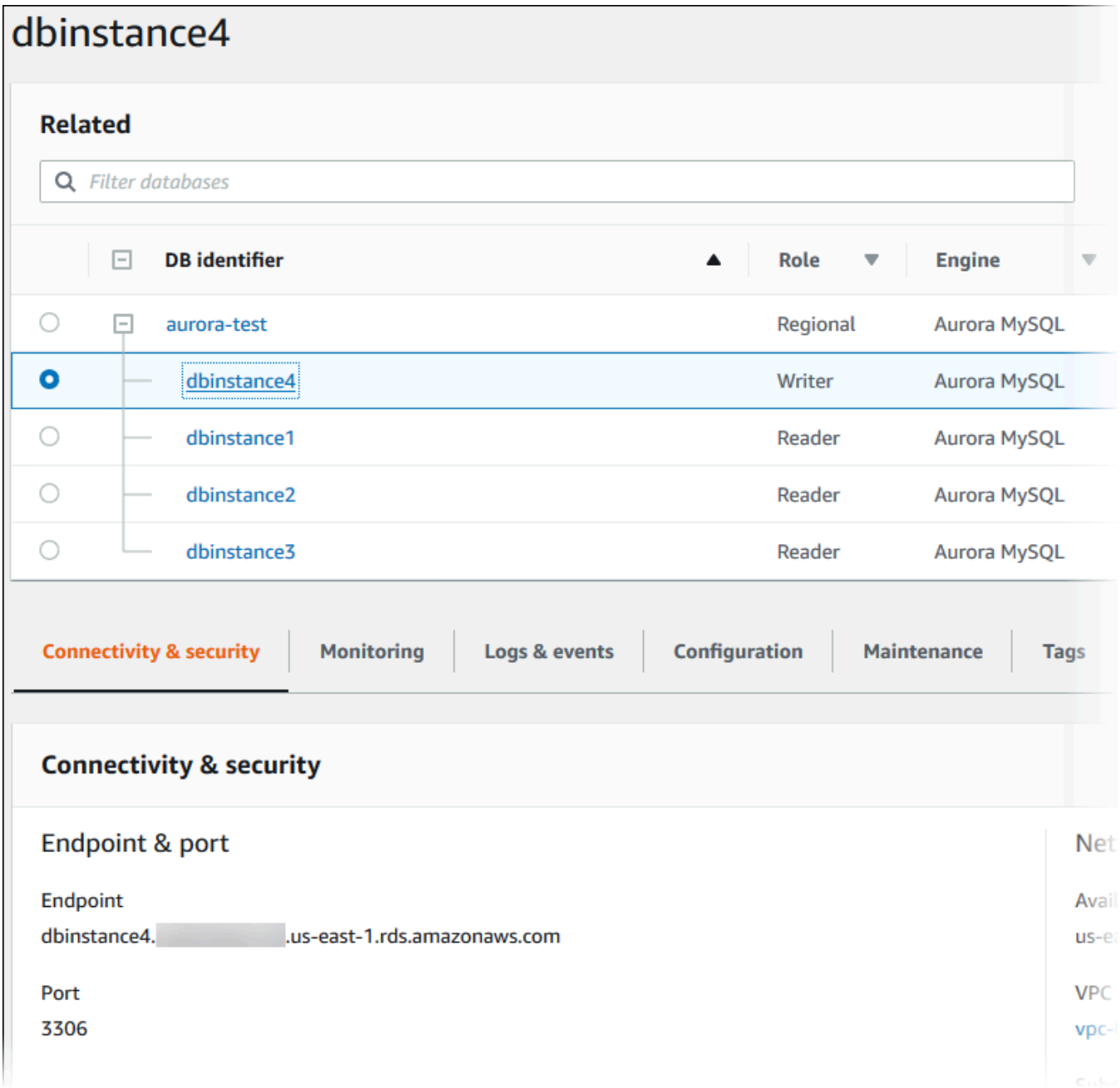

4. Untuk memodifikasi klaster DB, pilih klaster DB dari daftar dan pilih Modifikasi.

Untuk melihat atau memodifikasi instans DB klaster DB di konsol Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Lakukan salah satu hal berikut:

   - Untuk melihat instans DB, pilih salah satu dari daftar yang merupakan anggota klaster DB Aurora.

Misalnya, jika Anda memilih pengidentifikasi instans DB `dbinstance4`, konsol menampilkan halaman detail untuk instans DB `dbinstance4`, seperti yang ditampilkan di gambar berikut.




- Untuk memodifikasi instans DB, pilih instans DB dari daftar dan pilih Modifikasi. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

AWS CLI

Untuk melihat informasi cluster DB dengan menggunakan AWS CLI, gunakan describe-db-clustersperintah. Misalnya, AWS CLI perintah berikut mencantumkan informasi cluster DB untuk semua cluster DB di `us-east-1` wilayah modifikasi untuk AWS akun yang dikonfigurasi.

```
aws rds describe-db-clusters --region us-east-1
```

Perintah mengembalikan output berikut jika Anda AWS CLI dikonfigurasi untuk output JSON.

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
```

```
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
```

```
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

### API RDS

Untuk melihat informasi cluster DB menggunakan Amazon RDS API, gunakan DBClusters operasi Deskripsikan.

# Melihat status klaster DB

Status klaster DB menunjukkan kondisinya. Anda dapat melihat status cluster DB dan instance cluster dengan menggunakan konsol Amazon RDS, the AWS CLI, atau API.

> **Note**
>
> Aurora juga menggunakan status lain yang disebut status pemeliharaan, yang ditunjukkan di kolom Pemeliharaan pada konsol Amazon RDS. Nilai ini menunjukkan status patch pemeliharaan yang perlu diterapkan ke klaster DB. Status pemeliharaan bergantung pada status klaster DB. Untuk informasi lebih lanjut tentang status pemeliharaan, lihat Menerapkan pembaruan ke cluster DB.

Temukan kemungkinan nilai status untuk klaster DB dalam tabel berikut.

| Status klaster DB | Ditagih | Deskripsi |
|---|---|---|
| Available | Ditagih | Klaster DB berkondisi baik dan tersedia. Ketika klaster Nirserver Aurora tersedia dan dijeda, Anda akan ditagih hanya untuk penyimpanan. |
| Backing-up | Ditagih | Klaster DB saat ini sedang dicadangkan. |
| Backtracking | Ditagih | Backtrack sedang dilakukan untuk klaster DB. Status ini hanya berlaku untuk Aurora MySQL. |
| Cloning-failed | Tidak ditagih | Klaster DB gagal dikloning. |
| Creating | Tidak ditagih | Klaster DB sedang dibuat. Klaster DB tidak dapat diakses saat sedang dibuat. |
| Deleting | Tidak ditagih | Klaster DB sedang dihapus. |
| Failing-over | Ditagih | Failover dari instans utama ke Aurora Replica sedang dilakukan. |

| Status klaster DB | Ditagih | Deskripsi |
| --- | --- | --- |
| Aku naccessible-encryption-credentials | Tidak ditagih | Yang AWS KMS key digunakan untuk mengenkripsi atau mendekripsi cluster DB tidak dapat diakses atau dipulihkan. |
| Aku naccessible-encryption-cred entials-recoverable | Ditagih untuk penyimpanan | Kunci KMS digunakan untuk mengenkripsi atau mendekripsi klaster DB tidak dapat diakses. Namun, jika kunci KMS aktif, Anda dapat memulai ulang klaster DB untuk memulihka nnya.<br><br>Untuk informasi selengkapnya, lihat Mengenkripsi klaster DB Amazon Aurora. |
| Maintenance | Ditagih | Amazon RDS menerapkan pembaruan pemeliharaan pada klaster DB. Status ini digunakan untuk pemeliharaan tingkat klaster DB yang dijadwalkan RDS sejak jauh hari sebelumnya. |
| Migrating | Ditagih | Snapshot klaster DB sedang dipulihkan ke klaster DB. |
| Migration-failed | Tidak ditagih | Migrasi gagal. |
| Modifying | Ditagih | Klaster DB sedang dimodifikasi karena ada permintaan pelanggan untuk memodifikasi klaster DB. |
| Promoting | Ditagih | Replika baca sedang dipromosikan ke klaster DB mandiri. |
| P reparing-data-migration | Ditagih | Amazon RDS sedang bersiap untuk memigrasikan data ke Aurora. |
| Renaming | Ditagih | Nama klaster DB sedang diganti karena ada permintaan pelanggan untuk mengganti namanya. |

| Status klaster DB | Ditagih | Deskripsi |
| --- | --- | --- |
| R esetting-master-cr edentials | Ditagih | Kredensial master untuk klaster DB sedang direset karena ada permintaan pelanggan untuk meresetnya. |
| Starting | Ditagih untuk penyimpanan | Klaster DB dimulai. |
| Stopped | Ditagih untuk penyimpanan | Klaster DB dihentikan. |
| Stopping | Ditagih untuk penyimpanan | Klaster DB sedang dihentikan. |
| Storage-optimization | Ditagih | Instans DB Anda sedang dimodifikasi untuk mengubah ukuran atau jenis penyimpanan. Instans DB berfungsi sepenuhnya. Namun, meskipun status instans DB Anda adalah storage-optimization, Anda tidak dapat meminta perubahan apa pun untuk penyimpanan instans DB Anda. Proses pengoptimalan penyimpan an biasanya singkat, tetapi terkadang dapat memakan waktu hingga dan bahkan lebih dari 24 jam. |
| U pdate-iam-db-auth | Ditagih | Otorisasi IAM untuk klaster DB sedang diperbarui. |
| Meningkatkan | Ditagih | Mesin cluster DB atau versi sistem operasi sedang ditingkatkan. |

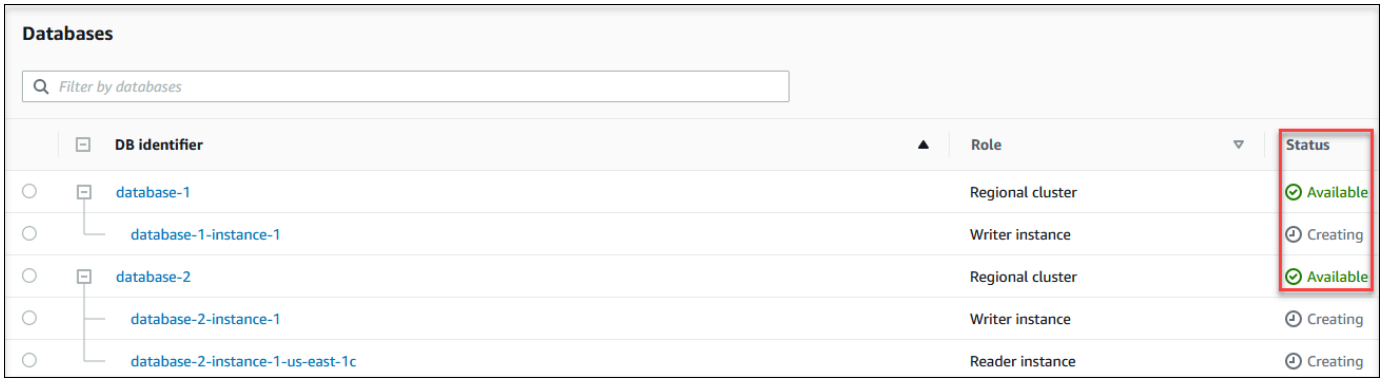
Konsol

Untuk melihat status klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Basis data.

   Halaman Basis Data muncul dengan daftar klaster DB. Nilai status untuk setiap klaster DB akan ditampilkan.




CLI

Untuk melihat hanya status cluster DB, gunakan kueri berikut di AWS CLI.

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

# Melihat status instans DB di klaster Aurora

Status instans DB dalam klaster Aurora menunjukkan kondisi instans DB. Anda dapat menggunakan prosedur berikut untuk melihat status instans DB klaster di konsol Amazon RDS, AWS CLI perintah, atau operasi API.

> **Note**
>
> Amazon RDS juga menggunakan status lain yang disebut status pemeliharaan, yang ditunjukkan di kolom Pemeliharaan pada konsol Amazon RDS. Nilai ini menunjukkan status patch pemeliharaan yang perlu diterapkan ke instans DB. Status pemeliharaan bergantung pada status instans DB. Untuk informasi lebih lanjut tentang status pemeliharaan, lihat Menerapkan pembaruan ke cluster DB.

Temukan kemungkinan nilai status untuk instans DB dalam tabel berikut. Tabel ini juga menunjukkan apakah Anda akan ditagih untuk instans DB dan penyimpanan, ditagih hanya untuk penyimpanan, atau Anda tidak dikenai tagihan. Untuk semua status instans DB, Anda selalu ditagih untuk penggunaan cadangan.

| Status instans DB | Ditagih | Deskripsi |
|---|---|---|
| tersedia | Ditagih | Instans DB berkondisi baik dan tersedia. |
| cadangan | Ditagih | Instans DB saat ini sedang dicadangkan. |
| mundur | Ditagih | Backtrack sedang dilakukan untuk instans DB. Status ini hanya berlaku untuk Aurora MySQL. |
| configuring-enhanced-monitoring | Ditagih | Pemantauan yang ditingkatkan diaktifkan atau dinonaktifkan untuk instans DB ini. |
| configuring-iam-database-auth | Ditagih | AWS Identity and Access Management (IAM) otentikasi database sedang diaktifkan atau dinonaktifkan untuk instans DB ini. |
| configuring-log-exports | Ditagih | Menerbitkan file log ke Amazon CloudWatch Logs sedang diaktifkan atau dinonaktifkan untuk instans DB ini. |

| Status instans DB | Ditagih | Deskripsi |
| --- | --- | --- |
| converting-to-vpc | Ditagih | Instans DB sedang dikonversi dari instans DB yang tidak ada di Amazon Virtual Private Cloud (Amazon VPC) menjadi instans DB yang ada di Amazon VPC. |
| menciptakan | Tidak ditagih (non-PITR)<br><br>Ditagih (hanya PITR) | Instans DB sedang dibuat. Instans DB tidak dapat diakses saat sedang dibuat.<br><br>Jika Anda memulihkan database selama point-in-time pemulihan (PITR), Anda ditagih saat database dalam status pembuatan. Ini adalah satu-satunya skenario di mana status pembuat menimbulkan biaya. |
| hapus-precheck | Tidak ditagih | Amazon RDS memvalidasi bahwa replika baca berkondisi baik dan aman untuk dihapus. |
| menghapus | Tidak ditagih | Instans DB sedang dihapus. |
| gagal | Tidak ditagih | Instans DB gagal dan Amazon RDS tidak dapat dipulihkan. Lakukan point-in-time pemulihan ke waktu restorable terbaru dari instans DB untuk memulihkan data. |
| inaccessible-encryption-credentials | Tidak ditagih | Yang AWS KMS key digunakan untuk mengenkripsi atau mendekripsi instans DB tidak dapat diakses atau dipulihkan. |
| inaccessible-encryption-credentials-recoverable | Ditagih untuk penyimpanan | Kunci KMS digunakan untuk mengenkripsi atau mendekripsi instans DB tidak dapat diakses. Namun, jika kunci KMS aktif, Anda dapat memulai ulang instans DB untuk memulihkannya.<br><br>Untuk informasi selengkapnya, lihat Mengenkripsi klaster DB Amazon Aurora. |

| Status instans DB | Ditagih | Deskripsi |
| --- | --- | --- |
| tidak kompatibel-buat | Tidak ditagih | Amazon RDS mencoba membuat instans DB tetapi tidak dapat melakukannya karena sumber daya tidak kompatibel dengan instans DB Anda. Status ini dapat terjadi jika, misalnya, profil instans untuk instans DB Anda tidak memiliki izin yang benar. |
| jaringan yang tidak kompatibel | Tidak ditagih | Amazon RDS mencoba melakukan tindakan pemulihan di instans DB tetapi tidak dapat melakukannya karena VPC berada dalam keadaan yang mencegah penyelesaian tindakan. Status ini dapat terjadi jika, misalnya, semua alamat IP yang tersedia di subnet sedang digunakan dan Amazon RDS tidak bisa mendapatkan alamat IP untuk instans DB. |
| incompatible-option-group | Ditagih | Amazon RDS mencoba menerapkan perubahan grup opsi tetapi tidak dapat melakukannya, dan Amazon RDS tidak dapat melakukan roll back ke status grup opsi sebelumnya. Untuk informasi selengkap nya, lihat daftar Peristiwa Terbaru untuk instans DB. Status ini dapat terjadi jika, misalnya, grup opsi berisi opsi seperti TDE dan instans DB tidak berisi informasi terenkripsi. |
| parameter yang tidak kompatibel | Ditagih | Amazon RDS tidak dapat memulai instans DB karena parameter yang ditentukan dalam grup parameter DB instans DB tidak kompatibel dengan instans DB. Kembalikan perubahan parameter atau buat parameter kompatibel dengan instans DB untuk mendapatkan akses ke instans DB Anda. Untuk informasi selengkapnya tentang parameter yang tidak kompatibel, periksa daftar Peristiwa Terbaru untuk instans DB. |

| Status instans DB | Ditagih | Deskripsi |
| --- | --- | --- |
| pemulihan yang tidak kompatibel | Tidak ditagih | Amazon RDS tidak dapat melakukan point-in-time pemulihan. Penyebab umum untuk status ini termasuk penggunaan tabel sementara atau penggunaan tabel MyISAM dengan MySQL. |
| ketidakcukupan kapasitas | Tidak ditagih | Amazon RDS tidak dapat membuat instans Anda karena kapasitas yang cukup saat ini tidak tersedia. Untuk membuat instans DB di AZ yang sama dengan jenis instans yang sama, hapus instans DB Anda, tunggu beberapa jam, lalu coba untuk membuat lagi. Atau, buat instans baru menggunakan kelas instans atau AZ yang berbeda. |
| pemeliharaan | Ditagih | Amazon RDS menerapkan pembaruan pemeliharaan pada instans DB. Status ini digunakan untuk pemeliharaan tingkat instans yang dijadwalkan RDS sejak jauh hari sebelumnya. |
| memodifikasi | Ditagih | Instans DB sedang dimodifikasi karena ada permintaan pelanggan untuk memodifikasi instans DB. |
| moving-to-vpc | Ditagih | Instans DB sedang dipindahkan ke Amazon Virtual Private Cloud (Amazon VPC) baru. |
| me-reboot | Ditagih | Instans DB sedang di-boot ulang karena permintaan pelanggan atau proses Amazon RDS yang memerlukan boot ulang instans DB. |
| resetting-master-credentials | Ditagih | Kredensial master untuk instans DB sedang direset karena ada permintaan pelanggan untuk meresetnya. |
| mengganti nama | Ditagih | Nama instans DB sedang diganti karena ada permintaan pelanggan untuk mengganti namanya. |

| Status instans DB | Ditagih | Deskripsi |
| --- | --- | --- |
| memulihkan-kesalahan | Ditagih | Instans DB mengalami kesalahan saat mencoba mengembalikan ke point-in-time atau dari snapshot. |
| memulai | Ditagih untuk penyimpanan | Instans DB dimulai. |
| berhenti | Ditagih untuk penyimpanan | Instans DB dihentikan. |
| berhenti | Ditagih untuk penyimpanan | Instans DB sedang dihentikan. |
| storage-config-upgrade | Ditagih | Konfigurasi sistem file penyimpanan instans DB sedang ditingkatkan. Status ini hanya berlaku untuk basis data green dalam deployment blue/green, atau untuk replika baca instans DB. |
| penyimpanan-penuh | Ditagih | Instans DB telah mencapai alokasi kapasitas penyimpanannya. Ini merupakan status kritis. Sebaiknya segera perbaiki masalah ini. Untuk melakukannya, perbesar penyimpanan Anda dengan memodifikasi instans DB. Untuk menghindari situasi ini, setel CloudWatch alarm Amazon untuk memperingatkan Anda saat ruang penyimpanan semakin rendah. |
| penyimpanan-inisialisasi | Ditagih | Instans DB memuat blok data dari Amazon S3 untuk mengoptimalkan kinerja volume setelah dipulihkan dari snapshot. Ini tetap tersedia untuk operasi, tetapi kinerja mungkin tidak sepenuhnya sampai inisialisasi selesai. |

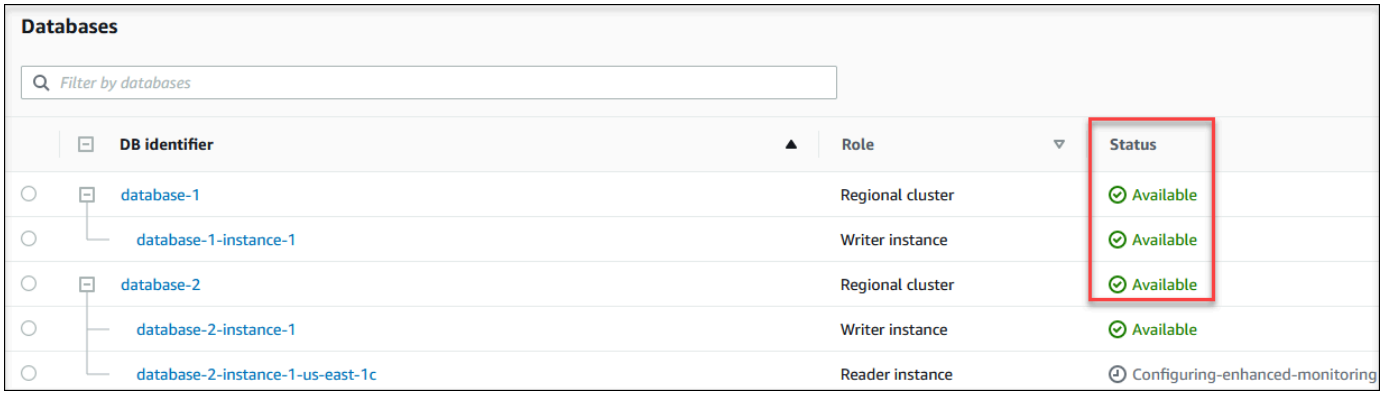
| Status instans DB | Ditagih | Deskripsi |
| --- | --- | --- |
| pengoptimalan penyimpanan | Ditagih | Amazon RDS mengoptimalkan penyimpanan instans DB Anda. Proses pengoptimalan penyimpanan biasanya singkat, tetapi terkadang dapat memakan waktu hingga lebih dari 24 jam.<br><br>Selama optimasi penyimpanan, instans DB tetap tersedia. Pengoptimalan penyimpanan adalah proses latar belakang yang tidak memengaruhi ketersediaan instans. |
| memutakhirkan | Ditagih | Mesin database atau versi sistem operasi sedang ditingkatkan. |

Konsol

Untuk melihat status instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.

   Halaman Basis Data muncul dengan daftar instans DB. Nilai status setiap instans DB dalam klaster akan ditampilkan.



CLI

Untuk melihat instans DB dan informasi statusnya dengan menggunakan AWS CLI, gunakan describe-db-instancesperintah. Misalnya, AWS CLI perintah berikut mencantumkan semua informasi instance DB.

```
aws rds describe-db-instances
```

Untuk melihat instans DB tertentu dan statusnya, panggil describe-db-instancesperintah dengan opsi berikut:

- `DBInstanceIdentifier` – Nama instans DB.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

Untuk melihat status semua instans DB, gunakan kueri berikut di AWS CLI.

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

Untuk melihat status instans DB menggunakan Amazon RDS API, panggil DBInstances operasi Deskripsikan.

Amazon Aurora memberikan rekomendasi otomatis untuk sumber daya database, seperti instans DB, cluster DB, , dan grup parameter DB. Rekomendasi ini memberikan panduan praktik terbaik dengan menganalisis konfigurasi klaster DB, konfigurasi, penggunaan, dan data performa instans DB.

Amazon RDS Performance Insights memantau metrik tertentu dan secara otomatis membuat ambang batas dengan menganalisis level apa yang dianggap berpotensi bermasalah untuk sumber daya tertentu. Ketika nilai metrik baru melewati ambang batas yang telah ditentukan selama periode waktu tertentu, Performance Insights menghasilkan rekomendasi proaktif. Rekomendasi ini membantu mencegah dampak kinerja database future. Misalnya, rekomendasi “Idle In Transaction” dibuat ketika sesi yang terhubung ke database tidak melakukan pekerjaan aktif, tetapi dapat membuat sumber daya database diblokir. Untuk menerima rekomendasi proaktif, Anda harus mengaktifkan Performance Insights dengan periode retensi tingkat berbayar. Untuk informasi tentang mengaktifkan Performance Insights, lihat. Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora Untuk informasi tentang harga dan retensi data untuk Performance Insights, lihat. Harga dan retensi data untuk Wawasan Performa

DevOpsGuru untuk RDS memantau metrik tertentu untuk mendeteksi kapan perilaku metrik menjadi sangat tidak biasa atau anomali. Anomali ini dilaporkan sebagai wawasan reaktif dengan rekomendasi. Misalnya, DevOps Guru for RDS mungkin menyarankan Anda untuk mempertimbangkan peningkatan CPU kapasitas atau menyelidiki peristiwa menunggu yang berkontribusi terhadap beban DB. DevOpsGuru untuk RDS juga memberikan rekomendasi proaktif berbasis ambang batas. Untuk rekomendasi ini, Anda harus mengaktifkan DevOps Guru untukRDS. Untuk informasi tentang mengaktifkan DevOps GuruRDS, lihatMengaktifkan DevOps Guru dan menentukan cakupan sumber daya.

Rekomendasi akan berada dalam salah satu status berikut: aktif, diberhentikan, tertunda, atau diselesaikan. Rekomendasi yang diselesaikan tersedia selama 365 hari.

Anda dapat melihat atau mengabaikan rekomendasi. Anda dapat segera menerapkan rekomendasi aktif berbasis konfigurasi, menjadwalkannya di jendela pemeliharaan berikutnya, atau mengabaikannya. Untuk rekomendasi reaktif berbasis proaktif dan pembelajaran mesin berbasis ambang batas, Anda perlu meninjau penyebab masalah yang disarankan dan kemudian melakukan tindakan yang disarankan untuk memperbaiki masalah.

Rekomendasi didukung sebagai berikut Wilayah AWS:

- AS Timur (Ohio)
- AS Timur (Virginia Utara)

- AS Barat (California Utara)
- AS Barat (Oregon)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- Kanada (Pusat)
- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)
- Eropa (Stockholm)
- Amerika Selatan (Sao Paulo)

Pelajari cara melihat, menerapkan, mengabaikan, dan memodifikasi rekomendasi dari Aurora di bagian berikut.

Topik

- Melihat rekomendasi Amazon Aurora
- Menerapkan Amazon Amazon Aurora
- Mengabaikan rekomendasi Aurora Amazon
- Memodifikasi rekomendasi Amazon RDS Aurora Aurora yang diberhentikan menjadi rekomendasi aktif
- Rekomendasi dari Amazon Aurora

# Melihat rekomendasi Amazon Aurora

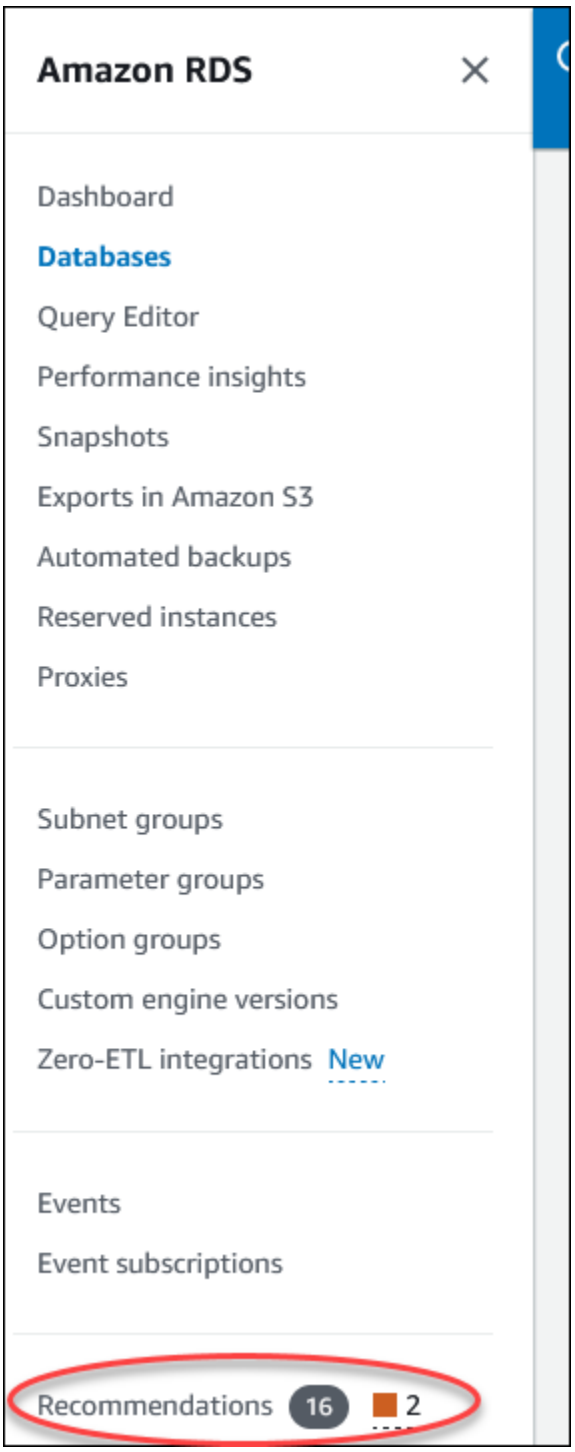
Menggunakan RDS konsol Amazon, Anda dapat melihat rekomendasi Aurora untuk sumber daya basis data Anda.  Untuk cluster DB, rekomendasi muncul untuk cluster DB dan instance-nya.

Konsol

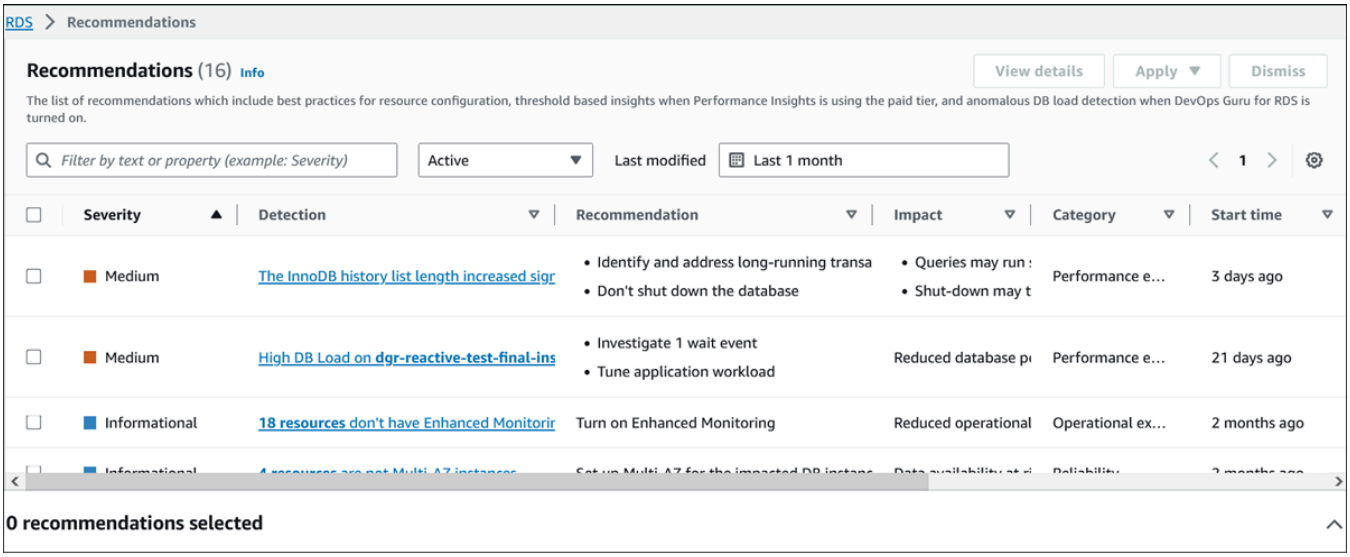
Untuk melihat rekomendasi Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi. Jumlah rekomendasi aktif untuk sumber daya Anda dan jumlah rekomendasi dengan tingkat keparahan tertinggi yang dihasilkan pada bulan lalu tersedia di sebelah Rekomendasi. Untuk menemukan jumlah rekomendasi aktif untuk setiap tingkat keparahan, pilih angka yang menunjukkan tingkat keparahan tertinggi.

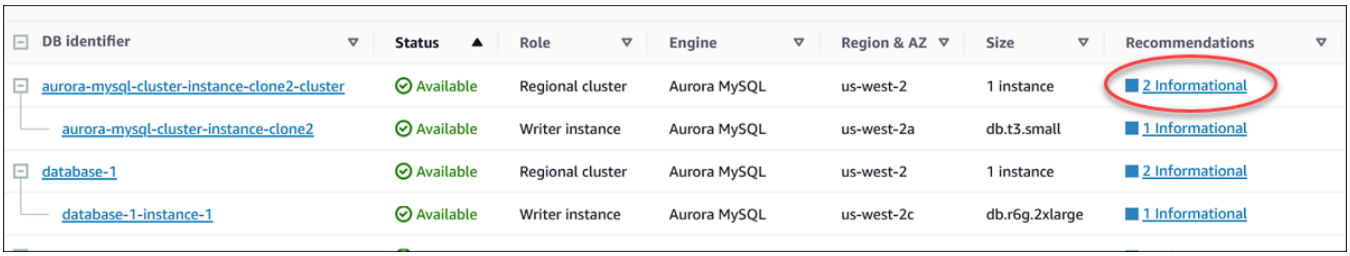



Secara default, halaman Rekomendasi menampilkan daftar rekomendasi baru dalam sebulan terakhir. Amazon Aurora memberikan rekomendasi untuk semua sumber daya di akun Anda dan mengurutkan rekomendasi berdasarkan tingkat keparahannya.

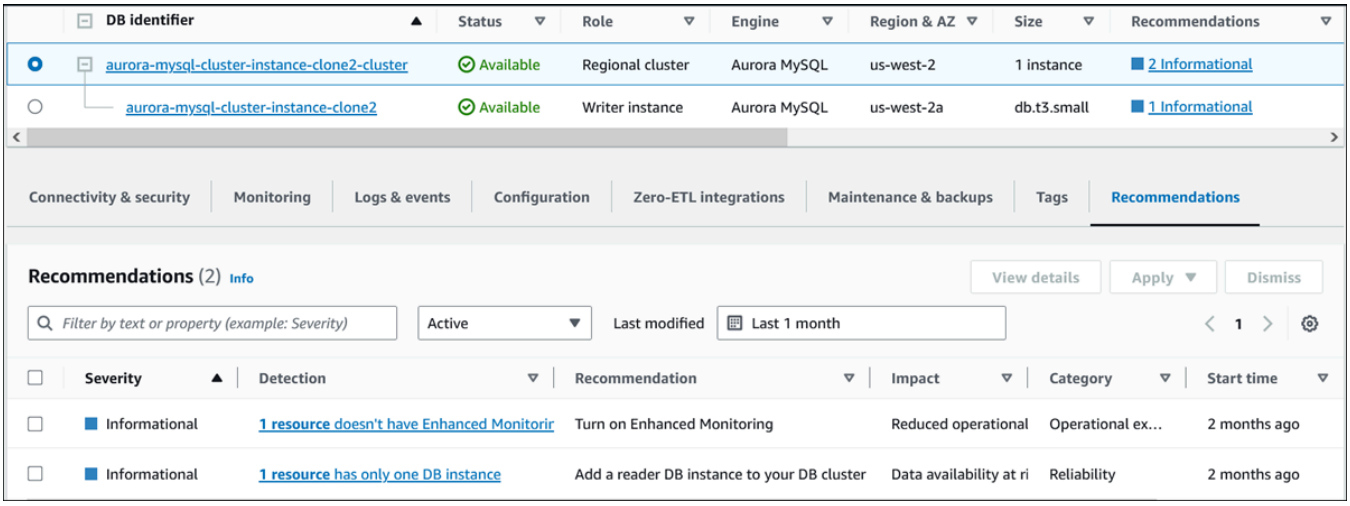



Anda dapat memilih rekomendasi untuk melihat bagian di bagian bawah halaman yang berisi sumber daya yang terpengaruh dan detail tentang bagaimana rekomendasi akan diterapkan.

- Di halaman Database, pilih Rekomendasi untuk sumber daya.



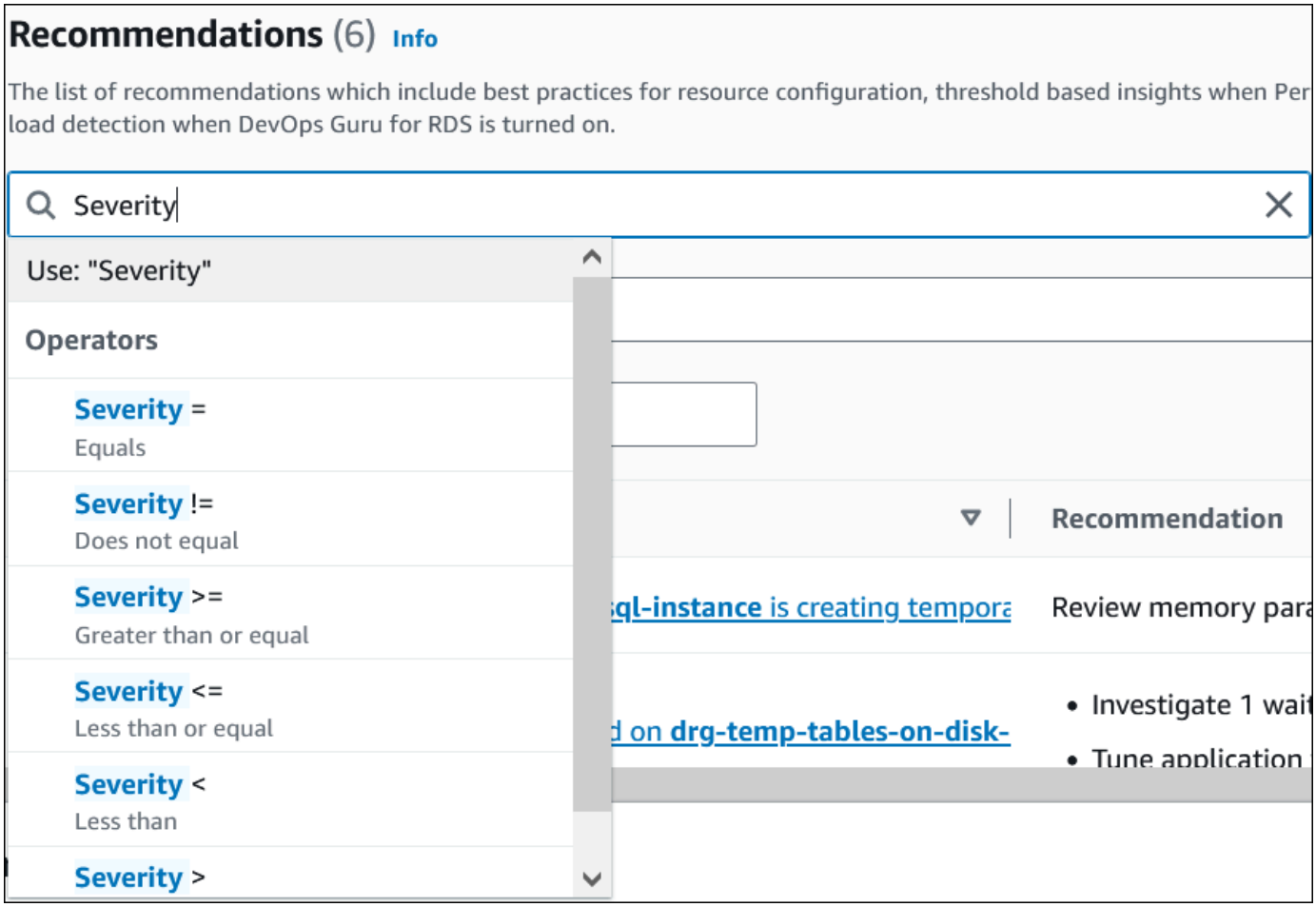

Tab Rekomendasi menampilkan rekomendasi dan detailnya untuk sumber daya yang dipilih.




Rincian berikut tersedia untuk rekomendasi:

- Keparahan — Tingkat implikasi dari masalah ini. Tingkat keparahannya adalah Tinggi, Sedang, Rendah, dan Informasi.
- Deteksi — Jumlah sumber daya yang terpengaruh dan deskripsi singkat tentang masalah ini. Pilih tautan ini untuk melihat rekomendasi dan detail analisis.
- Rekomendasi — Deskripsi singkat tentang tindakan yang disarankan untuk diterapkan.
- Dampak — Deskripsi singkat tentang kemungkinan dampak ketika rekomendasi tidak diterapkan.
- Kategori — Jenis rekomendasi. Kategori tersebut adalah Efisiensi kinerja, Keamanan, Keandalan, Optimalisasi biaya, keunggulan operasional, dan Keberlanjutan.
- Status — Status rekomendasi saat ini. Status yang mungkin adalah Semua, Aktif, Diberhentikan, Diselesaikan, dan Tertunda.
- Waktu mulai — Waktu ketika masalah dimulai. Misalnya, 18 jam yang lalu.
- Terakhir diubah - Waktu ketika rekomendasi terakhir diperbarui oleh sistem karena perubahan Tingkat Keparahan, atau waktu Anda menanggapi rekomendasi. Misalnya, 10 jam yang lalu.
- Waktu akhir - Waktu ketika masalah berakhir. Waktu tidak akan ditampilkan untuk masalah yang berkelanjutan.
- Pengenal sumber daya — Nama satu atau lebih sumber daya.

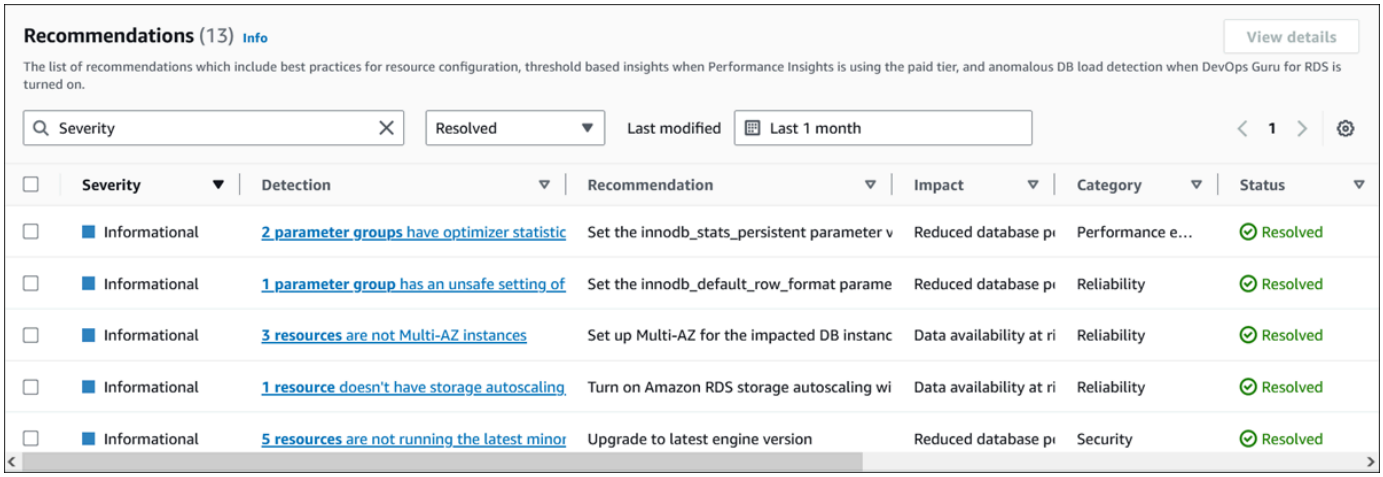
3. (Opsional) Pilih operator Keparahan atau Kategori di bidang untuk memfilter daftar rekomendasi.



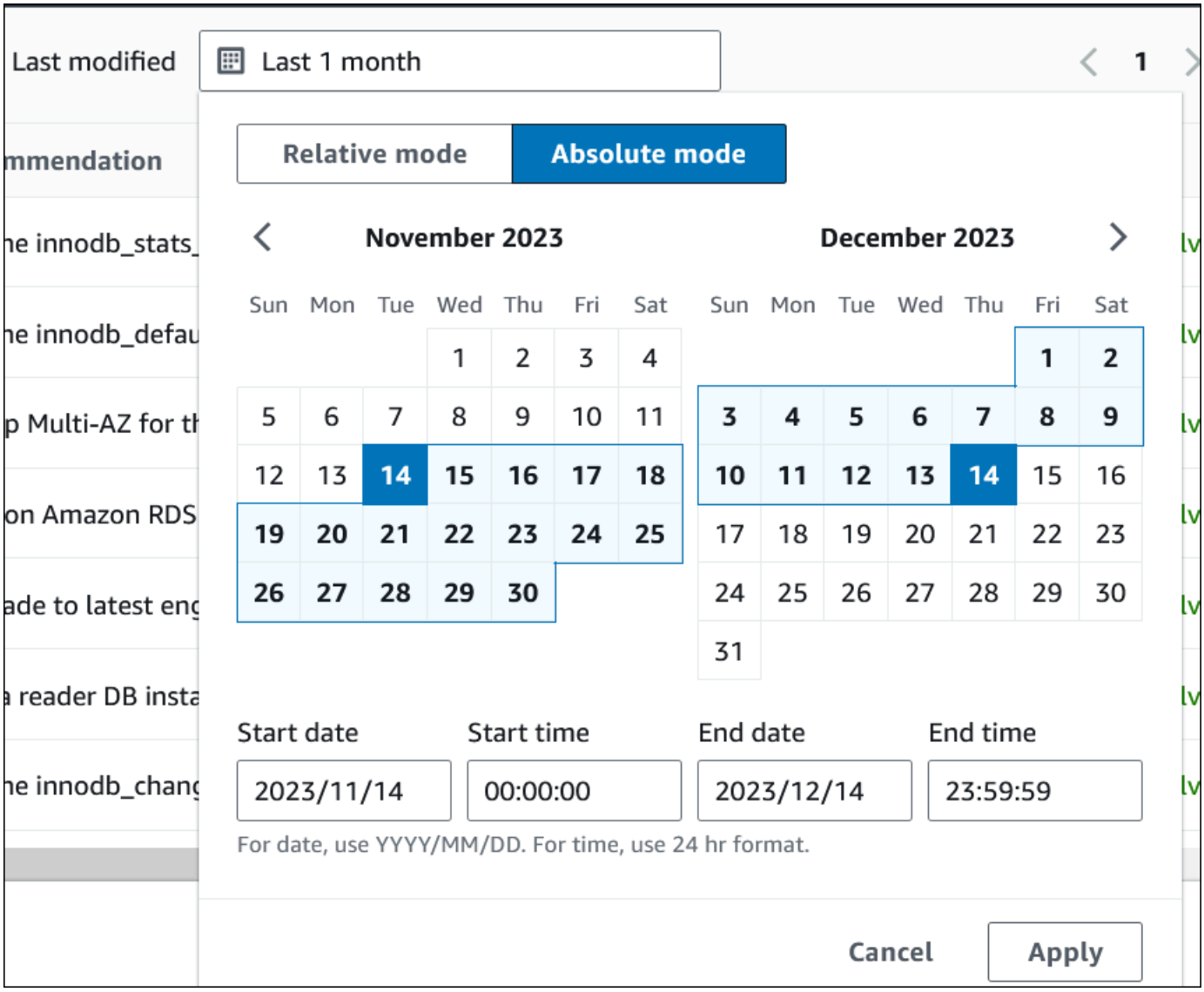

Rekomendasi untuk operasi yang dipilih muncul.

4. (Opsional) Pilih salah satu status rekomendasi berikut:

   - Aktif (default) - Menampilkan rekomendasi saat ini yang dapat Anda terapkan, menjadwalkannya untuk jendela pemeliharaan berikutnya, atau memberhentikan.
   - Semua - Menampilkan semua rekomendasi dengan status saat ini.
   - Diberhentikan — Menunjukkan rekomendasi yang diberhentikan.
   - Terselesaikan - Menunjukkan rekomendasi yang diselesaikan.
   - Tertunda — Menunjukkan rekomendasi yang tindakan rekomendasinya sedang berlangsung atau dijadwalkan untuk jendela pemeliharaan berikutnya.



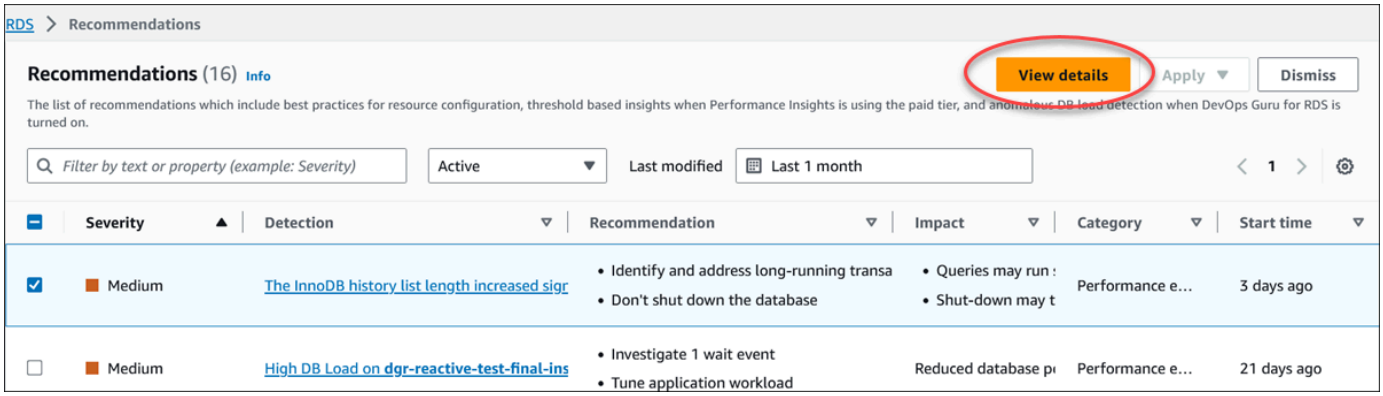

5. (Opsional) Pilih mode Relatif atau Mode absolut di Terakhir diubah untuk mengubah periode waktu. Halaman Rekomendasi menampilkan rekomendasi yang dihasilkan dalam periode waktu tersebut. Periode waktu default adalah bulan terakhir. Dalam mode Absolute, Anda dapat memilih periode waktu, atau memasukkan waktu di bidang Tanggal mulai dan Tanggal akhir.




Rekomendasi untuk tampilan periode waktu yang ditetapkan.

Perhatikan bahwa Anda dapat melihat semua rekomendasi untuk sumber daya di akun Anda dengan menyetel rentang ke Semua.

6. (Opsional) Pilih Preferensi di sebelah kanan untuk menyesuaikan detail yang akan ditampilkan. Anda dapat memilih ukuran halaman, membungkus baris teks, dan mengizinkan atau menyembunyikan kolom.
7. (Opsional) Pilih rekomendasi dan kemudian pilih Lihat detail.



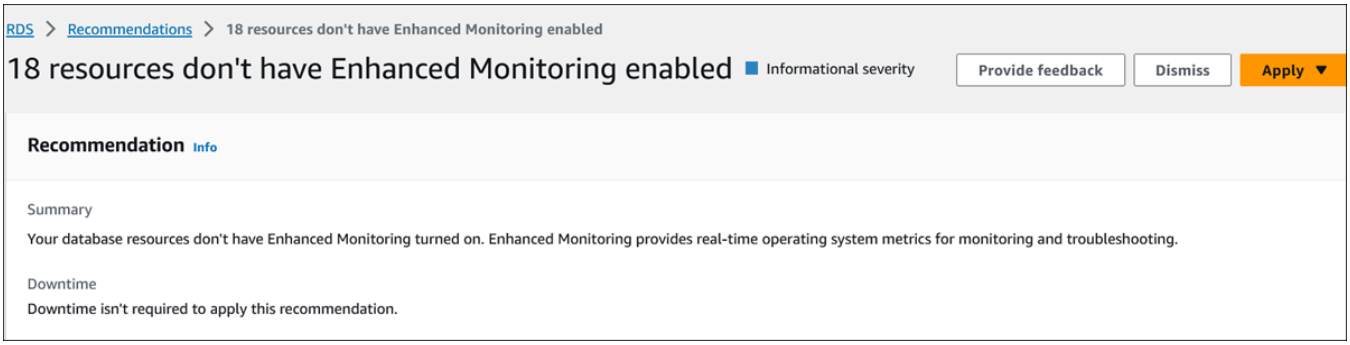

Halaman detail rekomendasi muncul. Judul memberikan jumlah total sumber daya dengan masalah yang terdeteksi dan tingkat keparahannya.

Untuk informasi tentang komponen di halaman detail untuk rekomendasi reaktif berbasis anomali, lihat Melihat anomali reaktif di Panduan Pengguna Amazon Guru. DevOps

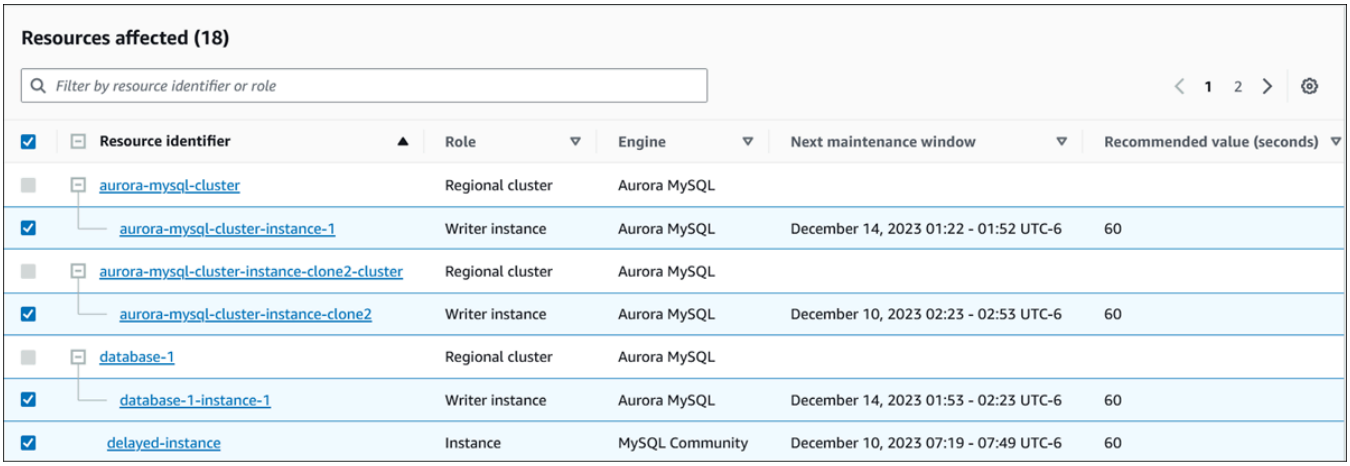
Untuk informasi tentang komponen pada halaman detail untuk rekomendasi proaktif berbasis ambang batas, lihatMelihat rekomendasi proaktif Performance Insights.

Rekomendasi otomatis lainnya menampilkan komponen berikut di halaman detail rekomendasi:

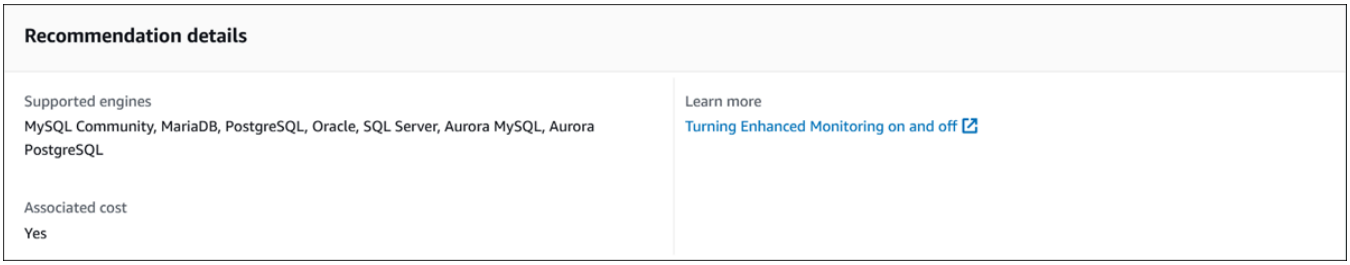
- Rekomendasi — Ringkasan rekomendasi dan apakah downtime diperlukan untuk menerapkan rekomendasi.




- Sumber daya yang terpengaruh — Detail sumber daya yang terpengaruh.




- Detail rekomendasi — Informasi mesin yang didukung, biaya terkait yang diperlukan untuk menerapkan rekomendasi, dan tautan dokumentasi untuk mempelajari lebih lanjut.




CLI

Untuk melihat RDS rekomendasi Amazon dari instans DB atau kluster DB, gunakan perintah berikut di. AWS CLI

```
aws rds describe-db-recommendations
```

RDS API

Untuk melihat RDS rekomendasi Amazon menggunakan Amazon RDSAPI, gunakan escribeDBRecommendations operasi D.

## Menerapkan Amazon Amazon Aurora

Untuk menerapkan rekomendasi Amazon Aurora menggunakan RDS konsol Amazon, pilih rekomendasi berbasis konfigurasi atau sumber daya yang terpengaruh di halaman detail. Kemudian, pilih untuk segera menerapkan rekomendasi atau jadwalkan untuk jendela pemeliharaan berikutnya. Sumber daya mungkin perlu dimulai ulang agar perubahan diterapkan. Untuk beberapa rekomendasi grup parameter DB, Anda mungkin perlu memulai ulang sumber daya.

Rekomendasi reaktif berbasis proaktif atau anomali berbasis ambang batas tidak akan memiliki opsi penerapan dan mungkin memerlukan tinjauan tambahan.

Konsol

Untuk menerapkan rekomendasi berbasis konfigurasi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi.

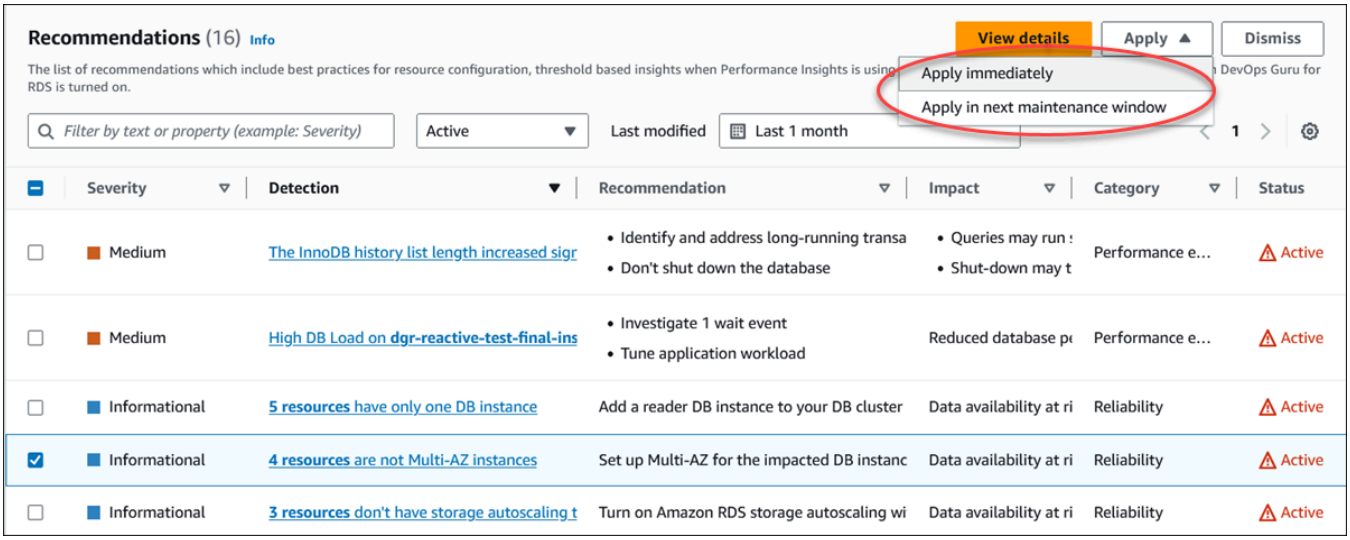
     Halaman Rekomendasi muncul dengan daftar semua rekomendasi.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Detailnya muncul di tab Rekomendasi untuk rekomendasi yang dipilih.
   - Pilih Deteksi untuk rekomendasi aktif di halaman Rekomendasi atau tab Rekomendasi di halaman Database.

     Halaman detail rekomendasi muncul.
3. Pilih rekomendasi, atau satu atau beberapa sumber daya yang terpengaruh di halaman detail rekomendasi, dan lakukan salah satu hal berikut:

   - Pilih Terapkan dan kemudian pilih Terapkan segera untuk segera menerapkan rekomendasi.
   - Pilih Terapkan dan kemudian pilih Terapkan di jendela pemeliharaan berikutnya untuk menjadwalkan di jendela pemeliharaan berikutnya.

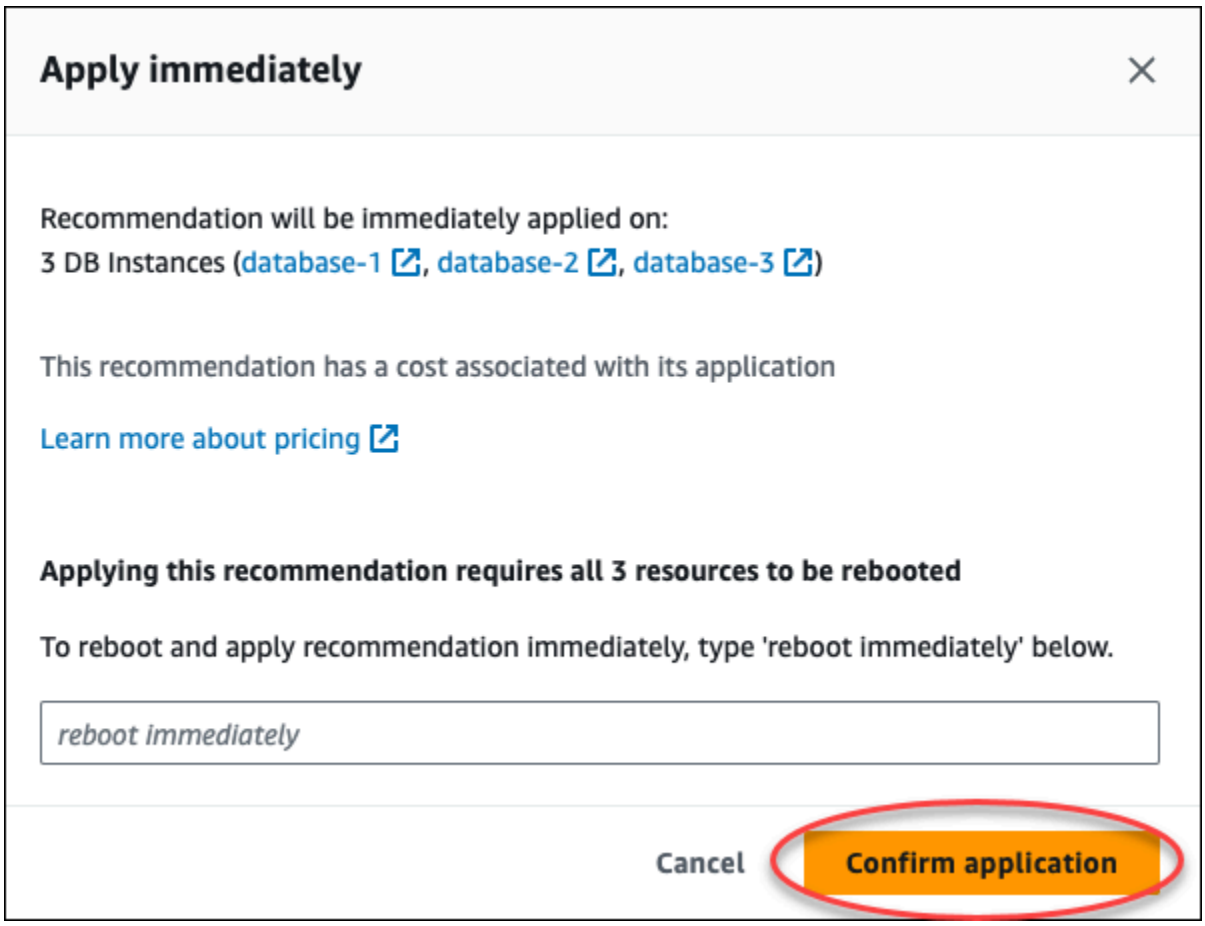
     Status rekomendasi yang dipilih diperbarui ke pending hingga jendela pemeliharaan berikutnya.



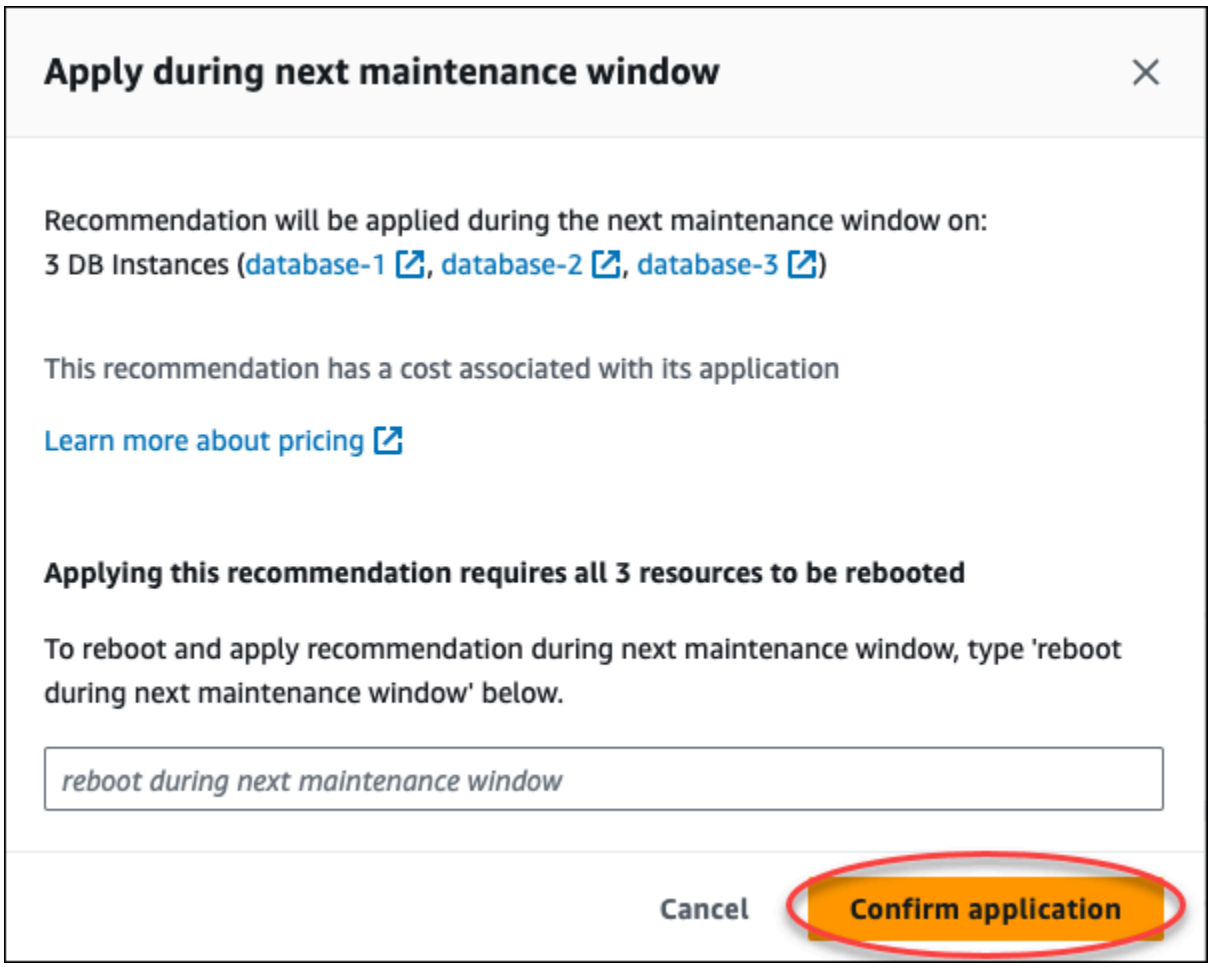

Jendela konfirmasi muncul.

4. Pilih Konfirmasi aplikasi untuk menerapkan rekomendasi. Jendela ini mengonfirmasi apakah sumber daya memerlukan restart otomatis atau manual agar perubahan diterapkan.

   Contoh berikut menunjukkan jendela konfirmasi untuk segera menerapkan rekomendasi.

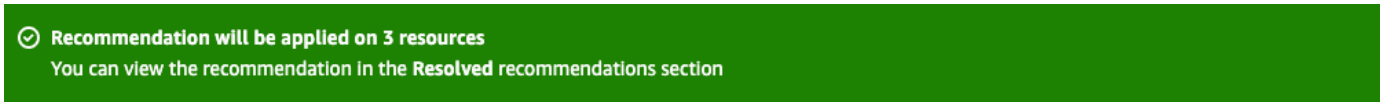



Contoh berikut menunjukkan jendela konfirmasi untuk menjadwalkan penerapan rekomendasi di jendela pemeliharaan berikutnya.




Spanduk menampilkan pesan ketika rekomendasi yang diterapkan berhasil atau gagal.

Contoh berikut menunjukkan spanduk dengan pesan yang berhasil.




Contoh berikut menunjukkan spanduk dengan pesan kegagalan.

### RDS API

Untuk menerapkan rekomendasi Aurora berbasis konfigurasi menggunakan Amazon RDS API

1. Gunakan escribeDBRecommendations operasi D. `RecommendedActions`Dalam output dapat memiliki satu atau lebih tindakan yang direkomendasikan.
2. Gunakan RecommendedActionobjek untuk setiap tindakan yang direkomendasikan dari langkah 1. Outputnya berisi `Operation` dan`Parameters`.

   Contoh berikut menunjukkan output dengan satu tindakan yang direkomendasikan.

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. Gunakan `operation` untuk setiap tindakan yang direkomendasikan dari output pada langkah 2 dan masukkan `Parameters` nilainya.
4. Setelah operasi pada langkah 2 berhasil, gunakan odifyDBRecommendation operasi M untuk mengubah status rekomendasi.

# Mengabaikan rekomendasi Aurora Amazon

Anda dapat mengabaikan satu atau lebih rekomendasi Aurora Amazon menggunakan konsol RDS Amazon AWS CLI,, atau Amazon. RDS API

Konsol

Untuk mengabaikan satu atau lebih rekomendasi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

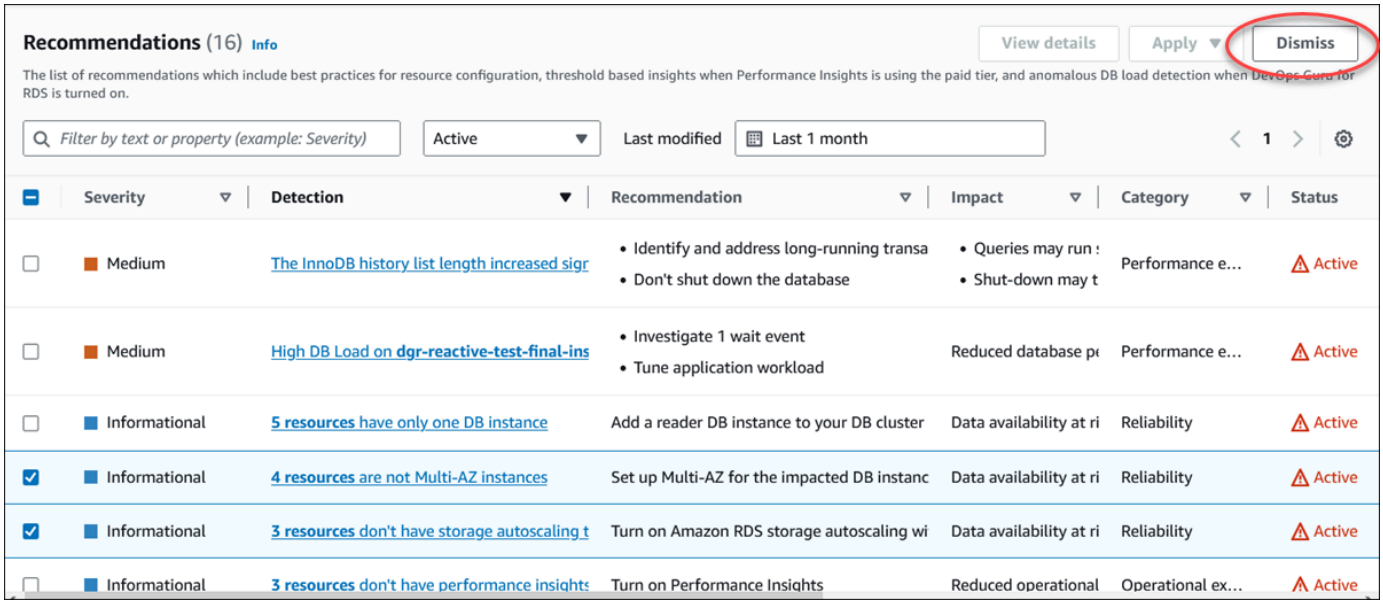
   - Pilih Rekomendasi.

     Halaman Rekomendasi muncul dengan daftar semua rekomendasi.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Detailnya muncul di tab Rekomendasi untuk rekomendasi yang dipilih.
   - Pilih Deteksi untuk rekomendasi aktif di halaman Rekomendasi atau tab Rekomendasi di halaman Database.

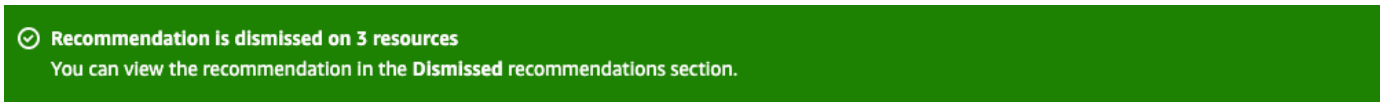
     Halaman detail rekomendasi menampilkan daftar sumber daya yang terpengaruh.
3. Pilih satu atau beberapa rekomendasi, atau satu atau beberapa sumber daya yang terpengaruh di halaman detail rekomendasi, lalu pilih Singkirkan.

   Contoh berikut menunjukkan halaman Rekomendasi dengan beberapa rekomendasi aktif yang dipilih untuk diberhentikan.




Spanduk menampilkan pesan ketika satu atau beberapa rekomendasi yang dipilih diberhentikan.

Contoh berikut menunjukkan spanduk dengan pesan yang berhasil.




Contoh berikut menunjukkan spanduk dengan pesan kegagalan.

CLI

AWS CLI

1. Jalankan perintah `aws rds describe-db-recommendations --filters "Name=status,Values=active"`.

   Output menyediakan daftar rekomendasi dalam `active` status.
2. Temukan `recommendationId` rekomendasi yang ingin Anda abaikan dari langkah 1.
3. Jalankan perintah `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` dengan `recommendationId` dari langkah 2 untuk mengabaikan rekomendasi.

RDS API

# Memodifikasi rekomendasi Amazon RDS Aurora Aurora yang diberhentikan menjadi rekomendasi aktif

Anda dapat memindahkan satu atau lebih rekomendasi Amazon Aurora Aurora yang diberhentikan ke rekomendasi aktif menggunakan konsol RDS Amazon AWS CLI,, atau Amazon. RDS API

Konsol

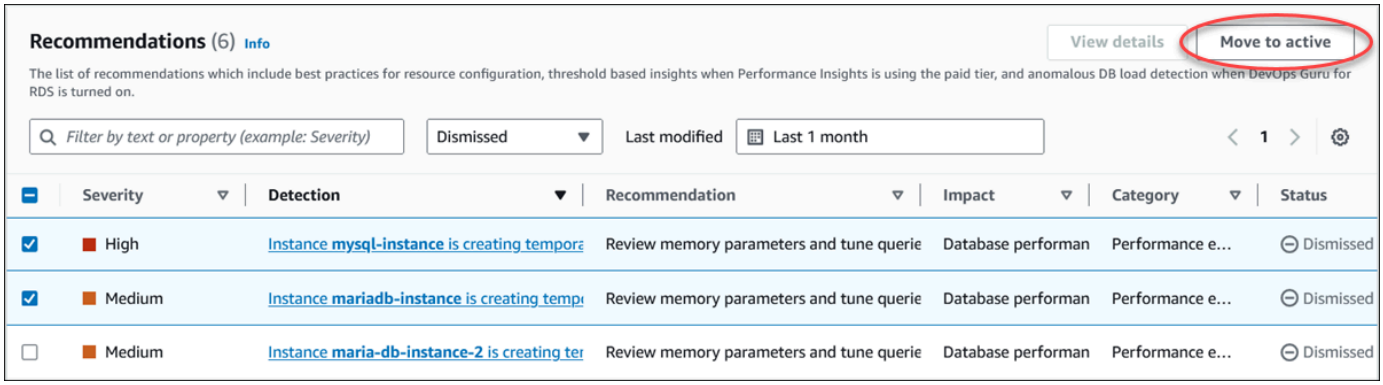
Untuk memindahkan satu atau lebih rekomendasi yang diberhentikan ke rekomendasi aktif

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi.

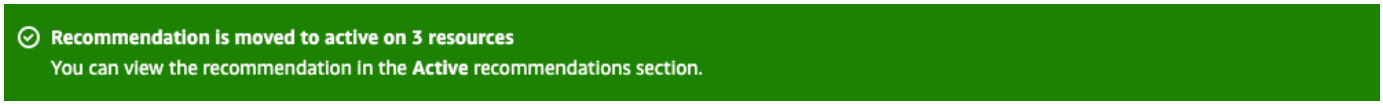
     Halaman Rekomendasi menampilkan daftar rekomendasi yang diurutkan berdasarkan tingkat keparahan semua sumber daya di akun Anda.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Tab Rekomendasi menampilkan rekomendasi dan detailnya untuk sumber daya yang dipilih.
3. Pilih satu atau beberapa rekomendasi yang diberhentikan dari daftar dan kemudian pilih Pindah ke aktif.




Spanduk menampilkan pesan yang berhasil atau gagal saat memindahkan rekomendasi yang dipilih dari diberhentikan ke status aktif.

Contoh berikut menunjukkan spanduk dengan pesan yang berhasil.




Contoh berikut menunjukkan spanduk dengan pesan kegagalan.

CLI

Untuk mengubah rekomendasi Aurora yang diberhentikan menjadi rekomendasi aktif menggunakan AWS CLI

1. Jalankan perintah `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"`.

   Output menyediakan daftar rekomendasi dalam `dismissed` status.
2. Temukan `recommendationId` rekomendasi yang ingin Anda ubah statusnya dari langkah 1.
3. Jalankan perintah `>aws rds modify-db-recommendation --status active --recommendationId <ID>` dengan `recommendationId` dari langkah 2 untuk mengubah ke rekomendasi aktif.

RDS API

Untuk mengubah rekomendasi Aurora yang diberhentikan menjadi rekomendasi aktif menggunakan RDS API Amazon, gunakan operasi M. odifyDBRecommendation

## Rekomendasi dari Amazon Aurora

Amazon Aurora menghasilkan rekomendasi untuk sumber daya saat sumber daya dibuat atau dimodifikasi. Anda dapat menemukan contoh rekomendasi dari Aurora di tabel berikut.

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| Sumber Daya Pencadang | Pencadangan otomatis tidak | Aktifkan pencadang an otomatis dengan | Ya | Gambaran umum pencadangan dan |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
|---|---|---|---|---|
| an otomatis dimatikan | diaktifkan untuk instans DB Anda. Pencadangan otomatis direkomen dasikan karena memungkinkan point-in-time pemulihan instans DB Anda. | periode retensi hingga 14 hari. | | pemulihan klaster DB Aurora<br><br>Mengungkap biaya penyimpanan cadangan RDS Amazon di AWS Blog Database |
| Diperlukan pemutakhiran versi minor engine | Sumber daya database Anda tidak menjalank an versi mesin DB minor terbaru. Versi minor terbaru berisi perbaikan keamanan terbaru dan peningkat an lainnya. | Tingkatkan ke versi mesin terbaru. | Ya | Memelihara klaster DB Amazon Aurora |
| Peningkatan Monitoring dimatikan | Sumber daya database Anda tidak mengaktifkan Enhanced Monitorin g. Pemantauan yang Ditingkatkan menyediakan metrik sistem operasi waktu nyata untuk pemantauan dan pemecahan masalah. | Aktifkan Pemantauan yang Ditingkatkan. | Tidak | Memantau metrik OS dengan Pemantauan yang Ditingkatkan |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| Enkripsi penyimpanan dimatikan | Amazon RDS mendukung enkripsi saat istirahat untuk semua mesin database dengan menggunakan kunci yang Anda kelola di AWS Key Management Service (AWS KMS). Pada instans DB aktif dengan RDS enkripsi Amazon, data yang disimpan saat istirahat di penyimpanan dienkripsi, mirip dengan pencadang an otomatis, replika baca, dan snapshot.<br><br>Jika enkripsi tidak diaktifkan saat membuat cluster Aurora DB, Anda harus mengembal ikan snapshot yang didekripsi ke cluster DB terenkripsi. | Aktifkan enkripsi data saat istirahat untuk cluster DB Anda. | Ya | Keamanan dalam Amazon Aurora |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
|---|---|---|---|---|
| Cluster DB dengan semua instance di Availability Zone yang sama | Cluster DB saat ini berada dalam satu Availability Zone. Gunakan beberapa Availability Zone untuk meningkatkan ketersediaan. | Tambahkan instans DB ke beberapa Availability Zone di cluster DB Anda. | Tidak | Ketersediaan yang tinggi untuk Amazon Aurora |
| Instans DB dalam cluster dengan ukuran instans heterogen | Kami menyarankan Anda menggunak an kelas dan ukuran instans DB yang sama untuk semua instans DB di cluster DB Anda. | Gunakan kelas dan ukuran instance yang sama untuk semua instans DB di cluster DB Anda. | Ya | Replikasi dengan Amazon Aurora |
| Instans DB dalam cluster dengan kelas instance heterogen | Kami menyarankan Anda menggunak an kelas dan ukuran instans DB yang sama untuk semua instans DB di cluster DB Anda. | Gunakan kelas dan ukuran instance yang sama untuk semua instans DB di cluster DB Anda. | Ya | Replikasi dengan Amazon Aurora |
| Instans DB dalam cluster dengan kelompok parameter heterogen | Kami merekomen dasikan bahwa semua instance DB di cluster DB menggunakan grup parameter DB yang sama. | Kaitkan instans DB dengan grup parameter DB yang terkait dengan instance penulis di cluster DB Anda. | Tidak | |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
|---|---|---|---|---|
| Cluster Amazon RDS DB memiliki satu instans DB | Tambahkan setidakny a satu instans DB lagi ke cluster DB Anda untuk meningkatkan ketersediaan dan kinerja. | Tambahkan instans DB pembaca ke cluster DB Anda. | Tidak | Ketersediaan yang tinggi untuk Amazon Aurora |
| Performan ce Insights dimatikan | Performance Insights memantau pemuatan instans DB untuk membantu Anda menganalisis dan menyelesaikan masalah kinerja database. Kami menyarankan Anda mengaktifkan Performance Insights. | Mengaktifkan Wawasan Performa. | Tidak | Memantau muatan DB dengan Wawasan Performa di Amazon Aurora |
| RDSsumber daya pembaruan versi utama diperlukan | Database dengan versi utama saat ini untuk mesin DB tidak akan didukung. Kami menyarankan Anda meningkatkan ke versi utama terbaru yang mencakup fungsiona litas dan peningkatan baru. | Tingkatkan ke versi utama terbaru untuk mesin DB. | Ya | Pembaruan Amazon Aurora<br><br>Membuat penyebara n biru/hijau di Amazon RDS Aurora |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
|---|---|---|---|---|
| Cluster DB hanya mendukung volume hingga 64 TiB | Cluster DB Anda mendukung volume hingga 64 TiB. Versi mesin terbaru mendukung volume hingga 128 TiB untuk cluster DB Anda. Kami menyarankan Anda meningkatkan versi mesin cluster DB Anda ke versi terbaru untuk mendukung volume hingga 128 TiB. | Tingkatkan versi mesin cluster DB Anda untuk mendukung volume hingga 128 TiB. | Ya | Batas ukuran Amazon Aurora |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| Cluster DB dengan semua instance pembaca di Availability Zone yang sama | Availability Zones (AZs) adalah lokasi yang berbeda satu sama lain untuk memberika n isolasi jika terjadi pemadaman di setiap AWS Wilayah. Kami menyarankan Anda mendistribusikan instans utama dan instans pembaca di cluster DB Anda di beberapa AZs untuk meningkatkan ketersediaan cluster DB Anda. Anda dapat membuat klaster Multi-AZ menggunak an AWS Managemen t Console AWS CLI, atau Amazon RDS API saat membuat klaster. Anda dapat memodifikasi cluster Aurora yang ada ke cluster multi-AZ dengan menambahk an instance pembaca baru dan menentukan AZ yang berbeda. | Cluster DB Anda memiliki semua instance bacaannya di Availability Zone yang sama. Kami menyarankan Anda mendistribusikan instance pembaca di beberapa Availabil ity Zone. Distribus i meningkatkan ketersediaan dan meningkatkan waktu respons dengan mengurangi latensi jaringan antara klien dan database. | Tidak | Ketersediaan yang tinggi untuk Amazon Aurora |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| Parameter memori DB menyimpang dari default | Parameter memori instans DB berbeda secara signifikan dari nilai default. Pengaturan ini dapat memengaruhi kinerja dan menyebabkan kesalahan.<br><br>Kami menyarankan Anda mengatur ulang parameter memori khusus untuk instans DB ke nilai defaultnya di grup parameter DB. | Setel ulang parameter memori ke nilai defaultnya. | Tidak | |
| Parameter cache kueri diaktifkan | Ketika perubahan mengharuskan cache kueri Anda dibersihk an, instans DB Anda akan tampak macet. Cache kueri tidak bermanfaat untuk sebagian besar beban kerja. Cache kueri telah dihapus dari My SQL 8.0 dan versi yang lebih tinggi. Kami menyarank an Anda mengatur parameter query_cac he_type ke 0. | Tetapkan nilai `query_cac he_type` parameter ke `0` dalam grup parameter DB Anda. | Ya | |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `log_outpu t` parameter diatur ke tabel | Ketika `log_output` diatur ke`TABLE`, lebih banyak penyimpanan digunakan daripada ketika `log_outpu t` diatur ke`FILE`. Kami menyarank an Anda mengatur parameter ke`FILE`, untuk menghinda ri mencapai batas ukuran penyimpanan. Setel ke secara `FILE` default di My SQL 8.4 dan versi yang lebih tinggi. | Tetapkan nilai `log_output` parameter ke `FILE` dalam grup parameter DB Anda. | Tidak | File log basis data Aurora MySQL |
| `autovacuu m` parameter dimatikan | Parameter autovacuu m dimatikan untuk cluster Anda. Mematikan autovacuu m meningkatkan tabel dan indeks kembung dan berdampak pada kinerja.<br><br>Kami menyarankan Anda mengaktifkan autovacuum di grup parameter DB Anda. | Aktifkan parameter autovacuum di grup parameter cluster DB Anda. | Tidak | Memahami autovacuu m di Amazon RDS untuk lingkungan Postgre SQL di Blog Database AWS |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
|---|---|---|---|---|
| `synchrono us_commit` parameter dimatikan | Ketika `synchrono us_commit` parameter dimatikan , data dapat hilang dalam kerusakan database. Daya tahan database berisiko.<br><br>Sebaiknya aktifkan parameter `synchrono us_commit` . | Aktifkan `synchrono us_commit` parameter di grup parameter DB Anda. | Ya | SQLParameter Amazon Aurora Postgre: Replikasi , keamanan, dan logging di Blog Database AWS |
| `track_cou nts` parameter dimatikan | Ketika `track_cou nts` parameter dimatikan, database tidak mengumpul kan statistik aktivitas database. Autovacuu m membutuhkan statistik ini untuk berfungsi dengan benar.<br><br>Sebaiknya tetapkan parameter `track_counts` ke 1. | Setel `track_cou nts` parameter ke1. | Tidak | Statistik Run-time untuk Postgre SQL |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `enable_in dexonlysc an` parameter dimatikan | Perencana kueri atau pengoptimal tidak dapat menggunakan jenis paket pemindaia n khusus indeks saat dimatikan.<br><br>Kami menyarank an Anda mengatur nilai `enable_in dexonlyscan` parameter ke`1`. | Tetapkan nilai `enable_in dexonlyscan` parameter ke`1`. | Tidak | Konfigurasi Metode Perencana untuk Postgre SQL |
| `enable_in dexscan` para dimatikan | Perencana kueri atau pengoptimal tidak dapat menggunak an jenis rencana pemindaian indeks saat dimatikan.<br><br>Kami menyarankan Anda menetapka n `enable_in dexscan` nilainya`1`. | Tetapkan nilai `enable_in dexscan` parameter ke`1`. | Tidak | Konfigurasi Metode Perencana untuk Postgre SQL |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `innodb_fl ush_log_a t_trx` parame dimatikan | Nilai `innodb_fl ush_log_at_trx` parameter instans DB Anda bukanlah nilai aman. Parameter ini mengontrol persisten si operasi commit ke disk.<br><br>Sebaiknya tetapkan parameter `innodb_fl ush_log_at_trx` ke 1. | Tetapkan nilai `innodb_fl ush_log_at_trx` parameter ke1. | Tidak | Mengonfigurasi seberapa sering buffer log di-flush |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `innodb_st ats_persi stent` parame dimatikan | Instans DB Anda tidak dikonfigurasi untuk mempertahankan statistik InnoDB ke disk. Ketika statistik tidak disimpan, mereka dihitung ulang setiap kali instance restart dan tabel diakses. Hal ini menyebabkan variasi dalam rencana eksekusi query. Anda dapat memodifikasi nilai parameter global ini di tingkat tabel.<br><br>Kami menyarank an Anda mengatur nilai `innodb_st ats_persistent` parameter ke`ON`. | Tetapkan nilai `innodb_st ats_persistent` parameter ke`ON`. | Tidak | |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `innodb_op en_files` Par rendah | `innodb_op en_files` Parameter mengontrol jumlah file InnoDB dapat membuka pada satu waktu. InnoDB membuka semua log dan file tablespace sistem saat mysqld berjalan.<br><br>Instans DB Anda memiliki nilai rendah untuk jumlah maksimum file yang dapat dibuka InnoDB pada satu waktu. Sebaiknya tetapkan parameter `innodb_op en_files` ke nilai minimum 65. | Atur `innodb_op en_files` parameter ke nilai minimum65. | Ya | InnoDB membuka file untuk My SQL |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `max_user_ connectio ns` Parameter rendah | Instans DB Anda memiliki nilai rendah untuk jumlah maksimum koneksi simultan untuk setiap akun basis data.<br><br>Kami merekomen dasikan pengatura n `max_user_ connections` parameter ke angka yang lebih besar dari5. | Tingkatkan nilai `max_user_ connections` parameter ke angka yang lebih besar dari5. | Ya | Menetapkan Batas Sumber Daya Akun untuk Saya SQL |
| Baca Replika terbuka dalam mode yang dapat ditulis | Instans DB Anda memiliki replika baca dalam mode yang dapat ditulis, yang memungkinkan pembaruan dari klien.<br><br>Kami menyarank an Anda mengatur `read_only` parameter `TrueIfReplica` agar replika baca tidak dalam mode yang dapat ditulis. | Tetapkan nilai `read_only` parameter keTrueIfReplica . | Tidak | |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `innodb_de fault_row _format` peng n parameter tidak aman | Instans DB Anda mengalami masalah yang diketahui: Tabel yang dibuat dalam SQL versi Saya lebih rendah dari 8.0.26 dengan `row_forma t` set ke `COMPACT` atau `REDUNDANT` akan tidak dapat diakses dan tidak dapat dipulihkan ketika indeks melebihi 767 byte.<br><br>Kami menyarank an Anda mengatur nilai `innodb_de fault_row _format` parameter ke`DYNAMIC`. | Tetapkan nilai `innodb_de fault_row _format` parameter ke`DYNAMIC`. | Tidak | Perubahan pada SQL 8.0.26 Saya |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| `general_l ogging` param dihidupkan | Pencatatan umum diaktifkan untuk instans DB Anda. Pengaturan ini berguna saat memecahkan masalah database. Namun, menyalaka n logging umum meningkatkan jumlah operasi I/O dan ruang penyimpanan yang dialokasikan, yang dapat mengakibatkan pertengkaran dan penurunan kinerja.<br><br>Periksa persyarat an Anda untuk penggunaan logging umum. Kami menyarankan Anda mengatur nilai `general_logging` parameter ke`0`. | Periksa persyarat an Anda untuk penggunaan logging umum. Jika tidak wajib, kami sarankan Anda untuk mengatur nilai `general_l ogging` parameter ke`0`. | Tidak | Ringkasan log basis data Aurora MySQL |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| Cluster DB kurang disediakan untuk beban kerja baca | Kami menyarankan Anda menambahkan instans DB pembaca ke cluster DB Anda dengan kelas dan ukuran instans yang sama dengan instance DB penulis di cluster. Konfigurasi saat ini memiliki satu instans DB dengan beban database yang terus menerus tinggi yang sebagian besar disebabka n oleh operasi baca. Distribusikan operasi ini dengan menambahkan instans DB lain ke cluster dan mengarahkan beban kerja baca ke titik akhir read-only cluster DB. | Tambahkan instance DB pembaca ke cluster. | Tidak | Menambahkan Replika Aurora ke klaster DB<br><br>Mengelola performa dan penskalaan untuk klaster DB Aurora<br><br>RDSHarga Amazon |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
|---|---|---|---|---|
| RDSinstan ce kurang disediaka n untuk kapasitas memori sistem | Kami menyarank an Anda menyetel kueri Anda untuk menggunakan memori yang lebih rendah atau menggunak an jenis instans DB dengan memori yang dialokasikan lebih tinggi. Ketika instance kehabisan memori, maka kinerja database terpengar uh. | Gunakan instans DB dengan kapasitas memori yang lebih tinggi | Ya | Menskalakan RDS Instans Amazon Anda Secara Vertikal dan Horizontal di Blog Database AWS<br><br>Jenis RDS instans Amazon<br><br>RDSHarga Amazon |
| RDSinstan ce kurang disediaka n untuk kapasitas sistem CPU | Sebaiknya Anda menyetel kueri untuk menggunakan lebih sedikit CPU atau memodifikasi instans DB Anda untuk menggunakan kelas instans DB dengan alokasi vCPUs yang lebih tinggi. Kinerja database mungkin menurun ketika instans DB kehabisan CPU. | Gunakan instans DB dengan CPU kapasitas lebih tinggi | Ya | Menskalakan RDS Instans Amazon Anda Secara Vertikal dan Horizontal di Blog Database AWS<br><br>Jenis RDS instans Amazon<br><br>RDSHarga Amazon |

| Tipe | Deskripsi | Rekomendasi | Diperlu n downtir | Informasi tambahan |
| --- | --- | --- | --- | --- |
| RDSsumber daya tidak memanfaat kan penyatuan koneksi dengan benar | Kami menyarank an agar Anda mengaktifkan Amazon RDS Proxy untuk menggabungkan dan berbagi koneksi database yang ada secara efisien. Jika Anda sudah menggunakan proxy untuk database Anda, konfigura sikan dengan benar untuk meningkatkan penyatuan koneksi dan penyeimba ngan beban di beberapa instans DB. RDS Proxy dapat membantu mengurangi risiko kelelahan koneksi dan downtime sekaligus meningkat kan ketersediaan dan skalabilitas. | Aktifkan RDS Proxy atau ubah konfigurasi proxy yang ada | Tidak | Menskalakan RDS Instans Amazon Anda Secara Vertikal dan Horizontal di Blog Database AWS<br><br>Proksi Amazon RDS untuk Aurora<br><br>Harga RDS Proxy Amazon |

# Melihat metrik di konsol Amazon RDS

Amazon RDS terintegrasi dengan Amazon CloudWatch untuk menampilkan berbagai metrik instans DB RDS cluster Aurora di konsol. RDS Beberapa metrik diterapkan di tingkat klaster, sedangkan metrik lainnya diterapkan di tingkat instans. Untuk deskripsi metrik tingkat instans dan tingkat klaster, lihat Referensi metrik untuk Aurora.

Untuk klater DB Aurora, kategori metrik berikut dipantau:

- CloudWatch— Menampilkan CloudWatch metrik Amazon untuk Aurora yang dapat Anda akses di RDS konsol. Anda juga dapat mengakses metrik ini di CloudWatch konsol. Setiap metrik berisi grafik yang menunjukkan metrik yang dipantau selama rentang waktu tertentu. Untuk daftar CloudWatch metrik, lihat .
- Pemantauan yang disempurnakan — Menampilkan ringkasan metrik sistem operasi saat DB cluster RDS Aurora DB Anda mengaktifkan Pemantauan yang Ditingkatkan. RDSmengirimkan metrik dari Enhanced Monitoring ke akun Amazon CloudWatch Logs Anda. Setiap metrik OS berisi grafik yang menunjukkan metrik yang dipantau selama rentang waktu tertentu. Untuk ikhtisar, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. Untuk daftar metrik Pemantauan yang Ditingkatkan, lihat Metrik OS dalam Pemantauan yang Ditingkatkan.
- Daftar Proses OS – Menampilkan detail setiap proses yang berjalan dalam klaster DB Anda.
- Performance Insights — Membuka dasbor Amazon RDS Performance Insights untuk instans DB di cluster Aurora DB Anda. Wawasan Performa tidak didukung di tingkat klaster. Untuk ikhtisar Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. Untuk daftar metrik Wawasan Performa, lihat CloudWatch Metrik Amazon untuk Amazon RDS Performance Insights.

Amazon RDS kini menyediakan tampilan konsolidasi Performance Insights CloudWatch dan metrik di dasbor Performance Insights. Untuk menggunakan tampilan ini, Wawasan Performa harus diaktifkan klaster DB Anda. Anda dapat memilih tampilan pemantauan baru di tab Pemantauan atau Wawasan Performa di panel navigasi. Untuk melihat petunjuk cara memilih tampilan ini, lihat Melihat metrik gabungan dengan dasbor Performance Insights.

# Melihat metrik gabungan dengan dasbor Performance Insights

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Amazon RDS menyediakan tampilan konsolidasi Performance Insights CloudWatch dan metrik untuk instans DB Anda di dasbor Performance Insights. Anda dapat menggunakan dasbor yang telah dikonfigurasi sebelumnya atau membuat dasbor kustom. Dasbor yang telah dikonfigurasi sebelumnya menyediakan metrik yang paling umum digunakan untuk membantu mendiagnosis masalah performa untuk mesin basis data. Alternatifnya, Anda dapat membuat dasbor kustom dengan metrik untuk mesin basis data yang memenuhi persyaratan analisis Anda. Kemudian, gunakan dasbor ini untuk semua instance DB dari jenis mesin database di AWS akun Anda.

Anda dapat memilih tampilan pemantauan di tab Monitoring atau Performance Insights di panel navigasi.

Wawasan Performa harus diaktifkan untuk klaster DB Anda untuk melihat metrik gabungan di dasbor Wawasan Performa. Untuk informasi selengkapnya tentang mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora.

Di bagian berikut, Anda dapat mempelajari cara menampilkan Performance Insights dan CloudWatch metrik.

Topik


- Memilih tampilan pemantauan baru dari tab Monitoring
- Memilih tampilan pemantauan baru dari halaman Performance Insights
- Membuat dasbor khusus dengan Performance Insights
- Memilih dasbor yang telah dikonfigurasi sebelumnya dengan Performance Insights


## Memilih tampilan pemantauan baru dari tab Monitoring

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database

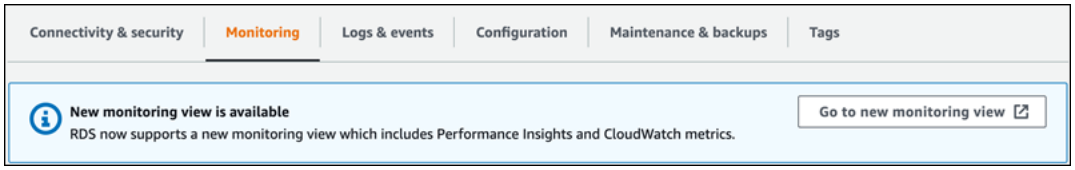
Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Dari konsol Amazon RDS, Anda dapat memilih tampilan pemantauan baru untuk melihat Performance Insights CloudWatch dan metrik untuk instans DB Anda.

Untuk memilih tampilan pemantauan baru di tab Monitoring

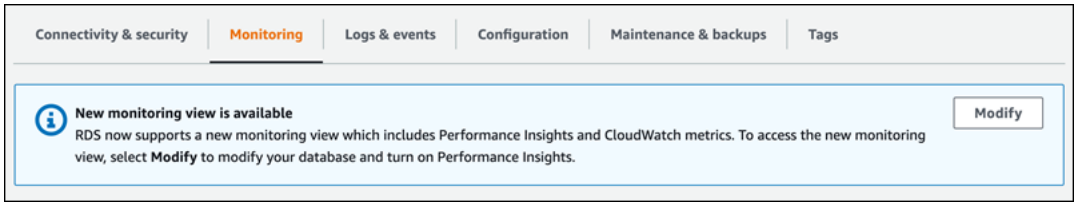
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pada panel navigasi kiri, pilih Basis data.
3. Pilih klaster DB Aurora yang ingin Anda pantau.
4. Gulir ke bawah dan pilih tab Pemantauan.

   Banner akan muncul dengan opsi untuk memilih tampilan pemantauan baru. Contoh berikut menampilkan banner untuk memilih tampilan pemantauan baru.




5.
6. (Opsional) Jika Wawasan Performa dinonaktifkan untuk instans DB Anda, banner akan muncul dengan opsi untuk memodifikasi instans DB dan mengaktifkan Wawasan Performa.

   Contoh berikut menampilkan banner untuk memodifikasi instans DB di tab Pemantauan.




   Pilih Modifikasi untuk memodifikasi instans DB dan mengaktifkan Wawasan Performa. Untuk informasi selengkapnya tentang mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora.

# Memilih tampilan pemantauan baru dari halaman Performance Insights

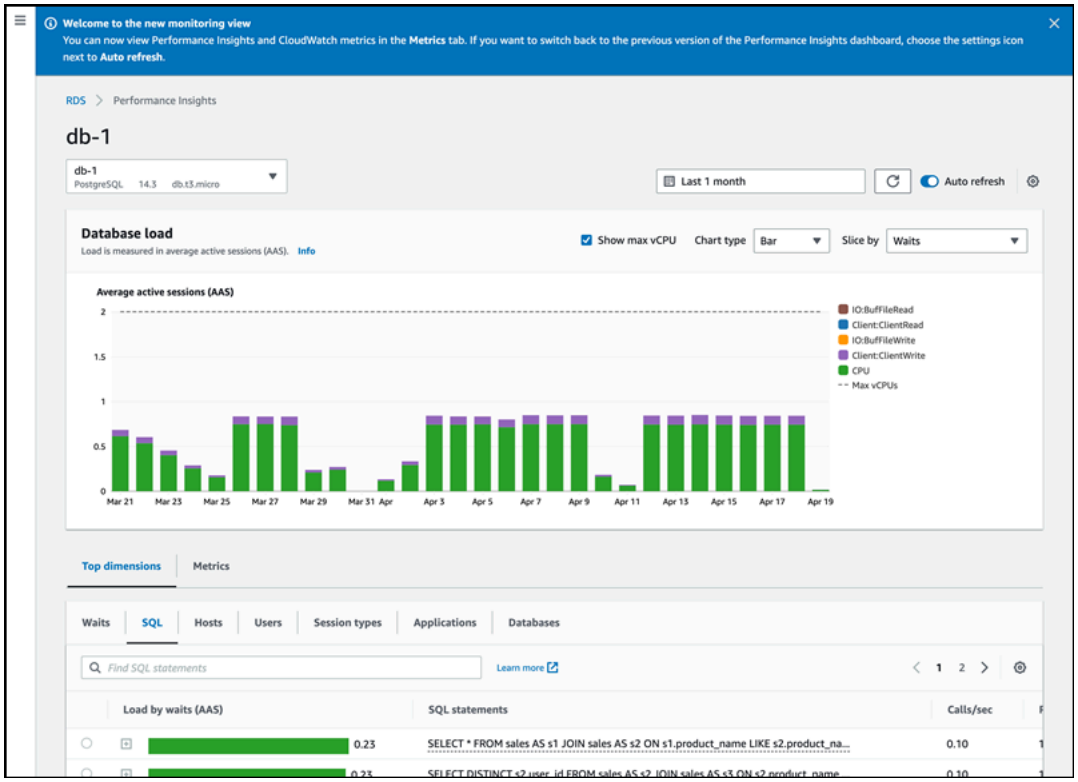
**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang meningkatkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Dari konsol Amazon RDS, Anda dapat memilih tampilan pemantauan baru untuk melihat Performance Insights CloudWatch dan metrik untuk instans DB Anda.

Untuk memilih tampilan pemantauan baru dengan Performance Insights di panel navigasi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB untuk melihat dasbor Performance Insights yang menampilkan Performance Insights CloudWatch dan metrik untuk instans DB Anda.




# Membuat dasbor khusus dengan Performance Insights

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan.

Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Di tampilan pemantauan baru, Anda dapat membuat dasbor kustom dengan metrik yang Anda butuhkan untuk memenuhi persyaratan analisis Anda.

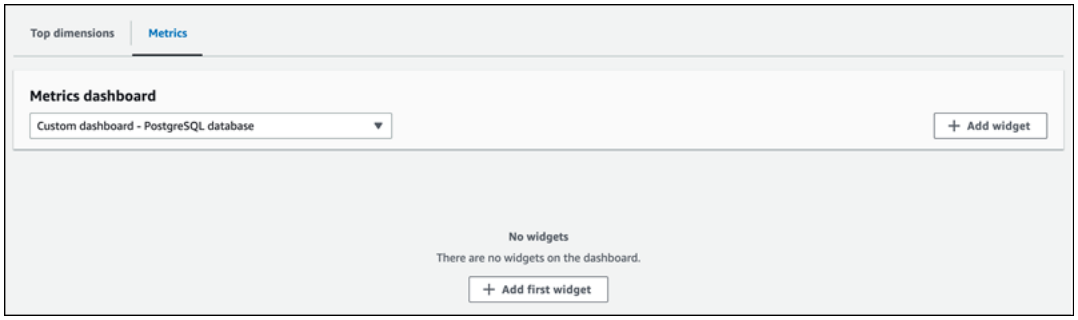
Anda dapat membuat dasbor khusus dengan memilih Performance Insights dan CloudWatch metrik untuk instans DB Anda. Anda dapat menggunakan dasbor khusus ini untuk instans DB lain dari jenis mesin database yang sama di AWS akun Anda.

> **Note**
>
> Dasbor yang disesuaikan mendukung hingga 50 metrik.

Gunakan menu pengaturan widget untuk mengedit atau menghapus dasbor, dan memindahkan atau mengubah ukuran jendela widget.

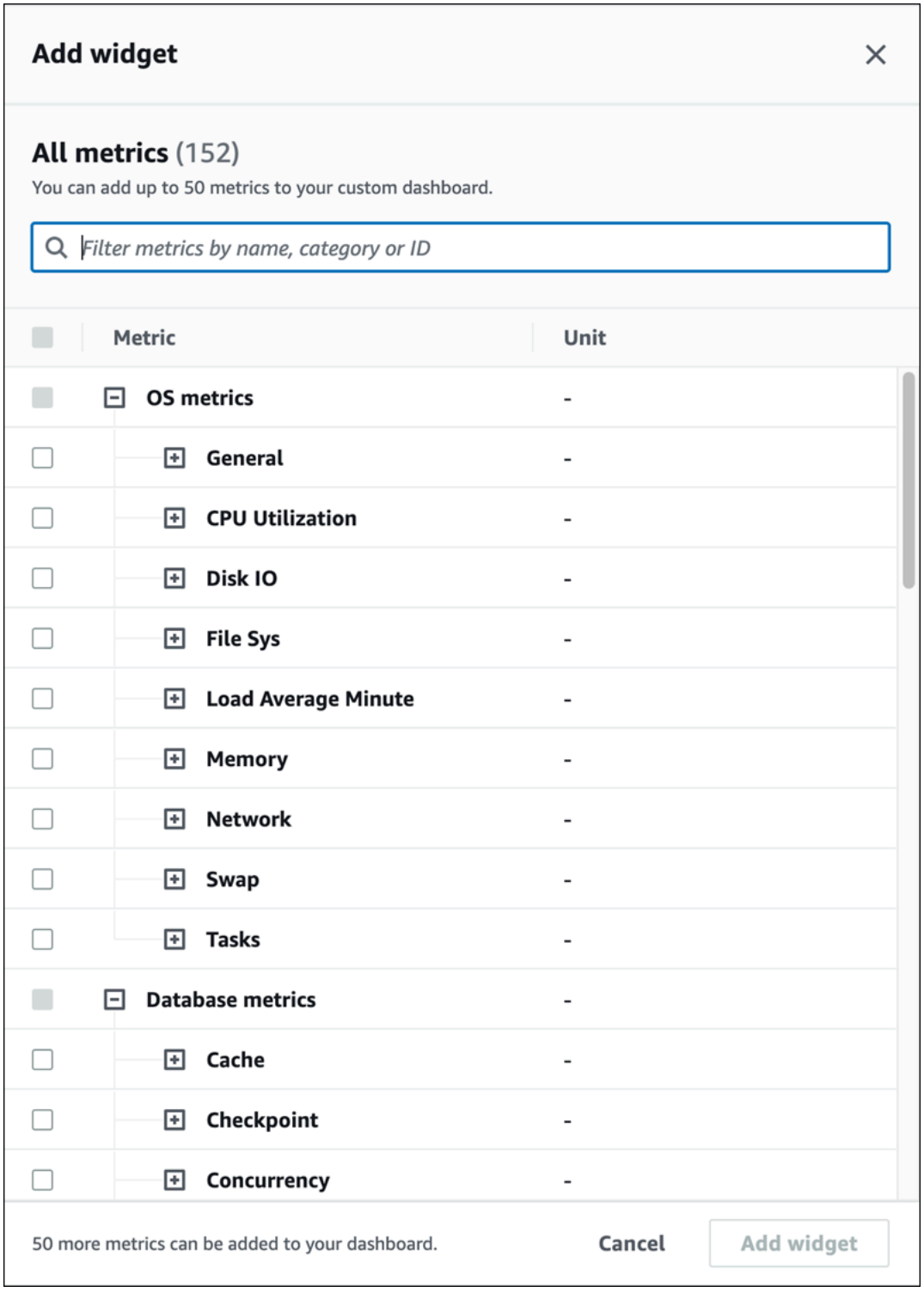
Untuk membuat dasbor kustom dengan Performance Insights di panel navigasi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Gulir ke bawah ke tab Metrik di jendela.
5. Pilih dasbor kustom dari daftar drop-down. Contoh berikut menunjukkan pembuatan dasbor kustom.




6. Pilih Tambahkan widget untuk membuka jendela Tambahkan widget. Anda dapat membuka dan melihat metrik sistem operasi (OS), metrik database, dan CloudWatch metrik yang tersedia di jendela.

   Contoh berikut menampilkan jendela Tambahkan widget dengan metrik.




7. Pilih metrik yang ingin Anda lihat di dasbor dan pilih Tambahkan widget. Anda dapat menggunakan bidang pencarian untuk menemukan metrik tertentu.

   Metrik yang dipilih akan muncul di dasbor Anda.

8. (Opsional) Jika Anda ingin memodifikasi atau menghapus dasbor Anda, pilih ikon pengaturan di kanan atas widget, lalu pilih salah satu tindakan berikut di menu.

   - Edit - Memodifikasi daftar metrik di jendela. Pilih Perbarui widget setelah Anda memilih metrik untuk dasbor Anda.
   - Hapus – Menghapus widget. Di jendela konfirmasi, pilih Hapus.

## Memilih dasbor yang telah dikonfigurasi sebelumnya dengan Performance Insights

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

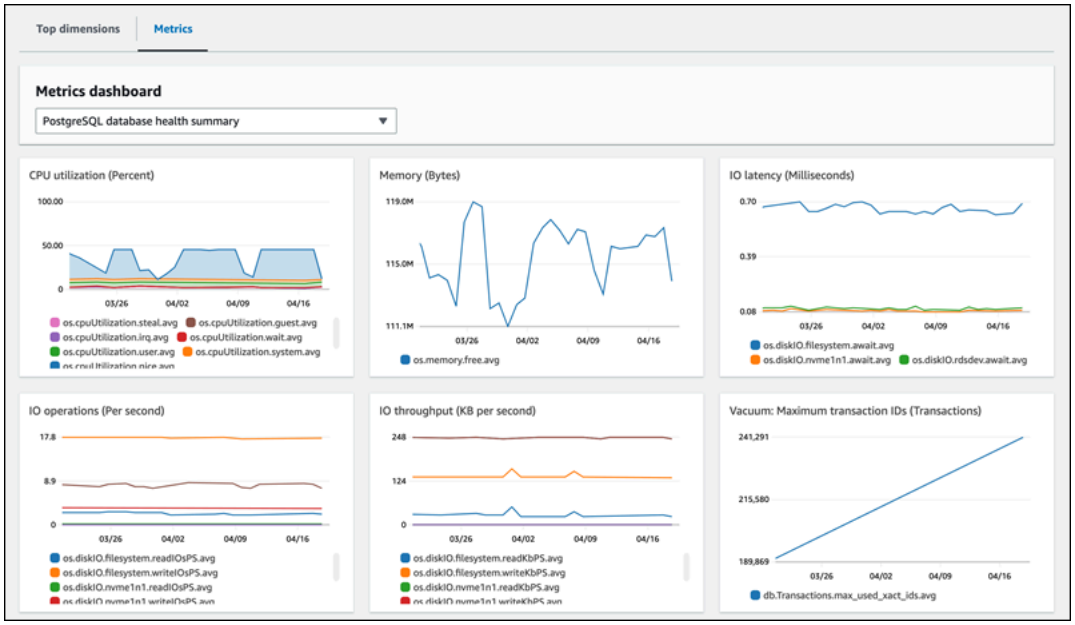
Anda dapat melihat metrik yang paling umum digunakan dengan dasbor yang telah dikonfigurasi sebelumnya. Dasbor ini membantu mendiagnosis masalah performa mesin basis data dan mengurangi waktu pemulihan rata-rata dari hitungan jam menjadi menit.

> **Note**
>
> Dasbor ini tidak dapat diedit.

Untuk memilih dasbor yang telah dikonfigurasi sebelumnya dengan Performance Insights di panel navigasi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Gulir ke bawah ke tab Metrik di jendela
5. Pilih dasbor yang telah dikonfigurasi sebelumnya dari daftar drop-down.

   Anda dapat melihat metrik untuk instans DB di dasbor. Contoh berikut menampilkan dasbor metrik yang telah dikonfigurasi sebelumnya.



# Memantau Amazon Aurora dengan Amazon CloudWatch

Amazon CloudWatch adalah repositori metrik. Repositori ini mengumpulkan dan mengolah data mentah dari Amazon Aurora menjadi metrik-metrik waktu nyaris nyata yang dapat dibaca. Untuk daftar lengkap metrik Aurora yang dikirim CloudWatch ke, Aurora.

Untuk menganalisis dan memecahkan masalah kinerja database Anda dalam skala besar, gunakan Database CloudWatch .

Topik

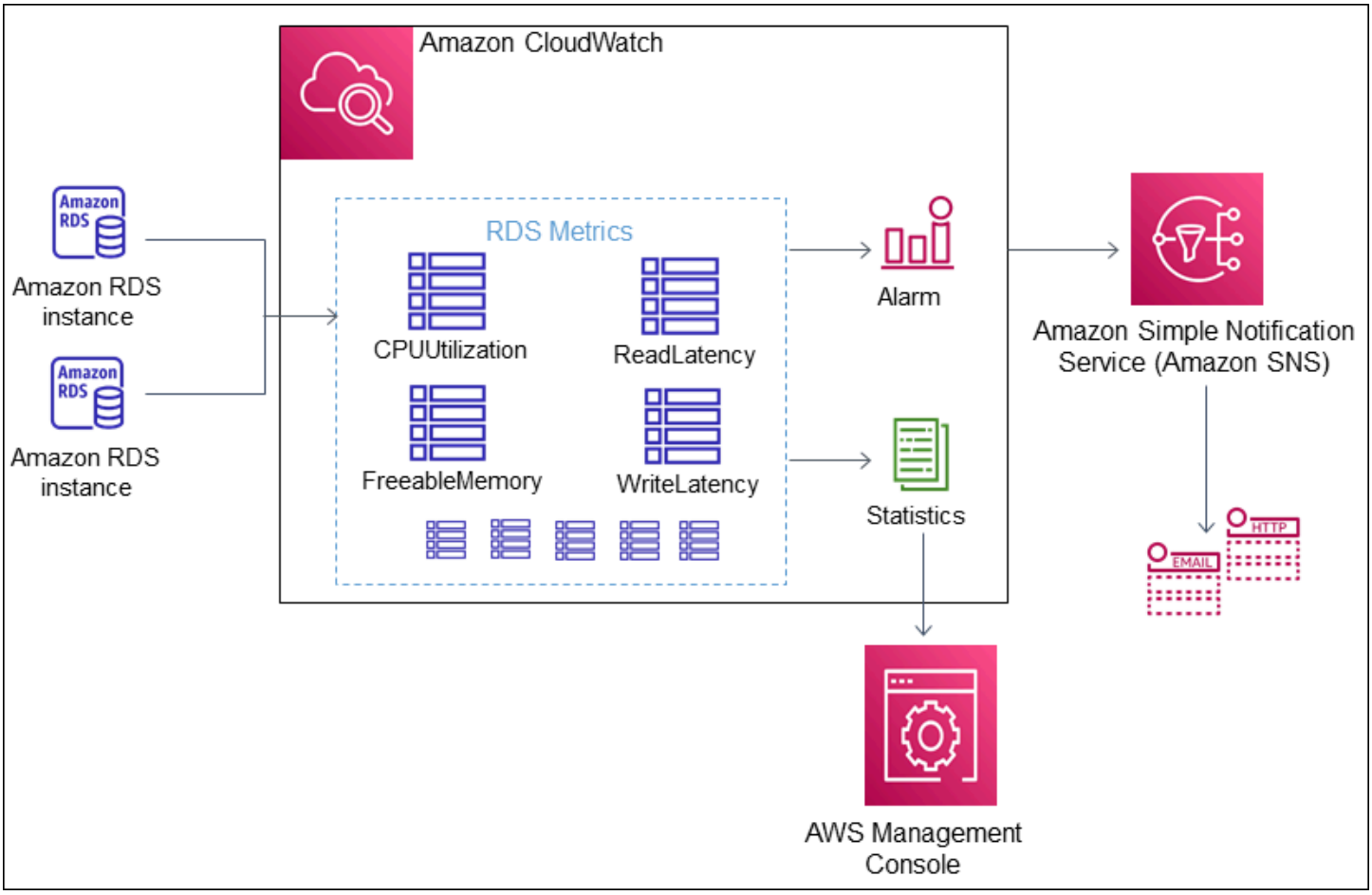
- Ikhtisar Amazon Aurora dan Amazon CloudWatch
- Melihat metrik cluster DB di CloudWatch konsol dan AWS CLI
- Mengekspor metrik Performance Insights ke CloudWatch
- Membuat CloudWatch alarm untuk memantau Aurora

# Ikhtisar Amazon Aurora dan Amazon CloudWatch

Secara default, Amazon Aurora secara otomatis mengirimkan data metrik CloudWatch ke dalam periode 1 menit. Misalnya, metrik `CPUUtilization` mencatat persentase pemanfaatan CPU untuk instans basis data dari waktu ke waktu. Titik data dengan periode 60 detik (1 menit) tersedia selama 15 hari. Ini berarti bahwa Anda dapat mengakses informasi historis dan melihat bagaimana aplikasi web atau layanan Anda beperforma.

Sekarang Anda dapat mengekspor dasbor metrik Performance Insights dari Amazon RDS ke Amazon. CloudWatch Anda dapat mengekspor dasbor metrik yang telah dikonfigurasi atau disesuaikan sebagai dasbor baru atau menambahkannya ke dasbor yang ada. CloudWatch Dasbor yang diekspor tersedia untuk dilihat di CloudWatch konsol. Untuk informasi selengkapnya tentang cara mengekspor dasbor metrik Performance Insights ke, lihat. CloudWatch Mengekspor metrik Performance Insights ke CloudWatch

Seperti yang ditunjukkan pada diagram berikut, Anda dapat mengatur alarm untuk CloudWatch metrik Anda. Misalnya, Anda dapat membuat alarm yang memberi sinyal ketika penggunaan CPU untuk sebuah instans melebihi 70%. Anda dapat mengonfigurasi Amazon Simple Notification Service untuk mengirim email kepada Anda saat ambang batas dilewati.




Amazon RDS menerbitkan jenis metrik berikut ke Amazon: CloudWatch

- Metrik-metrik Aurora di tingkat klaster dan instans

  Lihat tabel metrik-metrik ini di CloudWatch Metrik Amazon untuk Amazon Aurora.

- Metrik-metrik Wawasan Performa

  Lihat tabel metrik-metrik ini di CloudWatch Metrik Amazon untuk Amazon RDS Performance Insights dan Metrik penghitung Wawasan Performa.

- Metrik Pemantauan yang Ditingkatkan (dipublikasikan ke Amazon CloudWatch Logs)

  Lihat tabel metrik-metrik ini di Metrik OS dalam Pemantauan yang Ditingkatkan.

- Metrik penggunaan untuk kuota layanan Amazon RDS di Akun AWS

  Lihat tabel metrik-metrik ini di Metrik CloudWatch penggunaan Amazon untuk . Lihat informasi yang lebih lengkap tentang kuota Amazon RDS di Kuota dan batasan untuk Amazon Aurora.

Untuk informasi selengkapnya CloudWatch, lihat Apa itu Amazon CloudWatch? di Panduan CloudWatch Pengguna Amazon. Untuk informasi selengkapnya tentang retensi CloudWatch metrik, lihat Retensi metrik.

## Melihat metrik cluster DB di CloudWatch konsol dan AWS CLI

Setelah itu, Anda dapat menemukan detail tentang cara melihat metrik untuk instans DB Anda menggunakan CloudWatch. Untuk informasi tentang metrik pemantauan untuk sistem operasi instans DB Anda secara real time menggunakan CloudWatch Log, lihatMemantau metrik OS dengan Pemantauan yang Ditingkatkan.

Saat Anda menggunakan sumber daya Aurora Amazon, Aurora mengirimkan metrik dan dimensi ke Amazon setiap menit. CloudWatch

Sekarang Anda dapat mengekspor dasbor metrik Performance Insights dari Amazon RDS ke CloudWatch Amazon dan melihat metrik ini di konsol. CloudWatch Untuk informasi selengkapnya tentang cara mengekspor dasbor metrik Performance Insights ke, lihat. CloudWatch Mengekspor metrik Performance Insights ke CloudWatch

Gunakan prosedur berikut untuk melihat metrik Amazon Aurora di konsol CloudWatch dan. CLI

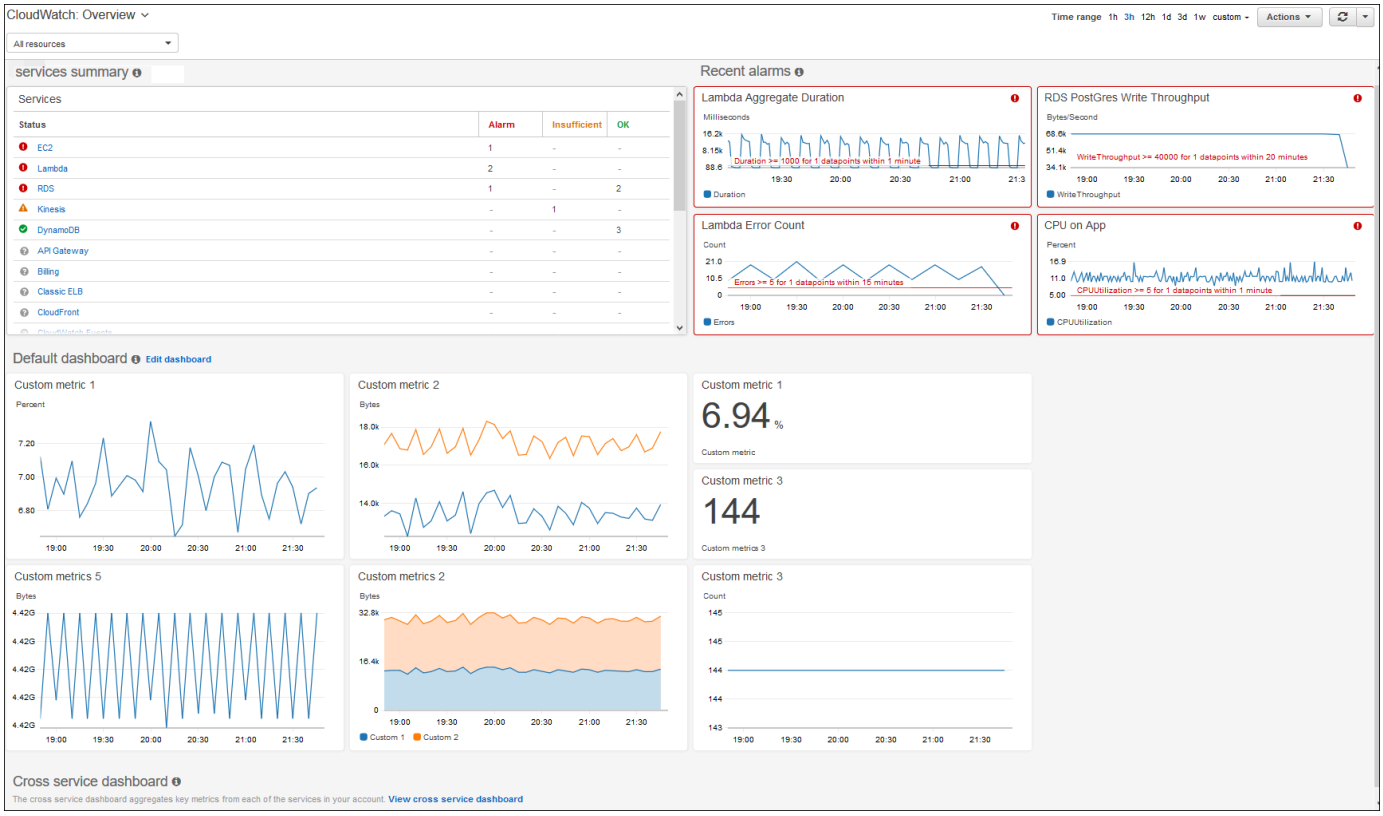
Konsol

Untuk melihat metrik menggunakan konsol Amazon CloudWatch

Metrik dikelompokkan terlebih dahulu berdasarkan namespace layanan, lalu berdasarkan berbagai kombinasi dimensi dalam setiap namespace.

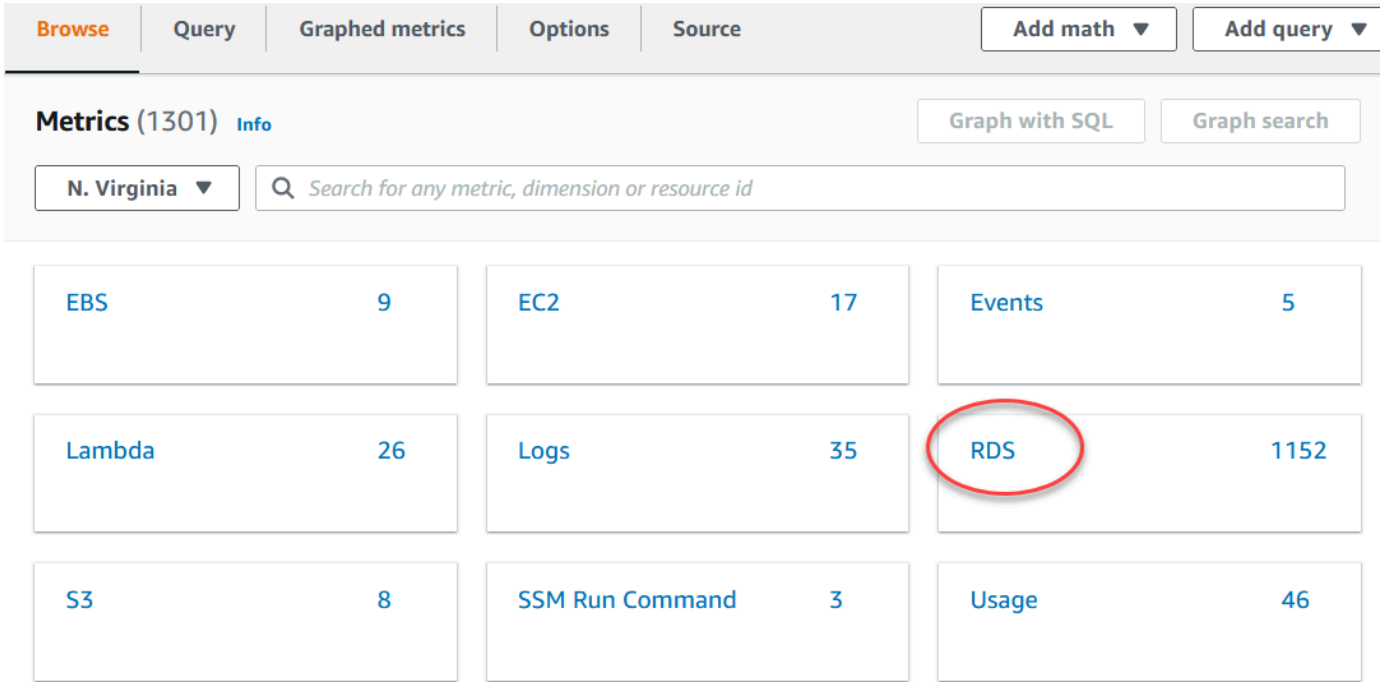
1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.

   Halaman beranda CloudWatch ikhtisar muncul.



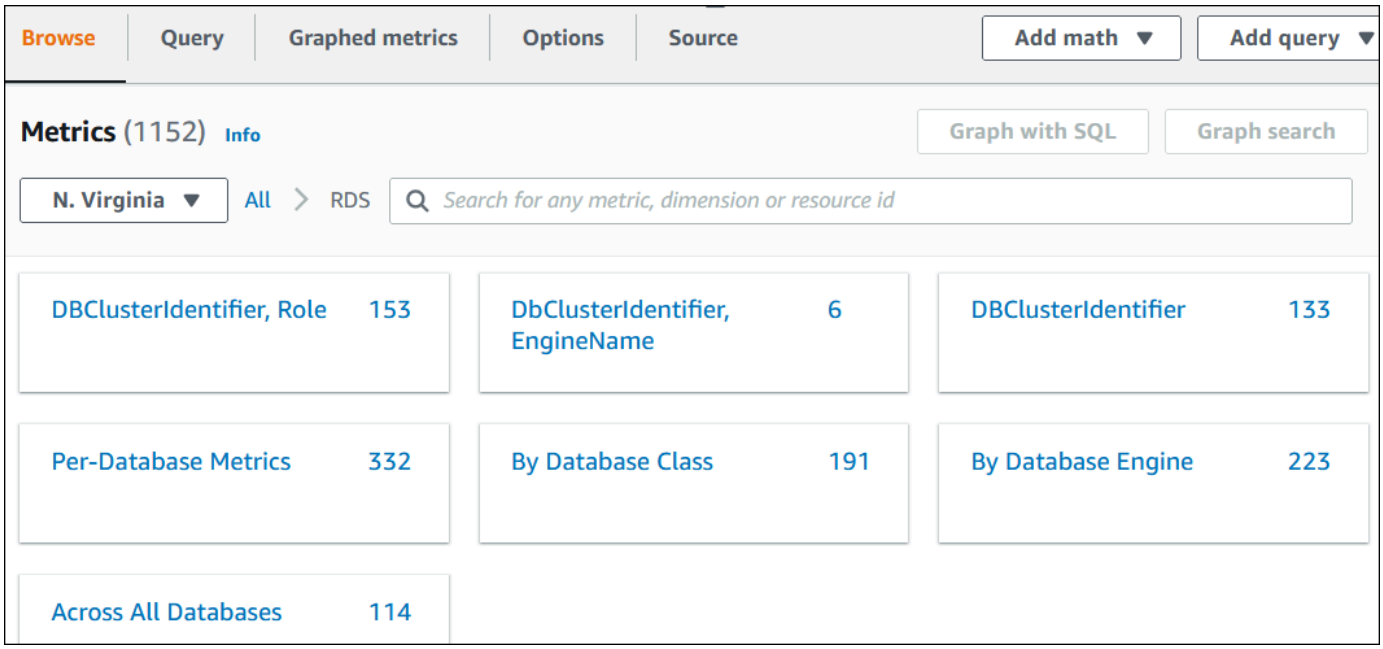

2. Jika perlu, ubah Wilayah AWS. Dari bilah navigasi, pilih Wilayah AWS tempat AWS sumber daya Anda berada. Untuk informasi selengkapnya, lihat Wilayah dan titik akhir.

3. Di panel navigasi, pilih Metrik, lalu Semua metrik.

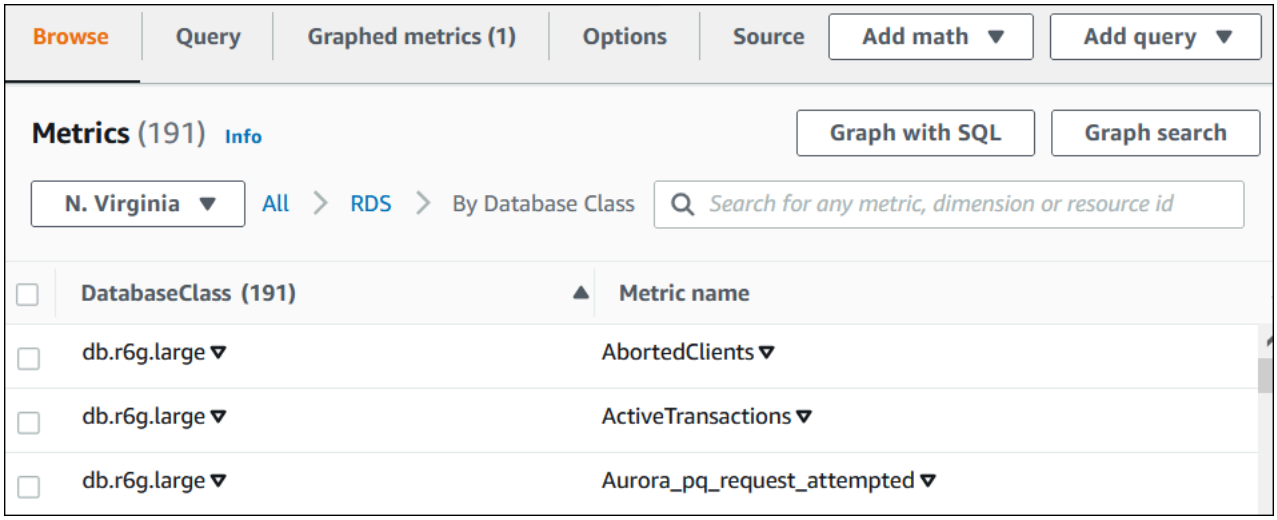



4. Gulir ke bawah dan pilih namespace RDSmetrik.

Halaman ini menampilkan dimensi Aurora. Untuk deskripsi semua dimensi ini, lihat CloudWatch Dimensi Amazon untuk Aurora.



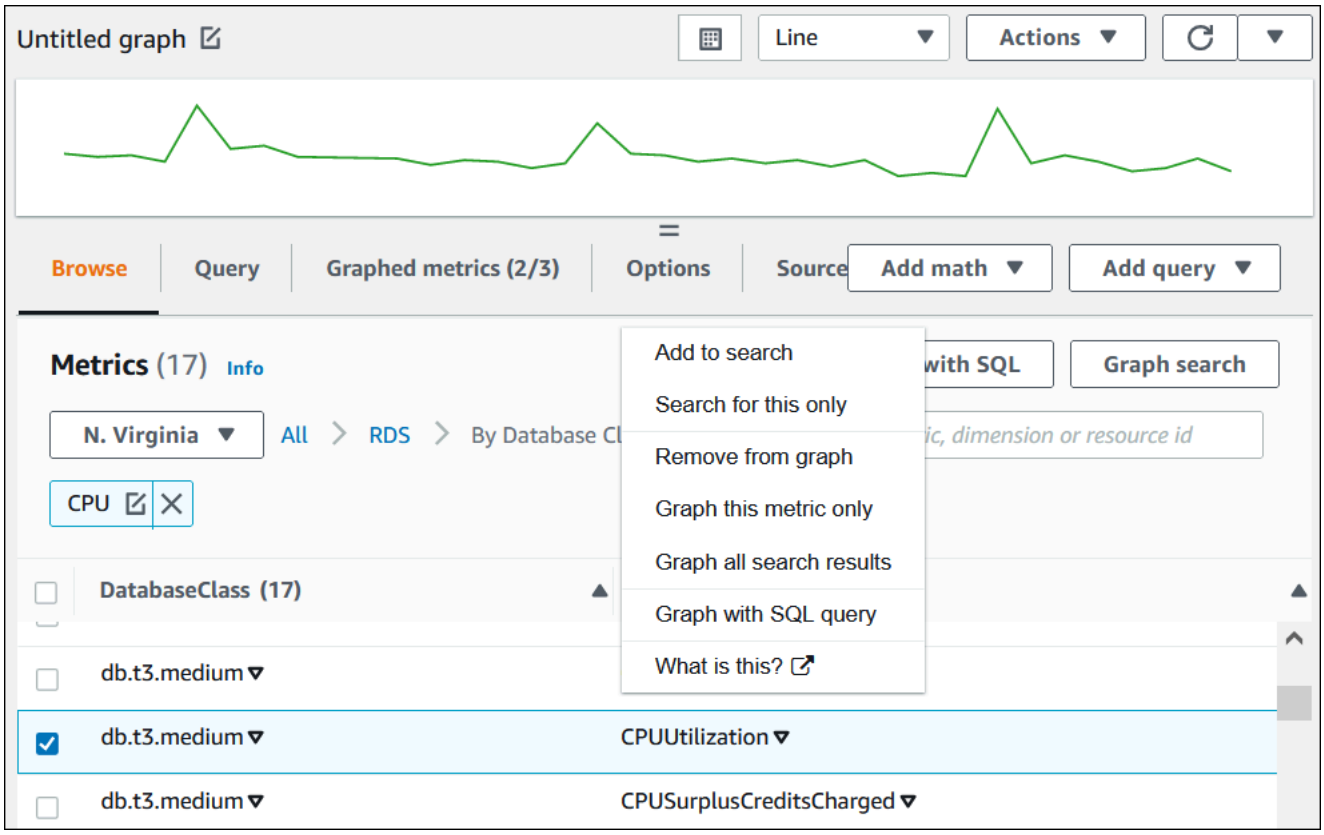

5. Pilih dimensi metrik, misalnya Berdasarkan Kelas Basis Data.




6. Lakukan salah satu tindakan berikut:

   - Untuk mengurutkan metrik, gunakan judul kolom.
   - Untuk membuat grafik metrik, pilih kotak centang di samping metrik.
   - Untuk memfilter berdasarkan sumber daya, pilih ID sumber daya, lalu pilih Tambahkan ke pencarian.
   - Untuk memfilter berdasarkan metrik, pilih nama metrik, lalu pilih Tambahkan ke pencarian.

Contoh berikut menyaring pada kelas db.t3.medium dan grafik metrik. CPUUtilization




Anda dapat menemukan detail tentang cara menganalisis penggunaan sumber daya untuk Aurora SQL Postgre menggunakan metrik. CloudWatch Untuk informasi selengkapnya, silakan lihat Menggunakan CloudWatch metrik Amazon untuk menganalisis penggunaan sumber daya untuk Aurora PostgreSQL

AWS CLI

Untuk mendapatkan informasi metrik dengan menggunakan AWS CLI, gunakan CloudWatch perintah `list-metrics`. Dalam contoh berikut, Anda mencantumkan semua metrik di namespace `AWS/RDS`.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

Untuk mendapatkan data metrik, gunakan perintah `get-metric-data`.

Contoh berikut mendapatkan `CPUUtilization` statistik misalnya `my-instance` selama periode 24 jam tertentu, dengan perincian 5 menit.

Buat JSON file `CPU_metric.json` dengan konten berikut.

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
   "EndTime" : "2023-12-26T00:00:00Z",
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

Untuk Linux, macOS, atau Unix:

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

Untuk Windows:

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

Contoh output muncul sebagai berikut:

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
```

```
                "2023-12-15T23:24:00+00:00",
                ...
            ],
            "Values": [
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

Untuk informasi selengkapnya, lihat Mendapatkan statistik untuk metrik di Panduan CloudWatch Pengguna Amazon.

## Mengekspor metrik Performance Insights ke CloudWatch

Performance Insights memungkinkan Anda mengekspor dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom untuk instans DB Anda ke Amazon. CloudWatch Anda dapat mengekspor dasbor metrik sebagai dasbor baru atau menambahkannya ke CloudWatch dasbor yang ada. Saat Anda memilih untuk menambahkan dasbor ke CloudWatch dasbor yang ada, Anda dapat membuat label header sehingga metrik muncul di bagian terpisah di CloudWatch dasbor.

Anda dapat melihat dasbor metrik yang diekspor di konsol. CloudWatch Jika Anda menambahkan metrik baru ke dasbor metrik Performance Insights setelah mengekspornya, Anda harus mengekspor dasbor ini lagi untuk melihat metrik baru di konsol. CloudWatch

Anda juga dapat memilih widget metrik di dasbor Performance Insights dan melihat data metrik di konsol. CloudWatch

Untuk informasi selengkapnya tentang melihat metrik di CloudWatch konsol, lihatMelihat metrik cluster DB di CloudWatch konsol dan AWS CLI.

Di bagian berikut, ekspor metrik Performance Insights ke CloudWatch dasbor baru atau yang sudah ada, lalu lihat metrik Performance Insights. CloudWatch

Topik

- Mengekspor metrik Performance Insights sebagai dasbor baru CloudWatch
- Menambahkan metrik Performance Insights ke dasbor yang ada CloudWatch
- Melihat widget metrik Performance Insights di CloudWatch

## Mengekspor metrik Performance Insights sebagai dasbor baru CloudWatch

Pilih dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom dari dasbor Performance Insights dan ekspor sebagai dasbor baru. CloudWatch Anda dapat melihat dasbor yang diekspor di CloudWatch konsol.

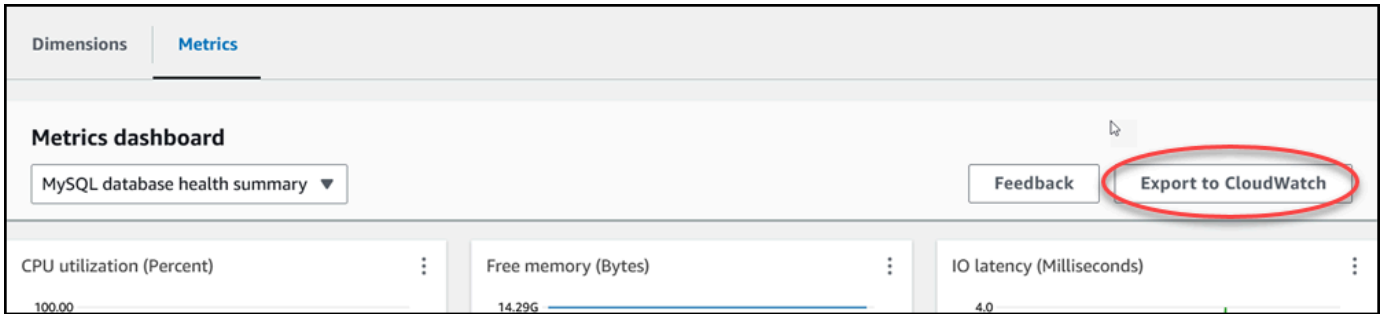
Untuk mengekspor dasbor metrik Performance Insights sebagai dasbor baru ke CloudWatch

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

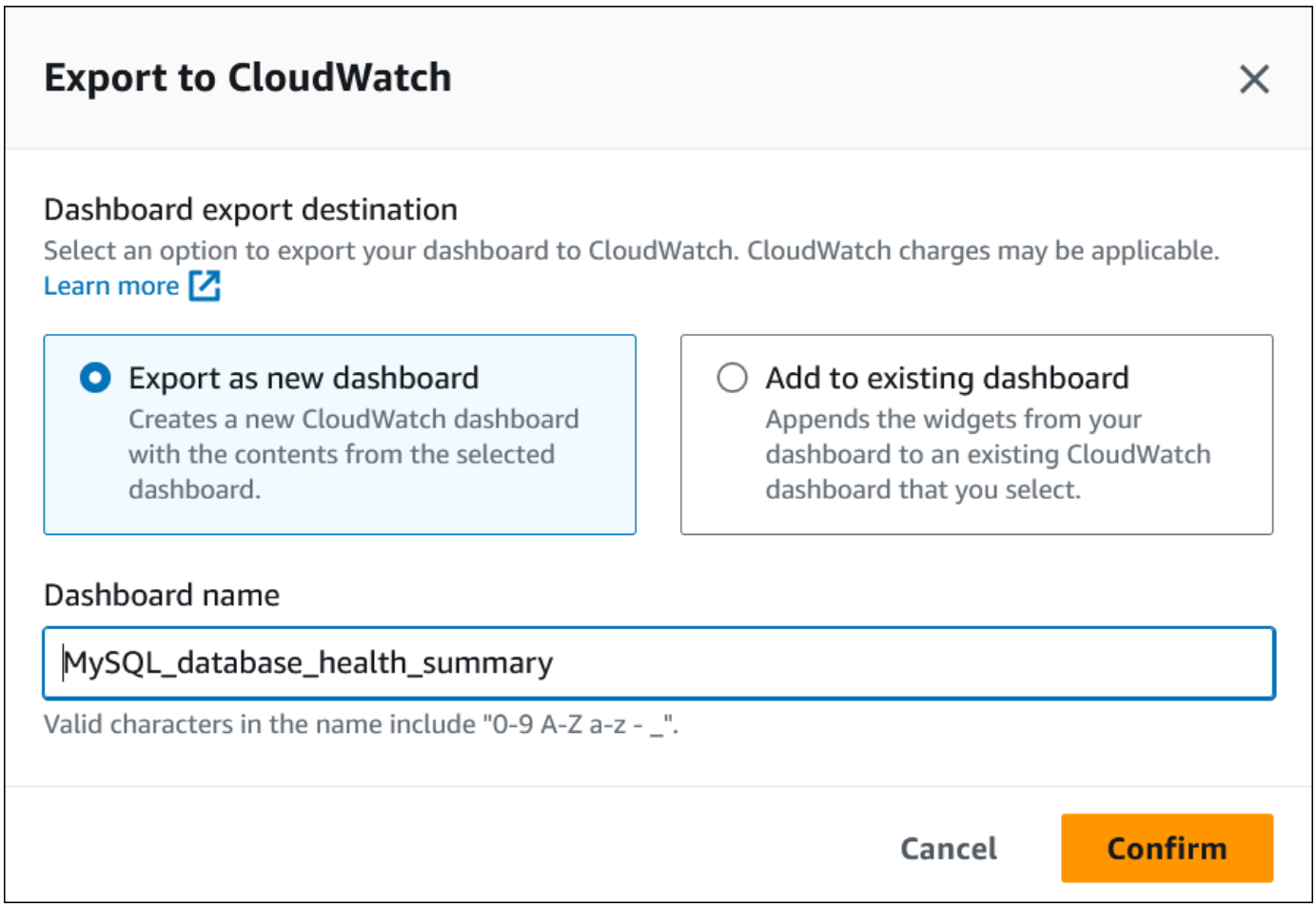
   Muncul dasbor Wawasan Performa untuk instans DB tersebut.
4. Gulir ke bawah dan pilih Metrik.

   Secara default, dasbor yang telah dikonfigurasi sebelumnya dengan metrik Wawasan Performa akan muncul.
5. Pilih dasbor yang telah dikonfigurasi atau kustom, lalu pilih Ekspor ke CloudWatch.

   CloudWatchJendela Ekspor ke muncul.

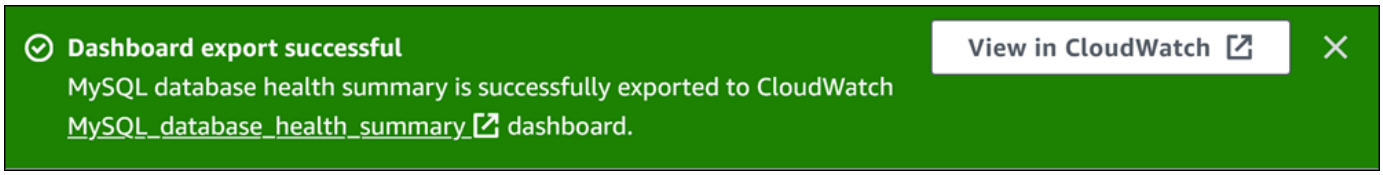



6. Pilih Ekspor sebagai dasbor baru.




7. Masukkan nama dasbor baru di kolom Nama dasbor dan pilih Konfirmasi.

   Banner menampilkan pesan setelah dasbor berhasil diekspor.




8. Pilih tautan atau Lihat CloudWatch di spanduk untuk melihat dasbor metrik di CloudWatch konsol.

## Menambahkan metrik Performance Insights ke dasbor yang ada CloudWatch

Tambahkan dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom ke dasbor yang ada CloudWatch . Anda dapat menambahkan label ke dasbor metrik untuk muncul di bagian terpisah di CloudWatch dasbor.

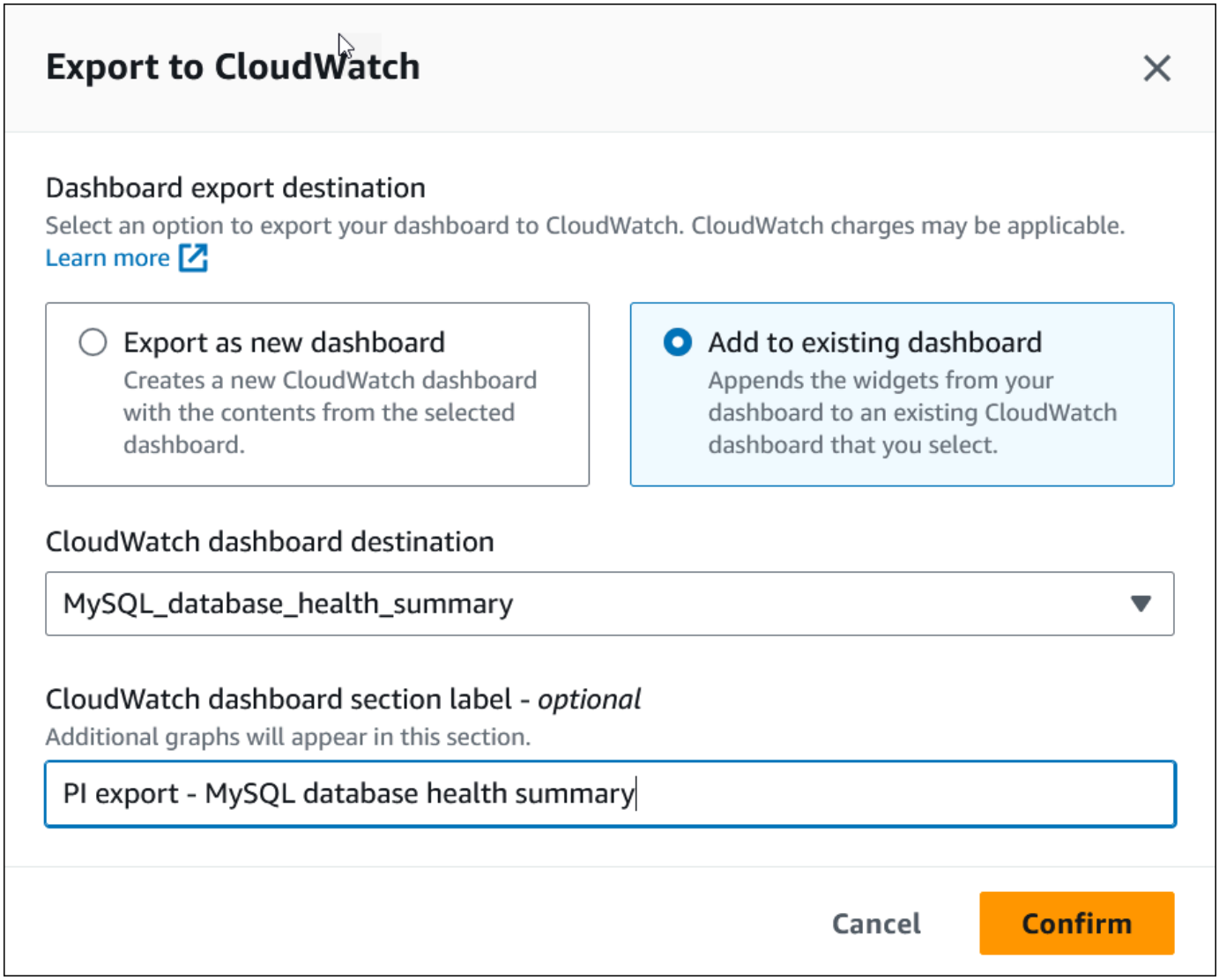
Untuk mengekspor metrik ke dasbor yang ada CloudWatch

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Muncul dasbor Wawasan Performa untuk instans DB tersebut.
4. Gulir ke bawah dan pilih Metrik.

   Secara default, dasbor yang telah dikonfigurasi sebelumnya dengan metrik Wawasan Performa akan muncul.
5. Pilih dasbor yang telah dikonfigurasi atau kustom, lalu pilih Ekspor ke CloudWatch.

   CloudWatchJendela Ekspor ke muncul.
6. Pilih Tambahkan ke dasbor yang ada.




7. Tentukan tujuan dan label dasbor, lalu pilih Konfirmasi.

   - CloudWatch tujuan dasbor - Pilih CloudWatch dasbor yang ada.
   - CloudWatch label bagian dasbor - opsional - Masukkan nama untuk metrik Performance Insights untuk muncul di bagian ini di dasbor. CloudWatch

   Banner menampilkan pesan setelah dasbor berhasil diekspor.

8. Pilih tautan atau Lihat CloudWatch di spanduk untuk melihat dasbor metrik di CloudWatch konsol.

## Melihat widget metrik Performance Insights di CloudWatch

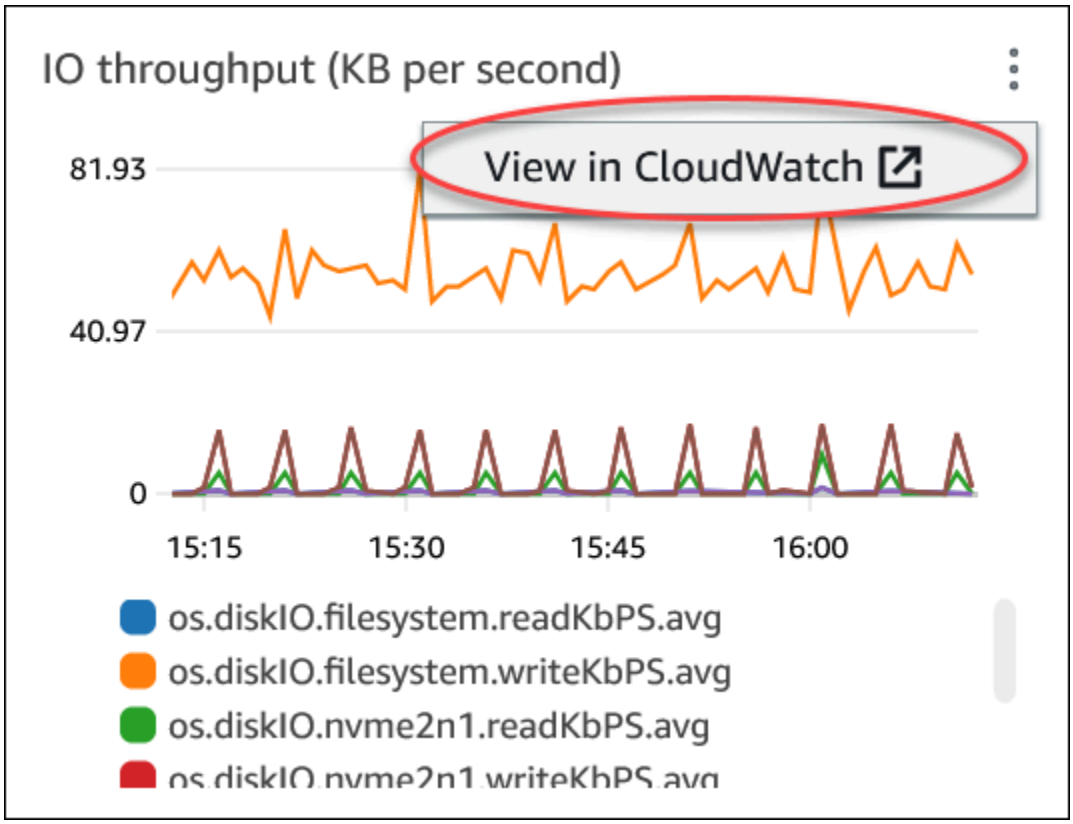
Pilih widget metrik Performance Insights di dasbor Amazon RDS Performance Insights dan lihat data metrik di konsol. CloudWatch

Untuk mengekspor widget metrik dan melihat data metrik di konsol CloudWatch

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Muncul dasbor Wawasan Performa untuk instans DB tersebut.
4. Gulir ke bawah ke Metrik.

   Secara default, dasbor yang telah dikonfigurasi sebelumnya dengan metrik Wawasan Performa akan muncul.
5. Pilih widget metrik dan kemudian pilih Lihat CloudWatch di dalam menu.




   Data metrik muncul di CloudWatch konsol.

# Membuat CloudWatch alarm untuk memantau Aurora

Anda dapat membuat CloudWatch alarm yang mengirimkan SNS pesan Amazon saat alarm berubah status. Alarm mengawasi metrik tunggal selama periode waktu yang Anda tentukan. Alarm tersebut juga dapat melakukan satu atau beberapa tindakan berdasarkan nilai metrik yang relatif terhadap ambang batas tertentu selama beberapa periode waktu. Tindakan tersebut adalah pemberitahuan yang dikirim ke SNS topik Amazon atau kebijakan EC2 Auto Scaling Amazon.

Alarm memanggil tindakan untuk perubahan status berkelanjutan saja. CloudWatch alarm tidak memanggil tindakan hanya karena mereka berada dalam keadaan tertentu. Status harus diubah dan dipertahankan selama jangka waktu tertentu.

> **Note**
>
> Untuk Aurora, gunakan metrik peran `WRITER` atau `READER` untuk menyiapkan alarm, bukan mengandalkan metrik untuk instans DB tertentu. Peran instans Aurora DB dapat mengubah peran dari waktu ke waktu. Anda dapat menemukan metrik berbasis peran ini di konsol. CloudWatch
> Aurora Auto Scaling mengatur alarm secara otomatis berdasarkan metrik peran `READER`. Untuk informasi selengkapnya tentang Aurora Auto Scaling, lihat Auto Scaling Amazon Aurora dengan Replika Aurora.

Anda dapat menggunakan fungsi matematika INSIGHTS metrik DB_ PERF _ di CloudWatch konsol untuk menanyakan Amazon RDS untuk metrik penghitung Performance Insights. INSIGHTSFungsi DB_ PERF _ juga mencakup DBLoad metrik pada interval sub-menit. Anda dapat mengatur CloudWatch alarm pada metrik ini.

Untuk detail selengkapnya tentang cara membuat alarm, lihat Membuat alarm di metrik penghitung Performance Insights dari database. AWS

Untuk mengatur alarm menggunakan AWS CLI

- Panggil `put-metric-alarm`. Untuk informasi selengkapnya, lihat Referensi Perintah AWS CLI.

Untuk mengatur alarm menggunakan CloudWatch API

- Panggil `PutMetricAlarm`. Untuk informasi selengkapnya, lihat CloudWatch APIReferensi Amazon

Untuk informasi selengkapnya tentang mengatur SNS topik Amazon dan membuat alarm, lihat Menggunakan CloudWatch alarm Amazon.

# Memantau database Amazon Aurora dengan Database CloudWatch Insights

Pantau beban database (Beban DB) untuk armada instans Amazon Aurora DB Amazon Aurora Anda dengan Database Insights. DB Load mengukur tingkat aktivitas sesi dalam database Anda. Anda dapat menggunakan Database Insights untuk menganalisis dan memecahkan masalah kinerja database Amazon Aurora Amazon RDS Aurora Anda dalam skala besar.

Dengan Database Insights, Anda dapat memvisualisasikan Beban DB pada armada database Anda dan memfilter beban dengan menunggu, pernyataan SQL, host, atau pengguna.

Secara default, RDS mengaktifkan mode Standar Wawasan Database untuk database Amazon Aurora RDS Anda. Saat Anda mengaktifkan mode Advanced Database Insights untuk cluster DB, RDS mengaktifkan Database Insights untuk setiap instans DB di cluster.

Untuk informasi tentang penggunaan Wawasan Database di CloudWatch konsol Amazon, lihat Wawasan CloudWatch Database di  CloudWatch Panduan Pengguna Amazon.

## Harga

Untuk informasi tentang harga, lihat  CloudWatch Harga Amazon.

Topik

- Amazon Aurora DB engine, Wilayah, dan dukungan kelas instans untuk Database Insights
- Mengaktifkan mode Lanjut Wawasan Database untuk
- Mengaktifkan mode Standar Wawasan Database untuk
- 
- Pertimbangan untuk Wawasan Database untuk Amazon Aurora

## Amazon Aurora DB engine, Wilayah, dan dukungan kelas instans untuk Database Insights

Tabel berikut menyediakan mesin Aurora DB yang mendukung Database Insights.

| Mesin DB Amazon Aurora | Versi mesin dan Wilayah yang didukung | Pembatasan kelas instans |
|---|---|---|
| Edisi yang Kompatibel dengan Amazon Aurora MySQL | Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah Database Insights dengan Aurora MySQL, lihat. Wawasan Performa dengan Aurora MySQL | Database Insights memiliki batasan kelas mesin berikut:<br>• db.t2 – Tidak didukung<br>• db.t3 – Tidak didukung<br>• db.t4g.micro dan db.t4g.small – Tidak didukung |
| Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL | Untuk informasi lebih lanjut tentang versi dan ketersediaan Wilayah Database Insights dengan Aurora PostgreSQL, lihat. Wawasan Performa dengan Aurora PostgreSQL | Tidak berlaku |
| Database Tanpa Batas Aurora PostgreSQL | Untuk informasi selengkapnya tentang menggunakan Database Insights dengan Aurora PostgreSQL Limitless Database, lihat. Memantau Database Tanpa Batas Aurora PostgreSQL dengan Wawasan Database CloudWatch | Tidak berlaku |

Database Insights mendukungAmazon Aurora Serverless v2.

## Amazon Aurora DB engine, Wilayah, dan dukungan kelas instans untuk fitur Database Insights

Tabel berikut menyediakan mesin Amazon Aurora DB yang mendukung fitur Database Insights.

| Fitur | Tingkat harga | Wilayah yang didukung | Mesin DB yang didukung | Kelas instans yang didukung |
|---|---|---|---|---|
| Statistik SQL untuk Wawasan Performa | Semua | Semua | Semua | Semua |
| Menganalisis performa basis data selama periode waktu tertentu | Khusus tingkat berbayar | Semua | Semua | Semua kecuali db.serverless (Aurora Serverless v2) |
| Melihat rekomendasi proaktif Performance Insights | Khusus tingkat berbayar | • AS Timur (Ohio)<br>• AS Timur (Virginia Utara)<br>• AS Barat (California Utara)<br>• AS Barat (Oregon)<br>• Asia Pasifik (Mumbai)<br>• Asia Pasifik (Seoul)<br>• Asia Pasifik (Singapura)<br>• Asia Pasifik (Sydney)<br>• Asia Pasifik (Tokyo) | Semua | Semua kecuali db.serverless (Aurora Serverless v2) |

| Fitur | Tingkat harga | Wilayah yang didukung | Mesin DB yang didukung | Kelas instans yang didukung |
|---|---|---|---|---|
| | | • Kanada (Pusat)<br>• Eropa (Frankfurt)<br>• Eropa (Irlandia)<br>• Eropa (London)<br>• Eropa (Paris)<br>• Eropa (Stockholm)<br>• Amerika Selatan (Sao Paulo) | | |

## Amazon Aurora Region dukungan untuk Database Insights

Aurora Amazon mendukung Database Insights berikut ini. Wilayah AWS

- Timur AS (N. Virginia)
- AS Timur (Ohio)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Afrika (Cape Town)
- Asia Pasifik (Hong Kong)
- Asia Pasifik (Hyderabad)
- Asia Pasifik (Jakarta)
- Asia Pasifik (Malaysia)
- Asia Pasifik (Melbourne)
- Asia Pasifik (Mumbai)

- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- (Canada (Central)
- Kanada Barat (Calgary)
- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Milan)
- Eropa (Paris)
- Eropa (Spanyol)
- Eropa (Stockholm)
- Eropa (Zürich)
- Israel (Tel Aviv)
- Timur Tengah (Bahrain)
- Timur Tengah (UEA)
- Amerika Selatan (Sao Paulo)
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

## Mengaktifkan mode Lanjut Wawasan Database untuk

Untuk mengaktifkan mode Lanjut Wawasan Database untuk Aurora, gunakan prosedur berikut.

### Mengaktifkan mode Advanced Database Insights saat membuat

Aktifkan mode Lanjutan Wawasan Database saat membuat database untuk Aurora.

Console

Di konsol, Anda dapat mengaktifkan mode Advanced Database Insights saat membuat cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Advanced Database Insights saat membuat menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih Buat basis data.
4. Di bagian Database Insights, pilih Mode lanjutan. Kemudian, pilih opsi berikut:
   - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Periode retensi harus 15 bulan untuk mode Advanced Database Insights.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Buat basis data.

AWS CLI

Untuk mengaktifkan mode Advanced Database Insights saat membuat cluster DB, panggil create-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--database-insights-mode advanced`untuk mengaktifkan mode Advanced Database Insights.
- `--engine`— Mesin database untuk cluster DB.
- `--db-cluster-identifier` — Pengidentifikasi untuk cluster DB .
- `--enable-performance-insights`untuk mengaktifkan Performance Insights untuk Database Insights.
- `--performance-insights-retention-period`— Periode retensi untuk data untuk Anda. Untuk mengaktifkan Database Insights, periode retensi harus minimal 465 hari.

Contoh berikut memungkinkan mode Advanced Database Insights saat membuat cluster DB.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --database-insights-mode advanced \
```

```
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --database-insights-mode advanced ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

Untuk mengaktifkan mode Advanced Database Insights saat Anda membuat , tentukan parameter berikut untuk operasi Amazon RDS API.

- `DatabaseInsightsMode` untuk `advanced`
- `EnablePerformanceInsights` untuk `True`
- `PerformanceInsightsRetentionPeriod`untuk setidaknya 465 hari

## Mengaktifkan mode Advanced Database Insights saat memodifikasi

Aktifkan Wawasan Database saat memodifikasi database untuk Amazon . Memodifikasi untuk mengaktifkan mode Advanced Database Insights tidak menyebabkan downtime.

> **Note**
>
> Untuk mengaktifkan Database Insights, setiap instans DB dalam cluster DB harus memiliki pengaturan Performance Insights dan Enhanced Monitoring yang sama.

Console

Di konsol, Anda dapat mengaktifkan mode Advanced Database Insights saat Anda memodifikasi cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Advanced Database Insights saat memodifikasi menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih instans DB, dan pilih Ubah.
4. Di bagian Database Insights, pilih Mode lanjutan. Kemudian, pilih opsi berikut:
   - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Periode retensi harus 15 bulan untuk mode Advanced Database Insights.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih Langsung terapkan. Jika Anda memilih Terapkan selama jendela pemeliharaan terjadwal berikutnya, database Anda akan mengabaikan pengaturan ini dan segera mengaktifkan mode Advanced Database Insights.
7. Pilih Modify cluster .

AWS CLI

Untuk mengaktifkan mode Advanced Database Insights saat memodifikasi cluster DB, panggil modify-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--database-insights-mode advanced`untuk mengaktifkan mode Advanced Database Insights.
- `--db-cluster-identifier` — Pengidentifikasi untuk cluster DB .
- `--enable-performance-insights`untuk mengaktifkan Performance Insights untuk Database Insights.
- `--performance-insights-retention-period`— Periode retensi untuk data untuk Anda. Untuk mengaktifkan mode Advanced Database Insights, periode retensi harus minimal 465 hari.

Contoh berikut memungkinkan mode Advanced Database Insights saat memodifikasi instans DB cluster .

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

Untuk mengaktifkan mode Advanced Database Insights saat Anda memodifikasi , tentukan parameter berikut untuk operasi Amazon RDS API Anda.

- `DatabaseInsightsMode` untuk `advanced`
- `EnablePerformanceInsights` untuk `True`
- `PerformanceInsightsRetentionPeriod`untuk setidaknya 465 hari

## Mengaktifkan mode Standar Wawasan Database untuk

Untuk mengaktifkan mode Standar Wawasan Database untuk Aurora, gunakan prosedur berikut.

### Mengaktifkan mode Standar Database Insights saat membuat

Aktifkan mode Standar Wawasan Database saat membuat database untuk Aurora.

Console

Di konsol, Anda dapat mengaktifkan mode Standar Database Insights saat membuat cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Standar Database Insights saat membuat menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih Buat basis data.
4. Di bagian Database Insights, pilih Mode standar. Kemudian, pilih dari opsi berikut untuk mengaktifkan atau menonaktifkan Performance Insights:
   - Untuk menonaktifkan Performance Insights, batalkan pilihan Aktifkan Performance Insights.
   - Untuk mengaktifkan Performance Insights, pilih Aktifkan Performance Insights. Untuk mengonfigurasi Performance Insights, tentukan opsi berikut:
     - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Periode retensi harus minimal 7 hari.
     - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Buat basis data.

AWS CLI

Untuk mengaktifkan mode Standar Database Insights saat membuat cluster DB, panggil create-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--database-insights-mode standard`untuk mengaktifkan mode Standar Wawasan Database.
- `--engine`— Mesin database untuk cluster DB.
- `--db-cluster-identifier` — Pengidentifikasi untuk cluster DB .
- `--enable-performance-insights`atau `--no-enable-performance-insights` untuk mengaktifkan atau menonaktifkan Performance Insights. Jika Anda menentukan`--enable-performance-insights`, Anda juga harus menentukan `--performance-insights-retention-period` — Periode retensi untuk data untuk Anda. Periode retensi harus minimal 7 hari.

Contoh berikut memungkinkan mode Standar Database Insights dan Performance Insights saat membuat instans DB DB.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

Untuk mengaktifkan mode Standar Database Insights saat Anda membuat , tentukan parameter berikut untuk operasi Amazon RDS API Anda.

- `DatabaseInsightsMode` untuk `standard`
- `EnablePerformanceInsights`ke `True` atau`False`. Jika Anda mengatur `EnablePerformanceInsights` ke`True`, Anda harus mengatur `PerformanceInsightsRetentionPeriod` ke setidaknya 7 hari.

## Mengaktifkan mode Standar Database Insights saat memodifikasi

Aktifkan mode Standar Wawasan Database saat memodifikasi database untuk Amazon . Memodifikasi untuk mengaktifkan mode Standar Database Insights tidak menyebabkan downtime.

> **Note**
>
> Untuk mengaktifkan Database Insights, setiap instans DB di cluster DB harus memiliki pengaturan Performance Insights dan Enhanced Monitoring yang sama.

Console

Di konsol, Anda dapat mengaktifkan mode Standar Database Insights saat Anda memodifikasi cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Standar Database Insights saat memodifikasi menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih instans DB, dan pilih Ubah.
4. Di bagian Database Insights, pilih Mode standar. Kemudian, pilih dari opsi berikut:
   - Untuk menonaktifkan Performance Insights, batalkan pilihan Aktifkan Performance Insights.
   - Untuk mengaktifkan Performance Insights, pilih Aktifkan Performance Insights. Untuk mengonfigurasi Performance Insights, tentukan opsi berikut:
     - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Periode retensi harus minimal 7 hari.
     - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat

dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.

5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih Langsung terapkan. Jika Anda memilih Terapkan selama jendela pemeliharaan terjadwal berikutnya, database Anda akan mengabaikan pengaturan ini dan segera mengaktifkan mode Standar Database Insights.
7. Pilih Modify cluster .

AWS CLI

Untuk mengaktifkan mode Standar Database Insights saat memodifikasi cluster DB, panggil modify-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--database-insights-mode standard`untuk mengaktifkan mode Standar Wawasan Database.
- `--db-cluster-identifier` — Pengidentifikasi untuk cluster DB .
- `--enable-performance-insights`atau `--no-enable-performance-insights` untuk mengaktifkan atau menonaktifkan Performance Insights. Jika Anda menentukan`--enable-performance-insights`, Anda juga harus menentukan `--performance-insights-retention-period` — Periode retensi untuk data untuk Anda. Periode retensi harus minimal 7 hari.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
```

```
    --performance-insights-retention-period 7
```

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

Untuk mengaktifkan mode Standar Database Insights saat Anda memodifikasi , tentukan parameter berikut untuk operasi Amazon RDS API Anda.

- `DatabaseInsightsMode` untuk `standard`
- `EnablePerformanceInsights`ke `True` atau`False`. Jika Anda mengatur `EnablePerformanceInsights` ke`True`, Anda harus mengatur `PerformanceInsightsRetentionPeriod` ke setidaknya 7 hari.

Untuk memantau kueri SQL yang lambat untuk database Anda, Anda dapat menggunakan bagian Kueri SQL Lambat di dasbor Database Insights. Sebelum mengonfigurasi database Anda untuk memantau kueri SQL yang lambat, bagian Kueri SQL Lambat kosong.

Untuk informasi selengkapnya tentang memantau kueri SQL yang lambat di dasbor Database Insights, lihat Melihat Dasbor Instans Database untuk Wawasan CloudWatch Database di Panduan Pengguna Amazon. CloudWatch

Untuk mengonfigurasi database Anda untuk memantau kueri SQL yang lambat dengan Database Insights, selesaikan langkah-langkah berikut:

1. Aktifkan ekspor log ke CloudWatch Log.

2. Buat atau modifikasi grup parameter cluster DB untuk grup parameter DB cluster .

Untuk informasi tentang mengonfigurasi ekspor log, lihat Amazon Aurora.

Untuk membuat atau memodifikasi grup parameter cluster DB Anda, lihat topik berikut.

- Membuat grup parameter cluster DB di Amazon Aurora
- Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora

Amazon Aurora MySQL

Untuk mengonfigurasi kluster DB MySQL Amazon Aurora Anda untuk memantau kueri SQL yang lambat, setel parameter berikut.

- `slow_query_log`— diatur ke `1`
- `long_query_time`— diatur ke `1.0`
- `log_output`— diatur ke `FILE`

Amazon Aurora PostgreSQL

Untuk mengonfigurasi klaster DB PostgreSQL Amazon Aurora Anda untuk memantau kueri SQL yang lambat, setel parameter berikut. Perhatikan bahwa menyetel parameter ini dapat mengurangi kinerja cluster DB Anda.

- `log_min_duration_statement`— diatur ke `1000`
- `log_statement`— diatur ke `none`
- `log_destination`— diatur ke `stderr`

> **Note**
>
> Untuk , Anda dapat mengkonfigurasi parameter dengan granularitas 1-mikrodetik. `long_query_time` Misalnya, Anda dapat mengatur parameter ini ke`0.000001`. Bergantung pada jumlah kueri pada instans DB, nilai parameter `long_query_time` dapat mengurangi kinerja. Mulailah dengan nilainya`1.0`, dan sesuaikan berdasarkan beban kerja Anda. Saat Anda mengatur parameter ini`0`, Database Insights mencatat semua kueri.

Untuk informasi tentang log Aurora MySQL dan Aurora PostgreSQL, lihat berikut ini.

- File log basis data Aurora MySQL
- File log basis data Aurora PostgreSQL

## Pertimbangan untuk Wawasan Database untuk Amazon Aurora

Berikut ini adalah pertimbangan untuk Database Insights untuk Amazon .

- Anda tidak dapat mengelola Database Insights untuk instans DB di cluster DB multi-AZ cluster .
- Untuk mengaktifkan mode Advanced Database Insights, Anda harus mengaktifkan Performance Insights dan menetapkan periode retensi Performance Insights setidaknya 465 hari (15 bulan). Tidak ada biaya tambahan untuk menetapkan periode retensi Performance Insights menjadi 15 bulan selain biaya Database Insights. Untuk informasi tentang harga untuk Wawasan Database, lihat CloudWatch Harga Amazon.
- Untuk mengaktifkan Database Insights, setiap instans DB di cluster DB harus memiliki pengaturan Performance Insights dan Enhanced Monitoring yang sama.
- Memodifikasi untuk mengaktifkan salah satu mode Database Insights tidak menyebabkan downtime.

# Memantau muatan DB dengan Wawasan Performa di Amazon Aurora

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Wawasan Performa memperluas fitur pemantauan Amazon Aurora yang sudah ada untuk mengilustrasikan dan membantu Anda menganalisis performa klaster. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan muatan basis data pada muatan klaster Amazon Aurora dan memfilter muatan menurut peristiwa tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi tentang cara menggunakan Wawasan Performa dengan Amazon DocumentDB, lihat Panduan Developer Amazon DocumentDB.

Topik

- Ikhtisar Wawasan Performa tentang Amazon Aurora

- Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora
- Ikhtisar Skema Kinerja untuk Performance Insights di Aurora
- Mengonfigurasi kebijakan akses untuk Wawasan Performa
- Menganalisis metrik dengan dasbor Wawasan Performa
- Melihat rekomendasi proaktif Performance Insights
- Mengambil metrik dengan Performance API Insights untuk Aurora
- Pencatatan panggilan Performance Insights menggunakan AWS CloudTrail
- Performance Insights API dan titik VPC akhir antarmuka ()AWS PrivateLink

## Ikhtisar Wawasan Performa tentang Amazon Aurora

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Secara default, RDS mengaktifkan Performance Insights di wizard pembuatan konsol untuk semua engine Amazon RDS. Jika Anda mengaktifkan Performance Insights di tingkat cluster DB, RDS mengaktifkan Performance Insights untuk setiap instans DB di cluster.Jika Anda memiliki lebih dari satu basis data di instans DB, Wawasan Performa akan menggabungkan data performa.

Anda dapat menemukan ikhtisar Wawasan Performa untuk Amazon Aurora dalam video berikut.

Menggunakan Wawasan Performa untuk Menganalisis Performa Amazon Aurora PostgreSQL

Topik

- Muatan basis data
- Maksimum CPU
- Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk Wawasan Performa
- Harga dan retensi data untuk Wawasan Performa

## Muatan basis data

Beban basis data (beban DB) mengukur tingkat aktivitas sesi dalam database Anda. `DBLoad`adalah metrik kunci dalam Performance Insights, dan Performance Insights mengumpulkan beban DB setiap detik.

Topik

- Sesi aktif
- Sesi aktif rata-rata
- Eksekusi aktif rata-rata
- Dimensi

### Sesi aktif

Sesi basis data mewakili dialog aplikasi dengan basis data relasional. Sesi aktif adalah koneksi yang mengirimkan tugas ke mesin DB dan sedang menunggu tanggapan.

Sesi dianggap aktif jika berjalan di CPU atau menunggu sumber daya tersedia sehingga dapat dilanjutkan. Misalnya, sesi aktif mungkin menunggu halaman (atau blok) dibaca ke dalam memori, dan kemudian menggunakan CPU saat membaca data dari halaman.

## Sesi aktif rata-rata

Sesi aktif rata-rata (AAS) adalah unit untuk metrik `DBLoad` dalam Wawasan Performa. Ini mengukur berapa banyak sesi yang aktif secara bersamaan di basis data.

Setiap detik, Wawasan Performa mengambil sampel jumlah sesi yang secara bersamaan menjalankan kueri. Untuk setiap sesi aktif, Wawasan Performa mengumpulkan data berikut:

- Pernyataan SQL
- Status sesi (berjalan pada CPU atau menunggu)
- Host
- Pengguna yang menjalankan SQL

Wawasan Performa menghitung AAS dengan membagi jumlah total sesi dengan jumlah sampel selama periode waktu tertentu. Misalnya, tabel berikut menunjukkan 5 sampel berturut-turut dari kueri yang berjalan yang diambil dengan interval 1 detik.

| Sampel | Jumlah sesi yang menjalankan kueri | AAS | Penghitungan |
|---|---|---|---|
| 1 | 2 | 2 | Total 2 sesi/1 sampel |
| 2 | 0 | 1 | Total 2 sesi/2 sampel |
| 3 | 4 | 2 | Total 6 sesi/3 sampel |
| 4 | 0 | 1.5 | Total 6 sesi/4 sampel |
| 5 | 4 | 2 | Total 10 sesi/5 sampel |

Pada contoh sebelumnya, muatan DB untuk interval waktu tersebut adalah 2 AAS. Pengukuran ini berarti bahwa rata-rata ada 2 sesi aktif pada waktu tertentu selama interval tersebut ketika 5 sampel diambil.

Eksekusi aktif rata-rata

Eksekusi aktif rata-rata (AAE) per detik berkaitan dengan AAS. Untuk menghitung AAE, Wawasan Performa membagi total waktu eksekusi kueri dengan interval waktu. Tabel berikut menunjukkan penghitungan AAE untuk kueri yang sama dalam tabel sebelumnya.

| Waktu yang telah berlalu (dtk) | Total waktu eksekusi (detik) | AAE | Penghitungan |
|---|---|---|---|
| 60 | 120 | 2 | 120 detik eksekusi/60 detik berlalu |
| 120 | 120 | 1 | 120 detik eksekusi/120 detik berlalu |
| 180 | 380 | 2.11 | 380 detik eksekusi/180 detik berlalu |
| 240 | 380 | 1,58 | 380 detik eksekusi/240 detik berlalu |
| 300 | 600 | 2 | 600 detik eksekusi/300 detik berlalu |

Dalam kebanyakan kasus, AAS dan AAE untuk sebuah kueri kira-kira sama. Namun, karena input ke penghitungan berupa sumber data yang berbeda, penghitungannya sering sedikit berbeda.

Dimensi

Metrik `db.load` berbeda dengan metrik deret waktu lainnya karena Anda dapat membaginya menjadi beberapa sub-komponen yang disebut dimensi. Anda dapat menganggap dimensi sebagai kategori “potong menurut” untuk berbagai karakteristik metrik `DBLoad`.

Saat Anda mendiagnosis masalah performa, dimensi berikut sering kali paling berguna:

Topik

- Peristiwa tunggu
- SQL Teratas

Untuk daftar lengkap dimensi untuk mesin Aurora, lihat Muatan DB diiris berdasarkan dimensi.

### Peristiwa tunggu

Peristiwa tunggu menyebabkan pernyataan SQL menunggu peristiwa tertentu terjadi sebelum dapat terus berjalan. Peristiwa tunggu adalah dimensi penting, atau kategori, untuk muatan DB karena menunjukkan di mana pekerjaan terhambat.

Setiap sesi aktif berjalan di CPU atau menunggu. Misalnya, sesi menggunakan CPU ketika mencari memori untuk buffer, melakukan penghitungan, atau menjalankan kode prosedural. Ketika tidak menggunakan CPU, sesi mungkin menunggu buffer memori menjadi kosong, file data dibaca, atau log untuk ditulis. Semakin banyak waktu untuk sesi menunggu sumber daya, semakin sedikit waktu untuk sesi dijalankan di CPU.

Ketika Anda menyetel basis data, Anda sering mencoba mencari tahu sumber daya yang sedang menunggu sesi. Misalnya, dua atau tiga peristiwa tunggu mungkin menyumbang 90 persen dari muatan DB. Ukuran ini berarti bahwa, rata-rata, sesi aktif menghabiskan sebagian besar waktunya menunggu sejumlah kecil sumber daya. Jika Anda dapat mengetahui penyebab peristiwa tunggu ini, Anda dapat mencoba solusinya.

Peristiwa tunggu bervariasi berdasarkan mesin DB:

- Untuk daftar peristiwa tunggu yang umum terjadi di Aurora MySQL, lihat Aurora Acara tunggu saya SQL. Untuk mempelajari cara menyetel menggunakan peristiwa tunggu ini, lihat Menyesuaikan Aurora MySQL.
- Untuk informasi tentang semua peristiwa tunggu MySQL, lihat Wait Event Summary Tables dalam dokumentasi MySQL.
- Untuk daftar peristiwa tunggu yang umum terjadi di Aurora PostgreSQL, lihat Peristiwa tunggu Amazon Aurora PostgreSQL. Untuk mempelajari cara menyetel menggunakan peristiwa tunggu ini, lihat Menyetel dengan peristiwa tunggu di Aurora PostgreSQL.
- Untuk informasi tentang semua peristiwa tunggu PostgreSQL, lihat The Statistics Collector > Wait Event tables dalam dokumentasi PostgreSQL.

### SQL Teratas

Saat peristiwa tunggu menunjukkan kemacetan, SQL teratas menunjukkan kueri mana yang paling berkontribusi pada pemuatan DB. Misalnya, saat ini mungkin ada banyak kueri yang berjalan di basis data, tetapi kueri tunggal mungkin menggunakan 99 persen dari muatan DB. Dalam hal ini, muatan tinggi mungkin menunjukkan masalah dalam kueri.

Secara default, konsol Wawasan Performa menampilkan kueri SQL teratas yang berkontribusi pada muatan basis data. Konsol juga menunjukkan statistik yang relevan untuk setiap pernyataan. Untuk mendiagnosis masalah performa untuk pernyataan tertentu, Anda dapat memeriksa rencana pelaksanaannya.

### Maksimum CPU

Di dasbor, bagan Basis data muatan mengumpulkan, menggabungkan, dan menampilkan informasi sesi. Untuk melihat apakah sesi aktif melebihi maksimumCPU, lihat hubungannya dengan CPU garis Max v. Performance Insights menentukan CPU nilai Max v dengan jumlah inti v CPU (virtualCPU) untuk instans DB Anda. Untuk Aurora Tanpa Server v2, Max v CPU mewakili perkiraan jumlah. vCPUs

Satu proses dapat berjalan pada v CPU pada satu waktu. Jika jumlah proses melebihi jumlahvCPUs, proses mulai mengantri. Saat antrian meningkat, kinerja database menurun. Jika beban DB sering di atas CPU garis Max v, dan status tunggu utama adalahCPU, kelebihan beban. CPU Dalam kasus ini, Anda mungkin ingin membatasi koneksi ke instance, menyetel SQL kueri apa pun dengan CPU beban tinggi, atau mempertimbangkan kelas instance yang lebih besar. Instans yang tinggi dan konsisten dari setiap status tunggu menunjukkan bahwa mungkin terjadi kemacetan atau masalah pertentangan sumber daya yang perlu diselesaikan. Ini bisa benar bahkan jika beban DB tidak melewati CPU garis Max v.

## Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk Wawasan Performa

**Important**

AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan

mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Tabel berikut berisi mesin DB Amazon Aurora yang mendukung Wawasan Performa.

| Mesin DB Amazon Aurora | Versi mesin dan Wilayah yang didukung | Pembatasan kelas instans |
|---|---|---|
| Edisi yang Kompatibel dengan Amazon Aurora MySQL | Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah Wawasan Performa dengan Aurora MySQL, lihat Wawasan Performa dengan Aurora MySQL. | Wawasan Performa memiliki batasan kelas mesin berikut:<br>• db.t2 – Tidak didukung<br>• db.t3 – Tidak didukung<br>• db.t4g.micro dan db.t4g.small – Tidak didukung |
| Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL | Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah Wawasan Performa dengan Aurora MySQL, lihat Wawasan Performa dengan Aurora PostgreSQL. | N/A |

Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Tabel berikut berisi mesin DB Amazon Aurora yang mendukung fitur Wawasan Performa.

| Fitur | Tingkat harga | Wilayah yang didukung | Mesin DB yang didukung | Kelas instans yang didukung |
|---|---|---|---|---|
| Statistik SQL untuk Wawasan Performa | Semua | Semua | Semua | Semua |
| Menganalisis performa basis data selama periode waktu tertentu | Khusus tingkat berbayar | Semua | Semua | Semua kecuali db.serverless (Aurora Serverless v2) |
| Melihat rekomendasi proaktif Performance Insights | Khusus tingkat berbayar | • AS Timur (Ohio)<br>• AS Timur (Virginia Utara)<br>• AS Barat (California Utara)<br>• AS Barat (Oregon)<br>• Asia Pasifik (Mumbai)<br>• Asia Pasifik (Seoul)<br>• Asia Pasifik (Singapura)<br>• Asia Pasifik (Sydney)<br>• Asia Pasifik (Tokyo) | Semua | Semua kecuali db.serverless (Aurora Serverless v2) |

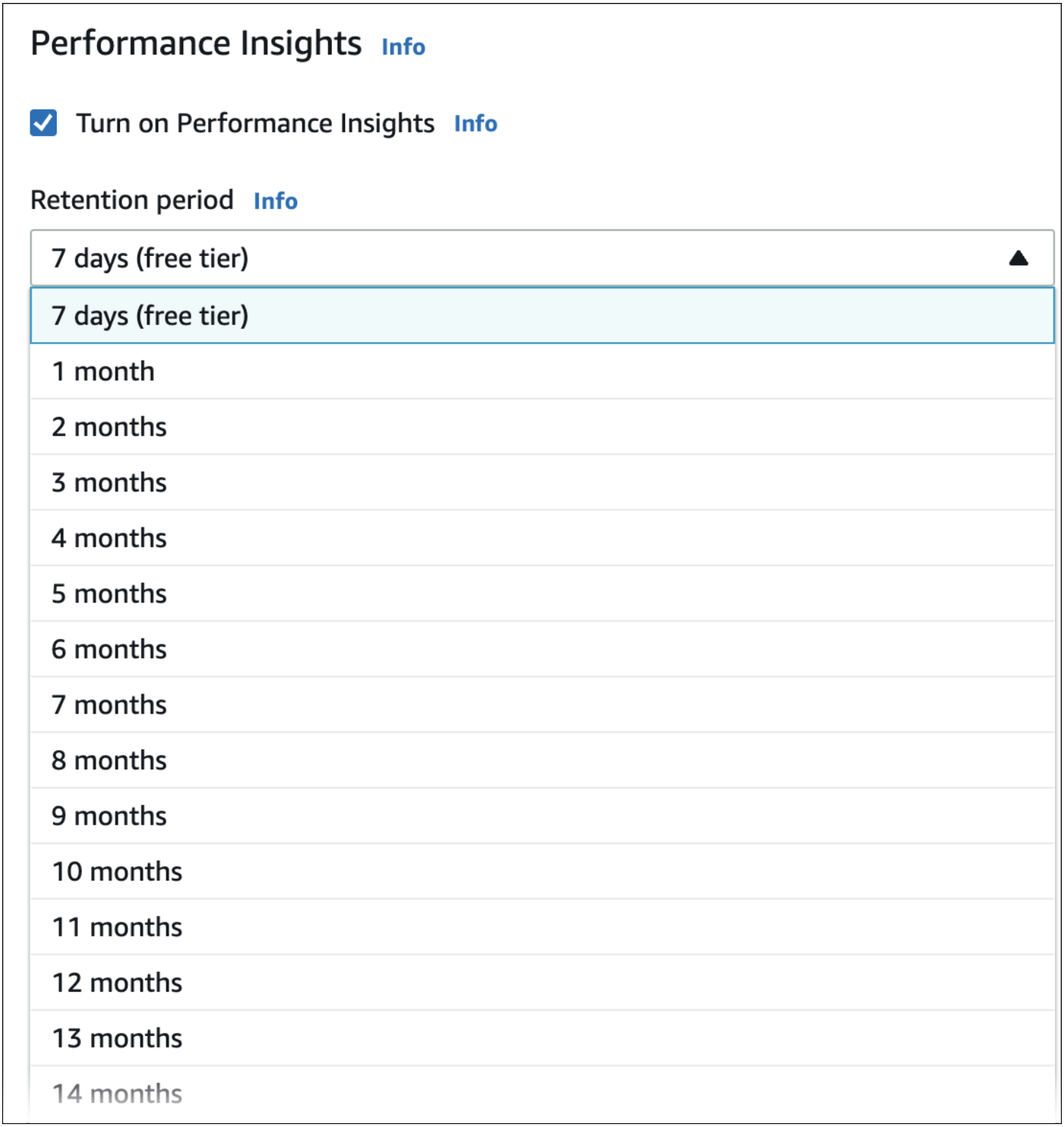
| Fitur | Tingkat harga | Wilayah yang didukung | Mesin DB yang didukung | Kelas instans yang didukung |
|---|---|---|---|---|
| | | • Kanada (Pusat)<br>• Eropa (Frankfurt)<br>• Eropa (Irlandia)<br>• Eropa (London)<br>• Eropa (Paris)<br>• Eropa (Stockholm)<br>• Amerika Selatan (Sao Paulo) | | |

## Harga dan retensi data untuk Wawasan Performa

Secara default, Wawasan Performa menawarkan tingkat gratis yang mencakup riwayat data performa selama 7 hari dan 1 juta permintaan API per bulan. Anda juga dapat membeli periode retensi yang lebih lama. Untuk informasi harga selengkapnya, lihat Harga Wawasan Performa.

Di konsol RDS, Anda dapat memilih salah satu periode retensi berikut untuk data Wawasan Performa:

- Default (7 hari)
- ***n***bulan, di ***n***mana angka dari 1-24




Untuk mempelajari cara mengatur periode retensi menggunakan AWS CLI, lihatMengaktifkan dan menonaktifkan Performance Insights untuk Aurora.

> **Note**
>
> Menghentikan dengan Performance Insights diaktifkan tidak memengaruhi retensi data. Sementara dihentikan, Performance Insights tidak akan mengumpulkan data apa pun.

## Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
> Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Anda dapat mengaktifkan Performance Insights untuk saat Anda membuatnya. Jika diperlukan, Anda dapat mematikannya nanti dengan memodifikasi Anda dari konsol. Mengaktifkan dan menonaktifkan Wawasan Performa tidak menyebabkan waktu henti, boot ulang, atau failover.

> **Note**
>
> Skema Performa adalah alat performa opsional yang digunakan oleh Aurora MySQL . Jika Anda mengaktifkan atau menonaktifkan Skema Performa, Anda perlu mem-boot ulang. Namun, jika Anda mengaktifkan atau menonaktifkan Wawasan Performa, Anda tidak perlu mem-boot ulang. Untuk informasi selengkapnya, lihat Ikhtisar Skema Kinerja untuk Performance Insights di Aurora.

Jika Anda menggunakan Performance Insights dengan database global Aurora, aktifkan Performance Insights satu per satu untuk database di masing-masing database. Wilayah AWS Untuk detailnya, lihat Memantau database global Amazon Aurora dengan Amazon Performance Insights RDS.

Agen Wawasan Performa menggunakan CPU dan memori terbatas di host DB. Ketika muatan DB tinggi, agen membatasi dampak performa dengan mengurangi frekuensi pengumpulan data.

Console

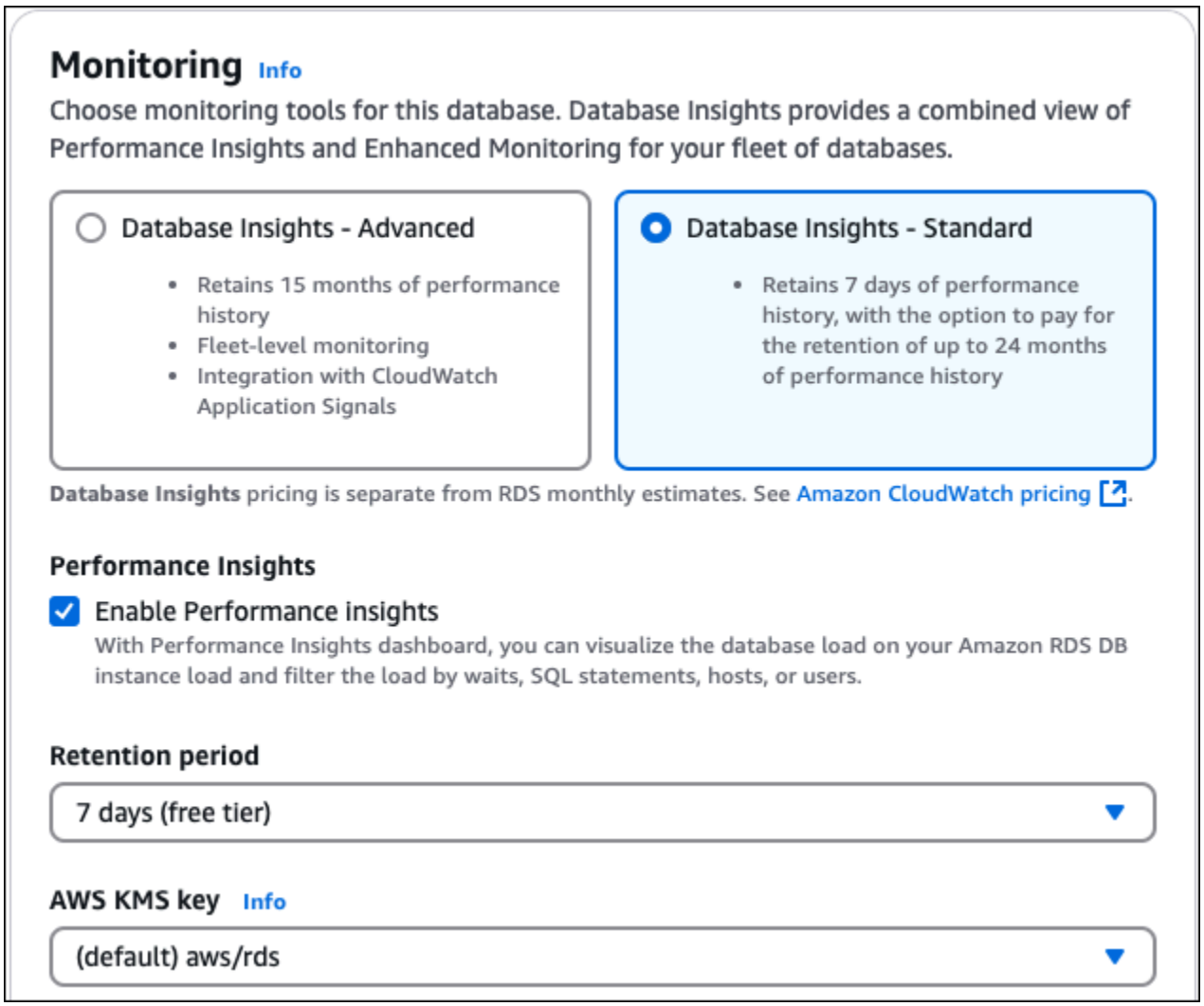
Di konsol, Anda dapat mengaktifkan atau menonaktifkan Wawasan Performa saat membuat klaster DB. Mengaktifkan Performance Insights memungkinkan Anda mengelola pengaturan dan opsi Performance Insights untuk klaster DB Anda. Pengaturan tingkat cluster berlaku untuk semua instans DB di cluster.

Mengaktifkan atau menonaktifkan Performance Insights saat membuat

Setelah membuat , Amazon RDS mengaktifkan Performance Insights secara default. Untuk menonaktifkan Performance Insights untuk cluster DB Anda, pilih opsi Database Insights — Standard dan batalkan pilihan opsi Aktifkan Performance Insights.

Untuk membuat cluster DB, ikuti petunjuk untuk mesin DB Anda diMembuat klaster DB Amazon Aurora.

Tangkapan layar berikut menunjukkan bagian Wawasan Performa.




Jika memilih Aktifkan Wawasan Performa, Anda akan memiliki opsi berikut:

- Retensi (hanya untuk mode Standar Database Insights) — Jumlah waktu untuk menyimpan data Performance Insights. Pengaturan retensi di tingkat gratis adalah Default (7 hari). Untuk mempertahankan data performa Anda lebih lama, tetapkan 1–24 bulan. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.
- AWS KMS key— Tentukan Anda AWS KMS key. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengubah AWS KMS kebijakan untuk Performance Insights.

Mengaktifkan atau menonaktifkan Performance Insights saat memodifikasi

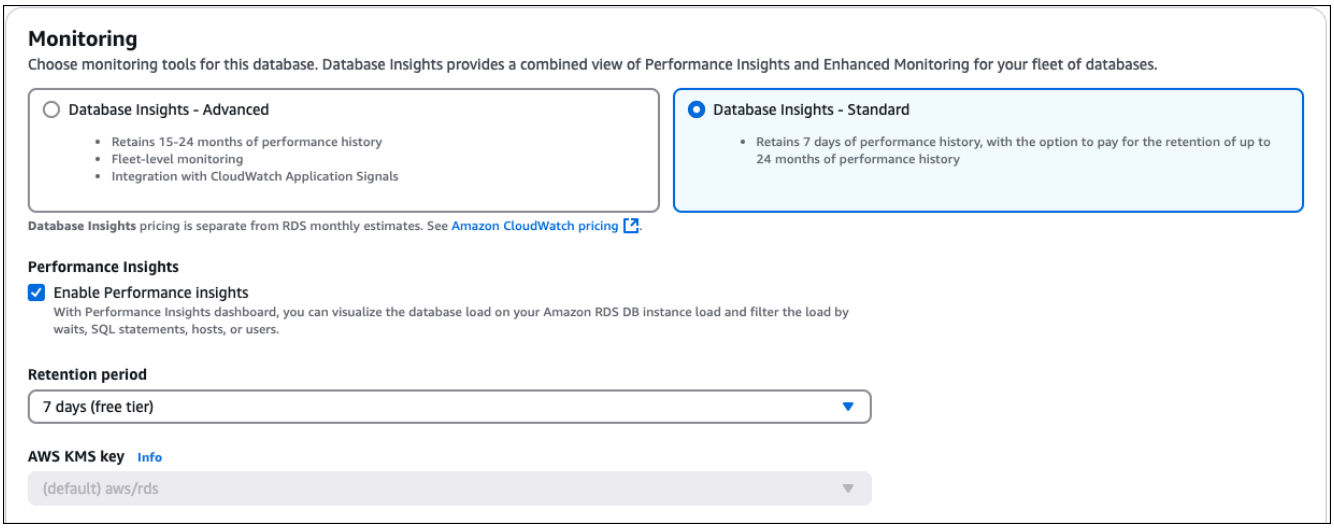
Di konsol, Anda dapat memodifikasi cluster DB untuk mengelola Performance Insights.

Untuk mengelola Performance Insights untuk menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih , dan pilih Modify.
4. Untuk mengaktifkan Performance Insights, pilih Aktifkan Performance Insights. Untuk menonaktifkan Performance Insights untuk cluster DB Anda, pilih opsi Database Insights — Standard dan batalkan pilihan opsi Aktifkan Performance Insights.

   Jika memilih Aktifkan Wawasan Performa, Anda akan memiliki opsi berikut:

   - Retensi (hanya untuk mode Standar Database Insights) — Jumlah waktu untuk menyimpan data Performance Insights. Pengaturan retensi di tingkat gratis adalah Default (7 hari). Untuk mempertahankan data performa Anda lebih lama, tetapkan 1–24 bulan. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.




5. Pilih Lanjutkan.

6. Untuk Penjadwalan Modifikasi, pilih Terapkan langsung. Jika Anda memilih Terapkan selama periode pemeliharaan terjadwal berikutnya, instans Anda akan mengabaikan pengaturan ini dan segera mengaktifkan Wawasan Performa.
7. Pilih Ubah instans.

AWS CLI

Saat Anda menggunakan create-db-instance AWS CLI perintah, aktifkan Performance Insights dengan menentukan `--enable-performance-insights` dan menyetel `--database-insights-mode` ke salah satu atau. `advanced` `standard` Untuk menonaktifkan Performance Insights, tentukan `--no-enable-performance-insights` dan setel `database-insights-mode` ke. `standard`

Anda juga dapat menentukan nilai-nilai ini menggunakan AWS CLI perintah berikut:

- create-db-cluster
- modify-db-cluster
- create-db-instance-read-replika
- modify-db-instance
- restore-db-instance-from-s3

Untuk mengelola Performance Insights untuk klaster DB menggunakan AWS CLI

- Panggil modify-db-cluster AWS CLI perintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama instans DB di klaster DB Anda.
  - `--enable-performance-insights` untuk mengaktifkan atau `--no-enable-performance-insights` untuk menonaktifkan
  - `database-insights-mode`— Mode Wawasan Database untuk . Untuk menonaktifkan Performance Insights, tetapkan nilai ini ke. `standard`

  Contoh berikut mengaktifkan Wawasan Performa untuk `sample-db-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
```

```
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights
```

Untuk Windows:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights
```

Saat mengaktifkan Wawasan Performa di CLI, Anda dapat secara opsional menentukan jumlah hari untuk mempertahankan data Wawasan Performa dengan opsi `--performance-insights-retention-period`. Anda dapat menentukan7, *`month`* * 31 (di *`month`* mana angka dari 1-23), atau. 731 Misalnya, jika Anda ingin mempertahankan data performa selama 3 bulan, tentukan 93, yakni 3 * 31. Nilai default-nya adalah 7 hari. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.

Contoh berikut mengaktifkan Wawasan Performa untuk `sample-db-cluster` dan menentukan bahwa data Wawasan Performa dipertahankan selama 93 hari (3 bulan).

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

Untuk Windows:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

Jika Anda menentukan periode retensi seperti 94 hari, yang bukan merupakan nilai yang valid, RDS akan mengeluarkan kesalahan.

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance
 operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93,
 124, 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

> **Note**
>
> Anda hanya dapat mengaktifkan Performance Insights untuk instance di klaster DB yang Performance Insights tidak dikelola di tingkat klaster.

RDS API

Saat Anda membuat instans DB baru di cluster DB menggunakan DBInstance operasi Create operation Amazon RDS API, aktifkan Performance Insights dengan `EnablePerformanceInsights` menyetelnya. `True` Untuk menonaktifkan Performance Insights, setel `EnablePerformanceInsights` ke `False` dan setel `DatabaseInsightsMode` ke. `standard`

Anda juga dapat menentukan nilai `EnablePerformanceInsights` menggunakan operasi API berikut:

- Buat DBCluster
- Memodifikasi DBCluster
- Memodifikasi DBInstance
- Buat DBInstance ReadReplica
- Kembalikan DBInstance dari3

Saat mengaktifkan Wawasan Performa, Anda dapat secara opsional menentukan jumlah waktu, dalam hari, untuk mempertahankan data Wawasan Performa dengan parameter `PerformanceInsightsRetentionPeriod`. Anda dapat menentukan7, *`month`* * 31 (di *`month`* mana angka dari 1-23), atau. `731` Misalnya, jika Anda ingin mempertahankan data performa selama 3 bulan, tentukan 93, yakni 3 * 31. Nilai default-nya adalah 7 hari. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.

# Ikhtisar Skema Kinerja untuk Performance Insights di Aurora

Skema Performa adalah fitur opsional untuk memantau performa runtime Aurora MySQL dengan tingkat detail rendah. Skema Performa dirancang untuk memiliki dampak minimal terhadap performa basis data. Wawasan Performa adalah fitur terpisah yang dapat digunakan dengan atau tanpa Skema Performa.

Topik


- Ikhtisar Skema Performa
- Wawasan Performa dan Skema Performa
- Manajemen Skema Performa otomatis berdasarkan Wawasan Performa
- Pengaruh boot ulang pada Skema Performa
- Menentukan apakah Wawasan Performa mengelola Skema Performa
- Aktifkan Skema Kinerja untuk Aurora


## Ikhtisar Skema Performa

Skema Performa memantau peristiwa dalam basis data Aurora MySQL. Peristiwa adalah tindakan server basis data yang memakan waktu dan telah diinstrumentasi sehingga informasi waktu dapat dikumpulkan. Contoh peristiwa antara lain:

- Panggilan fungsi
- Peristiwa tunggu untuk sistem operasi
- Tahapan eksekusi SQL
- Grup pernyataan SQL

Mesin penyimpanan `PERFORMANCE_SCHEMA` adalah mekanisme untuk mengimplementasikan fitur Skema Performa. Mesin ini mengumpulkan data peristiwa menggunakan instrumentasi dalam kode sumber basis data. Mesin menyimpan peristiwa dalam tabel hanya memori di basis data `performance_schema`. Anda dapat mengueri `performance_schema` sama seperti Anda mengueri tabel lainnya. Untuk informasi selengkapnya, lihat Skema Performa MySQL di Panduan Referensi MySQL.

## Wawasan Performa dan Skema Performa

Wawasan Performa dan Skema Performa adalah fitur terpisah, tetapi terhubung. Perilaku Wawasan Performa untuk Aurora MySQL bergantung pada apakah Skema Performa diaktifkan, dan jika demikian, apakah Wawasan Performa mengelola Skema Performa secara otomatis. Tabel berikut menjelaskan perilaku tersebut.

| Skema Performa diaktifkan | Mode manajemen Wawasan Performa | Perilaku Wawasan Performa |
|---|---|---|
| Ya | Otomatis | • Mengumpulkan informasi pemantauan tingkat rendah yang mendetail<br>• Mengumpulkan metrik sesi aktif setiap detik<br>• Menampilkan muatan DB yang dikategorikan berdasarkan peristiwa tunggu mendetail, yang dapat digunakan untuk mengidentifikasi kemacetan |
| Ya | Manual | • Mengumpulkan peristiwa tunggu dan metrik per SQL<br>• Mengumpulkan metrik sesi aktif setiap detik<br>• Melaporkan status pengguna seperti memasukkan dan mengirim, yang tidak membantu Anda mengidentifikasi kemacetan |
| Tidak | N/A | • Tidak mengumpulkan peristiwa tunggu, metrik per SQL, atau informasi pemantauan tingkat rendah mendetail lainnya<br>• Mengumpulkan metrik sesi aktif setiap lima detik, bukan setiap detik<br>• |

| Skema Performa diaktifkan | Mode manajemen Wawasan Performa | Perilaku Wawasan Performa |
|---|---|---|
| | | Melaporkan status pengguna seperti memasukkan dan mengirim, yang tidak membantu Anda mengidentifikasi kemacetan |

## Manajemen Skema Performa otomatis berdasarkan Wawasan Performa

Ketika Anda membuat instans DB Aurora MySQL dengan Wawasan Performa diaktifkan, Skema Performa juga diaktifkan. Dalam kasus ini, Wawasan Performa secara otomatis mengelola parameter Skema Performa Anda. Ini adalah konfigurasi yang disarankan.

Saat Performance Insights mengelola Skema Kinerja secara otomatis, Sumber is. `performance_schema` `System default`

> **Note**
>
> Manajemen Skema Performa otomatis tidak didukung untuk kelas instans t4g.medium.

Anda juga dapat mengelola Skema Performa secara manual. Jika Anda memilih opsi ini, atur parameter sesuai dengan nilai dalam tabel berikut.

| Nama parameter | Nilai parameter |
|---|---|
| `performance_schema` | 1 (Kolom Sumber memiliki nilai `Modified`) |
| `performance-schema-consumer-events-waits-current` | `ON` |
| `performance-schema-instrument` | `wait/%=ON` |
| `performance_schema_consumer_global_instrumentation` | `1` |

| Nama parameter | Nilai parameter |
| --- | --- |
| `performance_schema_consumer_thread_instrumentation` | 1 |

Jika Anda mengubah nilai parameter `performance_schema` secara manual, dan kemudian ingin mengubah ke manajemen otomatis di lain waktu, lihat Aktifkan Skema Kinerja untuk Aurora.

> **Important**
>
> Saat Wawasan Performa mengaktifkan Skema Performa, nilai grup parameter tidak akan diubah. Namun, nilainya diubah pada instans DB yang sedang berjalan. Satu-satunya cara untuk melihat nilai yang diubah adalah dengan menjalankan perintah `SHOW GLOBAL VARIABLES`.

## Pengaruh boot ulang pada Skema Performa

Wawasan Performa dan Skema Performa berbeda dalam persyaratannya untuk boot ulang instans DB:

Skema Performa

Untuk mengaktifkan atau menonaktifkan fitur ini, Anda harus mem-boot ulang instans DB.

Wawasan Performa

Untuk mengaktifkan atau menonaktifkan fitur ini, Anda tidak harus mem-boot ulang instans DB.

Jika Skema Performa saat ini tidak diaktifkan, dan Anda mengaktifkan Wawasan Performa tanpa mem-boot ulang instans DB, Skema Performa tidak akan diaktifkan.

## Menentukan apakah Wawasan Performa mengelola Skema Performa

Untuk mengetahui apakah Performance Insights saat ini mengelola Skema Kinerja untuk semua versi engine utama yang didukung, tinjau tabel berikut.

| Pengaturan parameter performance_schema | Pengaturan kolom Source | Wawasan Performa mengelola Skema Performa? |
|---|---|---|
| `0` | `System default` | Ya |
| `0` atau `1` | `Modified` | Tidak |

Dalam prosedur berikut, Anda menentukan apakah Performance Insights mengelola Skema Kinerja secara otomatis.

Untuk menentukan apakah Wawasan Performa mengelola Skema Performa secara otomatis

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Grup parameter.
3. Pilih nama grup parameter untuk instans DB Anda.
4. Masukkan **`performance_schema`** ke bilah pencarian.
5. Periksa apakah Sumber adalah default sistem dan Nilai adalah 0. Jika demikian, Wawasan Performa mengelola Skema Performa secara otomatis.

   Dalam contoh yang ditampilkan di sini, Performance Insights tidak mengelola Skema Kinerja secara otomatis.

## Aktifkan Skema Kinerja untuk Aurora

Asumsikan bahwa Wawasan Performa diaktifkan untuk instans DB, tetapi saat ini tidak mengelola Wawasan Performa. Jika Anda ingin mengizinkan Wawasan Performa mengelola Skema Performa secara otomatis, selesaikan langkah-langkah berikut.

Untuk mengonfigurasi Skema Performa untuk manajemen otomatis

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Grup parameter.
3. Pilih nama grup parameter untuk instans DB.
4. Pilih Edit.
5. Masukkan **`performance_schema`** ke bilah pencarian.
6. Pilih parameter `performance_schema`.
7. Pilih Setel ke nilai default.
8. Konfirmasikan dengan memilih Tetapkan nilai ke default.
9. Pilih Simpan Perubahan.
10. Boot ulang instans DB.

> **Important**
>
> Setiap kali Anda mengaktifkan atau menonaktifkan Skema Performa, pastikan untuk mem-boot ulang instans DB.

Untuk informasi tentang cara mengubah parameter instans, lihat Memodifikasi parameter dalam grup parameter DB di . Untuk informasi selengkapnya tentang dasbor, lihat Menganalisis metrik dengan dasbor Wawasan Performa. Untuk informasi selengkapnya tentang skema kinerja MySQL, lihat MySQL Performance Schema (untuk 8.0) dan MySQL Performance Schema (untuk 8.4) dalam dokumentasi MySQL.

## Mengonfigurasi kebijakan akses untuk Wawasan Performa

Untuk mengakses Performance Insights, prinsipal harus memiliki izin yang sesuai dari AWS Identity and Access Management (IAM). Anda dapat memberikan akses dengan cara berikut:

- Melampirkan kebijakan `AmazonRDSPerformanceInsightsReadOnly` terkelola ke kumpulan izin atau peran untuk mengakses semua operasi hanya-baca dari API Wawasan Performa. Lampirkan CloudWatch izin berikut:, `GetMetricStatisticsListMetrics`, dan. `GetMetricData` Untuk informasi selengkapnya tentang CloudWatch izin, lihat referensi CloudWatch izin Amazon.

- Melampirkan kebijakan `AmazonRDSPerformanceInsightsFullAccess` terkelola ke kumpulan izin atau peran untuk mengakses semua operasi API Wawasan Performa. Lampirkan CloudWatch izin berikut:, `GetMetricStatistics`, `ListMetrics`, dan. `GetMetricData` Untuk informasi selengkapnya tentang CloudWatch izin, lihat referensi CloudWatch izin Amazon.
- Membuat kebijakan IAM kustom dan melampirkannya ke kumpulan izin atau peran.

Jika Anda menetapkan kunci terkelola pelanggan saat mengaktifkan Performance Insights, pastikan bahwa pengguna di akun Anda memiliki `kms:Decrypt` dan `kms:GenerateDataKey` izin di akun. AWS KMS key

Di bagian berikut, lampirkan kebijakan AWS terkelola ke prinsipal IAM, buat kebijakan IAM khusus, ubah AWS KMS kebijakan, dan berikan akses halus untuk Performance Insights.

Topik

- Melampirkan AmazonRDSPerformanceInsightsReadOnly kebijakan untuk kepala sekolah IAM
- Melampirkan AmazonRDSPerformanceInsightsFullAccess kebijakan untuk kepala sekolah IAM
- Membuat IAM kebijakan khusus untuk Performance Insights
- Mengubah AWS KMS kebijakan untuk Performance Insights
- Memberikan akses halus untuk Performance Insights

## Melampirkan AmazonRDSPerformanceInsightsReadOnly kebijakan untuk kepala sekolah IAM

`AmazonRDSPerformanceInsightsReadOnly`adalah kebijakan AWS terkelola yang memberikan akses ke semua operasi hanya-baca dari Amazon RDS Performance Insights API.

Jika Anda melampirkan `AmazonRDSPerformanceInsightsReadOnly` ke kumpulan izin atau peran, penerima dapat menggunakan Wawasan Performa beserta fitur konsol lainnya.

Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: A mazonRDSPerformance InsightsReadOnly.

## Melampirkan AmazonRDSPerformanceInsightsFullAccess kebijakan untuk kepala sekolah IAM

`AmazonRDSPerformanceInsightsFullAccess`adalah kebijakan AWS terkelola yang memberikan akses ke semua operasi Amazon RDS Performance Insights API.

Jika Anda melampirkan `AmazonRDSPerformanceInsightsFullAccess` ke kumpulan izin atau peran, penerima dapat menggunakan Wawasan Performa beserta fitur konsol lainnya.

Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: A mazonRDSPerformance InsightsFullAccess.

## Membuat IAM kebijakan khusus untuk Performance Insights

Untuk pengguna yang tidak memiliki `AmazonRDSPerformanceInsightsFullAccess` kebijakan `AmazonRDSPerformanceInsightsReadOnly` atau kebijakan, Anda dapat memberikan akses ke Performance Insights dengan membuat atau memodifikasi kebijakan yang dikelola pengguna. IAM Jika Anda melampirkan kebijakan ke set IAM izin atau peran, penerima dapat menggunakan Performance Insights.

Untuk membuat kebijakan kustom

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Pada halaman Buat Kebijakan, pilih JSONopsi.
5. Salin dan tempel teks yang disediakan di bagian dokumen JSON kebijakan di Panduan Referensi Kebijakan AWS Terkelola untuk AmazonRDSPerformanceInsightsReadOnly atau AmazonRDSPerformanceInsightsFullAccesskebijakan.
6. Pilih Tinjau kebijakan.
7. Berikan nama untuk kebijakan tersebut dan secara opsional deskripsi, lalu pilih Buat kebijakan.

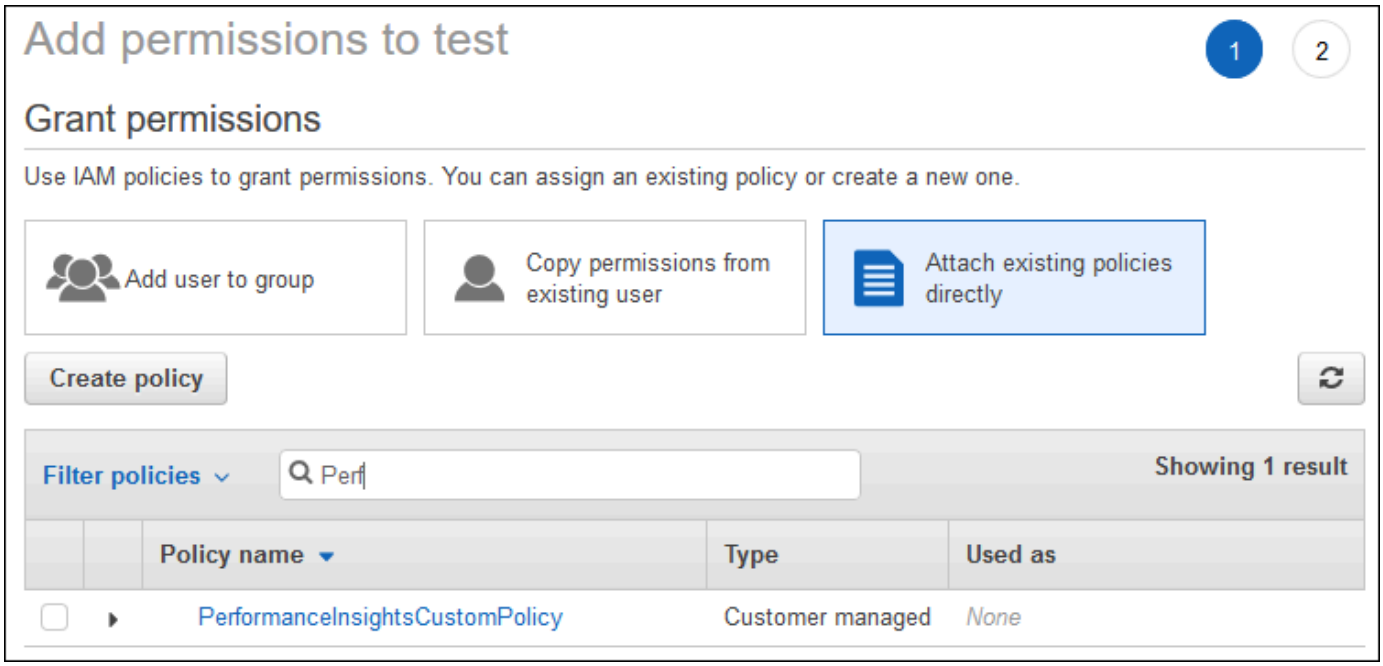
Sekarang, Anda dapat menyisipkan kebijakan ke kumpulan izin atau peran. Prosedur berikut mengasumsikan bahwa Anda sudah memiliki pengguna yang tersedia untuk tujuan ini.

Untuk melampirkan kebijakan ini ke pengguna

1. Membuka IAM konsol di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Pengguna.
3. Pilih pengguna yang ada dari daftar.

> **Important**
>
> Untuk menggunakan Performance Insights, pastikan Anda memiliki akses ke Amazon RDS selain kebijakan kustom. Misalnya, kebijakan yang `AmazonRDSPerformanceInsightsReadOnly` telah ditentukan sebelumnya menyediakan akses hanya-baca ke Amazon. RDS Untuk informasi selengkapnya, lihat Mengelola akses menggunakan kebijakan.

4. Di halaman Ringkasan, pilih Tambahkan izin.
5. Pilih Lampirkan kebijakan yang sudah ada secara langsung. Untuk Penelusuran, ketik beberapa karakter pertama dari nama kebijakan Anda, seperti yang ditunjukkan pada gambar berikut.




6. Pilih kebijakan Anda, lalu pilih Berikutnya: Tinjauan.
7. Pilih Tambahkan izin.

## Mengubah AWS KMS kebijakan untuk Performance Insights

Performance Insights menggunakan perangkat AWS KMS key untuk mengenkripsi data sensitif. Saat mengaktifkan Performance Insights melalui API atau konsol, Anda dapat melakukan salah satu hal berikut:

- Pilih default Kunci yang dikelola AWS.

Amazon RDS menggunakan instans DB baru Anda. Kunci yang dikelola AWS Amazon RDS menciptakan Kunci yang dikelola AWS untuk Anda Akun AWS. Anda Akun AWS memiliki perbedaan Kunci yang dikelola AWS untuk Amazon RDS untuk masing-masing Wilayah AWS.

- Memilih kunci yang dikelola pelanggan.

  Jika Anda menentukan kunci yang dikelola pelanggan, pengguna di akun Anda yang memanggil Performance Insights API memerlukan izin `kms:Decrypt` dan `kms:GenerateDataKey` izin pada kunci tersebut. KMS Anda dapat mengonfigurasi izin ini melalui IAM kebijakan. Namun, kami menyarankan Anda mengelola izin ini melalui kebijakan KMS utama Anda. Untuk informasi selengkapnya, lihat Kebijakan kunci di AWS KMS di Panduan Developer AWS Key Management Service .

Example

Contoh berikut menunjukkan cara menambahkan pernyataan ke kebijakan KMS kunci Anda. Pernyataan ini mengizinkan akses ke Wawasan Performa. Bergantung pada bagaimana Anda menggunakan KMS kunci, Anda mungkin ingin mengubah beberapa batasan. Sebelum menambahkan pernyataan ke kebijakan, hapus semua komentar.

```
{
"Version" : "2012-10-17",
 "Id" : "your-policy",
 "Statement" : [ {
    //This represents a statement that currently exists in your policy.
 }
 ....,
 //Starting here, add new statement to your policy for Performance Insights.
 //We recommend that you add one new statement for every RDS instance
{
    "Sid" : "Allow viewing RDS Performance Insights",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            //One or more principals allowed to access Performance Insights
            "arn:aws:iam::444455556666:role/Role1"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
```

```
    ],
    "Resource": "*",
    "Condition" : {
        "StringEquals" : {
            //Restrict access to only RDS APIs (including Performance Insights).
            //Replace region with your AWS Region.
            //For example, specify us-west-2.
            "kms:ViaService" : "rds.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
            //Restrict access to only data encrypted by Performance Insights.
            "kms:EncryptionContext:aws:pi:service": "rds",
            "kms:EncryptionContext:service": "pi",

            //Restrict access to a specific RDS instance.
            //The value is a DbiResourceId.
           "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBBCCCCDDDDDEEEEE"
        }
    }
}
```

Bagaimana Performance Insights menggunakan kunci yang dikelola AWS KMS pelanggan

Wawasan Performa menggunakan kunci yang dikelola pelanggan untuk mengenkripsi data sensitif. Saat mengaktifkan Performance Insights, Anda dapat memberikan AWS KMS kunci melalui fitur. API Performance Insights membuat KMS izin pada kunci ini. Ini menggunakan kunci dan melakukan operasi yang diperlukan untuk memproses data sensitif. Data sensitif mencakup bidang-bidang seperti pengguna, database, aplikasi, dan teks SQL kueri. Wawasan Performa memastikan bahwa data tetap terenkripsi baik saat tidak aktif maupun saat transit.

Cara kerja Performance Insights IAM AWS KMS

IAMmemberikan izin untuk spesifikAPIs. Performance Insights memiliki publik berikutAPIs, yang dapat Anda batasi menggunakan kebijakan: IAM

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`

- `ListAvailableResourceMetrics`

Anda dapat menggunakan API permintaan berikut untuk mendapatkan data sensitif.

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

Saat Anda menggunakan API untuk mendapatkan data sensitif, Performance Insights memanfaatkan kredensi pemanggil. Pemeriksaan ini memastikan bahwa akses ke data sensitif terbatas pada mereka yang memiliki akses ke KMS kunci.

Saat memanggil iniAPIs, Anda memerlukan izin untuk memanggil API melalui IAM kebijakan dan izin untuk menjalankan `kms:decrypt` tindakan melalui kebijakan utama AWS KMS .

`GetResourceMetrics`APIDapat mengembalikan data sensitif dan tidak sensitif. Parameter permintaan menentukan apakah respons harus menyertakan data sensitif. APIMengembalikan data sensitif ketika permintaan menyertakan dimensi sensitif baik dalam parameter filter atau grup-menurut.

Untuk informasi lebih lanjut tentang dimensi yang dapat Anda gunakan dengan `GetResourceMetrics`API, lihat DimensionGroup.

Example Contoh

Contoh berikut meminta data sensitif untuk grup `db.user`:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
```

```
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

Contoh berikut meminta data non-sensitif untuk metrik `db.load.avg`:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

## Memberikan akses halus untuk Performance Insights

Kontrol akses berbutir halus menawarkan cara tambahan untuk mengontrol akses ke Performance Insights. Kontrol akses ini dapat mengizinkan atau menolak akses ke dimensi individual untuk `GetResourceMetricsDescribeDimensionKeys`, dan tindakan `GetDimensionKeyDetails` Performance Insights. Untuk menggunakan akses berbutir halus, tentukan dimensi dalam kebijakan IAM dengan menggunakan tombol kondisi. Evaluasi akses mengikuti logika evaluasi kebijakan IAM. Untuk informasi selengkapnya, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM. Jika pernyataan kebijakan IAM tidak menentukan dimensi apa pun, maka pernyataan tersebut mengontrol akses ke semua dimensi untuk tindakan yang ditentukan. Untuk daftar dimensi yang tersedia, lihat DimensionGroup.

Untuk mengetahui dimensi yang diizinkan untuk diakses oleh kredensi Anda, gunakan `AuthorizedActions` parameter `ListAvailableResourceDimensions` dan tentukan tindakannya. Nilai yang diizinkan untuk `AuthorizedActions` adalah sebagai berikut:

- `GetResourceMetrics`
- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`

Misalnya, jika Anda `GetResourceMetrics` menentukan `AuthorizedActions` parameter, `ListAvailableResourceDimensions` mengembalikan daftar dimensi yang diizinkan untuk diakses oleh `GetResourceMetrics` tindakan. Jika Anda menentukan beberapa tindakan dalam `AuthorizedActions` parameter, kemudian `ListAvailableResourceDimensions` mengembalikan persimpangan dimensi bahwa tindakan tersebut diizinkan untuk mengakses.

Example

Contoh berikut menyediakan akses ke dimensi yang ditentukan untuk `GetResourceMetrics` dan `DescribeDimensionKeys` tindakan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
```

```
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    // only these dimensions are allowed. Dimensions not included in
                    // a policy with "Allow" effect will be denied
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
}
```

Berikut ini adalah respons untuk dimensi yang diminta:

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
```

```
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

Contoh berikut menentukan satu izinkan dan dua tolak akses untuk dimensi.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
          },
```

```
          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },

        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
```

```
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

Berikut ini adalah tanggapan untuk dimensi yang diminta:

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },
```

```
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // allowed for DescribeDimensionKeys because our IAM Policy
                  // denies it only for GetResourceMetrics
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
```

```
        ] }
    ]
}
```

## Menganalisis metrik dengan dasbor Wawasan Performa

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalaman konsol Performance Insights, periode retensi fleksibel (1-24 bulan), dan harga terkait. Performance Insights API akan terus ada tanpa perubahan harga. Biaya untuk API Performance Insights akan muncul di AWS tagihan Anda dengan biaya CloudWatch Database Insights. Sebaiknya Anda meningkatkan DB cluster DB apa pun menggunakan tingkat berbayar Performance Insights ke mode Advanced Database Insights sebelum 30 November 2025. Untuk informasi tentang memutakhirkan ke mode Advanced Database Insights, lihat. Mengaktifkan mode Lanjut Wawasan Database untuk
> Jika Anda tidak mengambil tindakan, DB mengelompokkan yang menggunakan Performance Insights akan secara default menggunakan mode Standar Database Insights. Dengan mode Standar Database Insights, Anda mungkin kehilangan akses ke riwayat data kinerja setelah 7 hari dan mungkin tidak dapat menggunakan paket eksekusi dan fitur analisis sesuai permintaan di konsol Amazon RDS. Setelah 30 November 2025, hanya mode Advanced Database Insights yang akan mendukung rencana eksekusi dan analisis sesuai permintaan. Dengan CloudWatch Database Insights, Anda dapat memantau beban database untuk armada database Anda dan menganalisis serta memecahkan masalah kinerja dalam skala besar. Untuk informasi selengkapnya tentang Wawasan Database, lihatMemantau database Amazon Aurora dengan Database CloudWatch Insights. Untuk informasi harga, lihat CloudWatch Harga Amazon.

Dasbor Wawasan Performa berisi informasi performa basis data untuk membantu Anda menganalisis dan memecahkan masalah performa. Di halaman dasbor utama, Anda dapat melihat informasi tentang muatan basis data. Anda dapat “mengiris” muatan DB berdasarkan dimensi seperti peristiwa tunggu atau SQL.

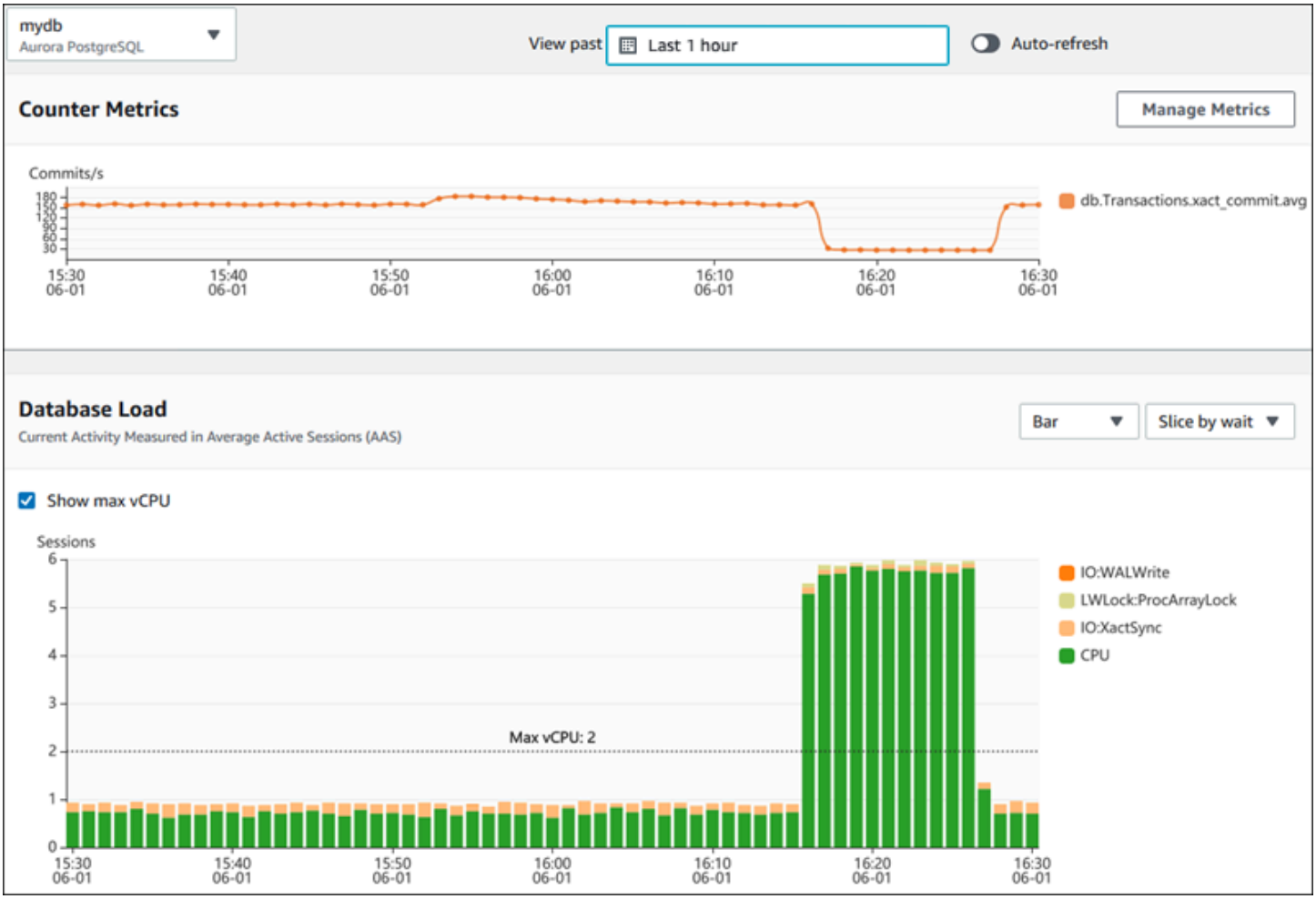
Dasbor Wawasan Performa

- Ikhtisar dasbor Wawasan Performa

- Mengakses dasbor Wawasan Performa
- Menganalisis muatan DB menurut peristiwa tunggu
- Menganalisis performa basis data selama periode waktu tertentu
- Menganalisis kueri dengan SQL tab Teratas di Performance Insights

## Ikhtisar dasbor Wawasan Performa

Dasbor adalah cara termudah untuk berinteraksi dengan Wawasan Performa. Contoh berikut menunjukkan dasbor untuk instance Postgre SQL DB.




Topik

- Filter rentang waktu
- Bagan metrik penghitung
- Bagan muatan basis data

- Tabel Dimensi teratas

Filter rentang waktu

Secara default, dasbor Wawasan Performa menampilkan muatan DB selama satu jam terakhir. Anda dapat menyesuaikan rentang ini menjadi sesingkat 5 menit atau selama 2 tahun. Anda juga dapat memilih rentang relatif kustom.

Anda dapat memilih rentang absolut dengan tanggal dan waktu awal dan akhir. Contoh berikut menunjukkan rentang waktu yang dimulai pada tengah malam pada 9/25/24 dan berakhir pada 11:59 PM pada 9/28/24.

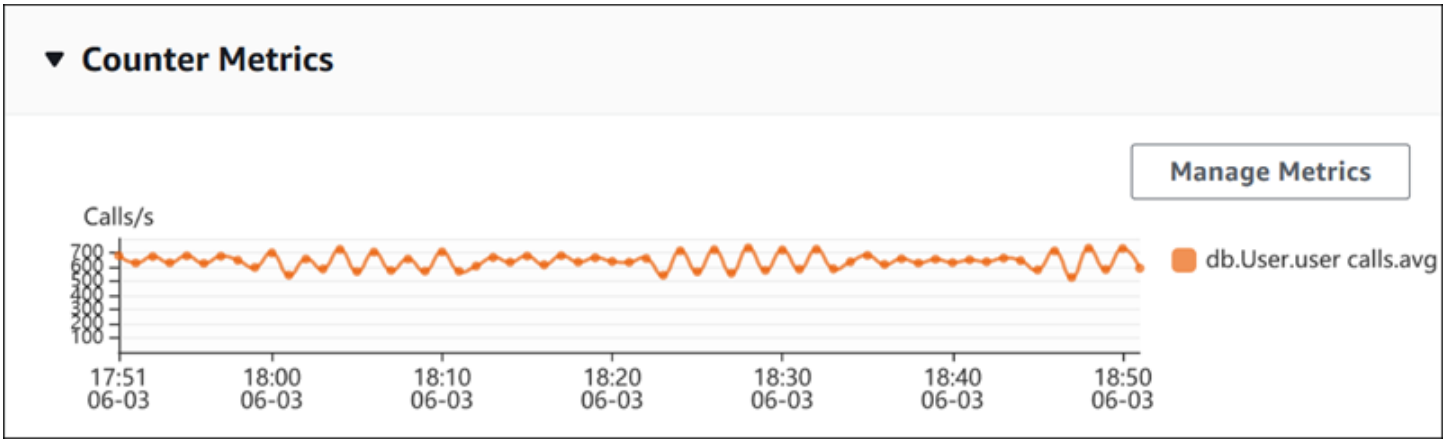
Secara default, zona waktu untuk dasbor Performance Insights adalah Coordinated Universal Time (). UTC Anda juga dapat memilih zona waktu setempat.

Bagan metrik penghitung

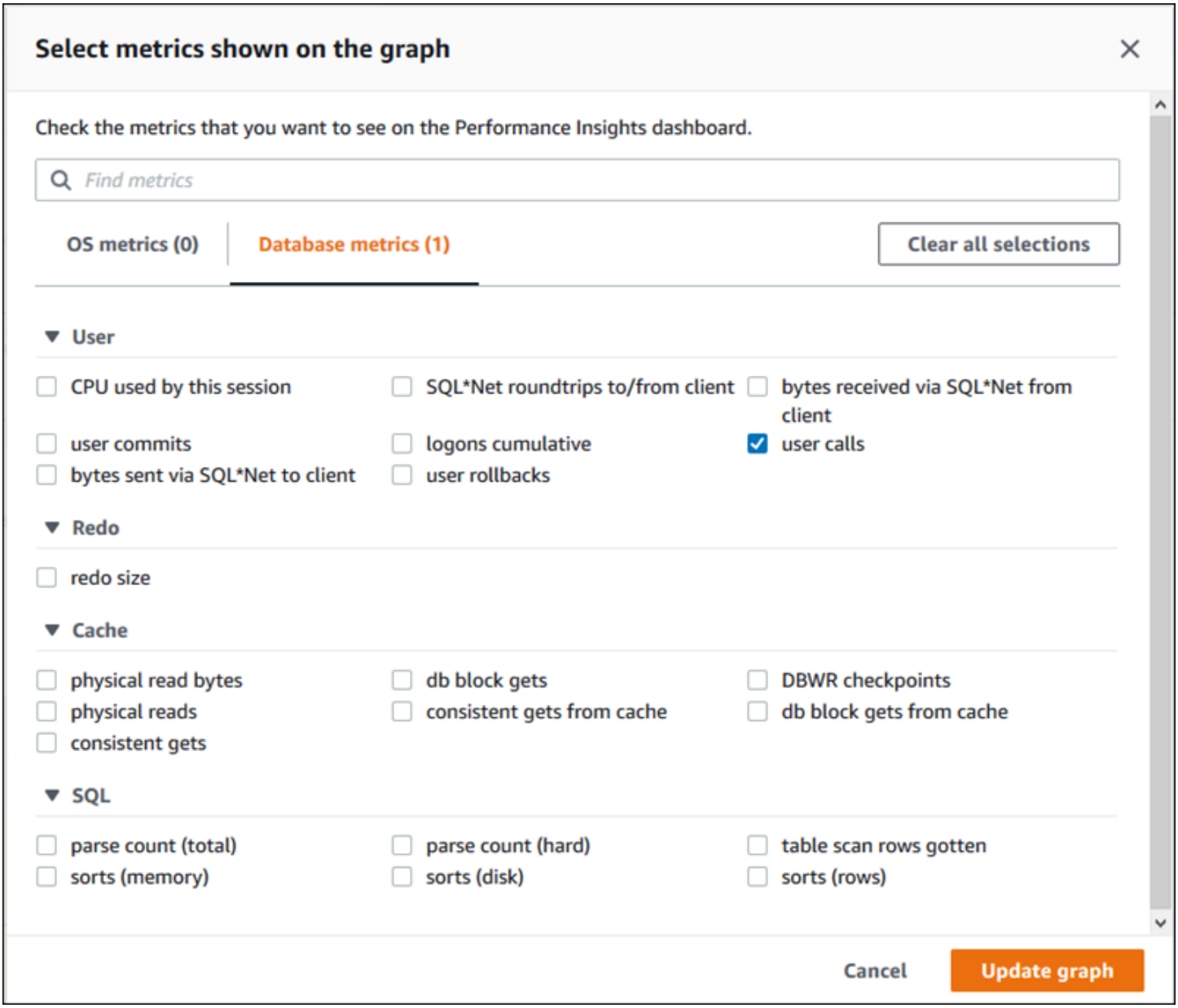
Dengan metrik penghitung, Anda dapat menyesuaikan dasbor Wawasan Performa untuk menyertakan hingga 10 grafik tambahan. Grafik ini menunjukkan pilihan dari sejumlah sistem operasi dan metrik performa basis data. Anda dapat menghubungkan informasi ini dengan muatan DB untuk membantu mengidentifikasi dan menganalisis masalah performa.

Bagan Metrik Penghitung menampilkan data untuk penghitung performa. Metrik default bergantung pada mesin DB:

- Aurora Saya - SQL `db.SQL.Innodb_rows_read.avg`
- Aurora Postgre — SQL `db.Transactions.xact_commit.avg`



Untuk mengubah penghitung performa, pilih Kelola Metrik. Anda dapat memilih beberapa Metrik OS atau Metrik basis data, seperti yang ditunjukkan di tangkapan layar berikut. Untuk melihat detail setiap metrik, arahkan kursor ke nama metrik.

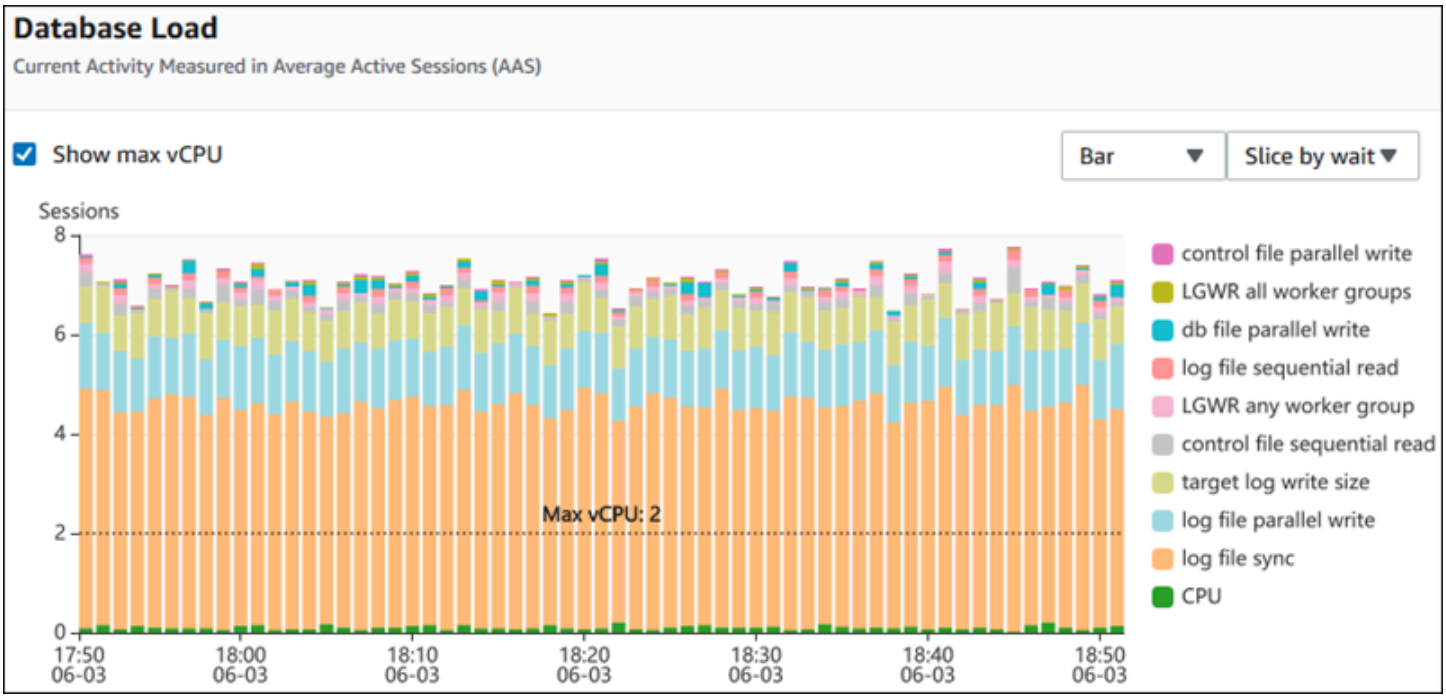



Untuk deskripsi metrik penghitung yang dapat ditambahkan untuk setiap mesin DB, lihat Metrik penghitung Wawasan Performa.

Bagan muatan basis data

Bagan beban Database menunjukkan bagaimana aktivitas database dibandingkan dengan kapasitas instans DB seperti yang diwakili oleh CPU baris Max v. Secara default, bagan garis bertumpuk

mewakili muatan DB sebagai sesi aktif rata-rata per unit waktu. Muatan DB diiris (dikelompokkan) berdasarkan status tunggu.

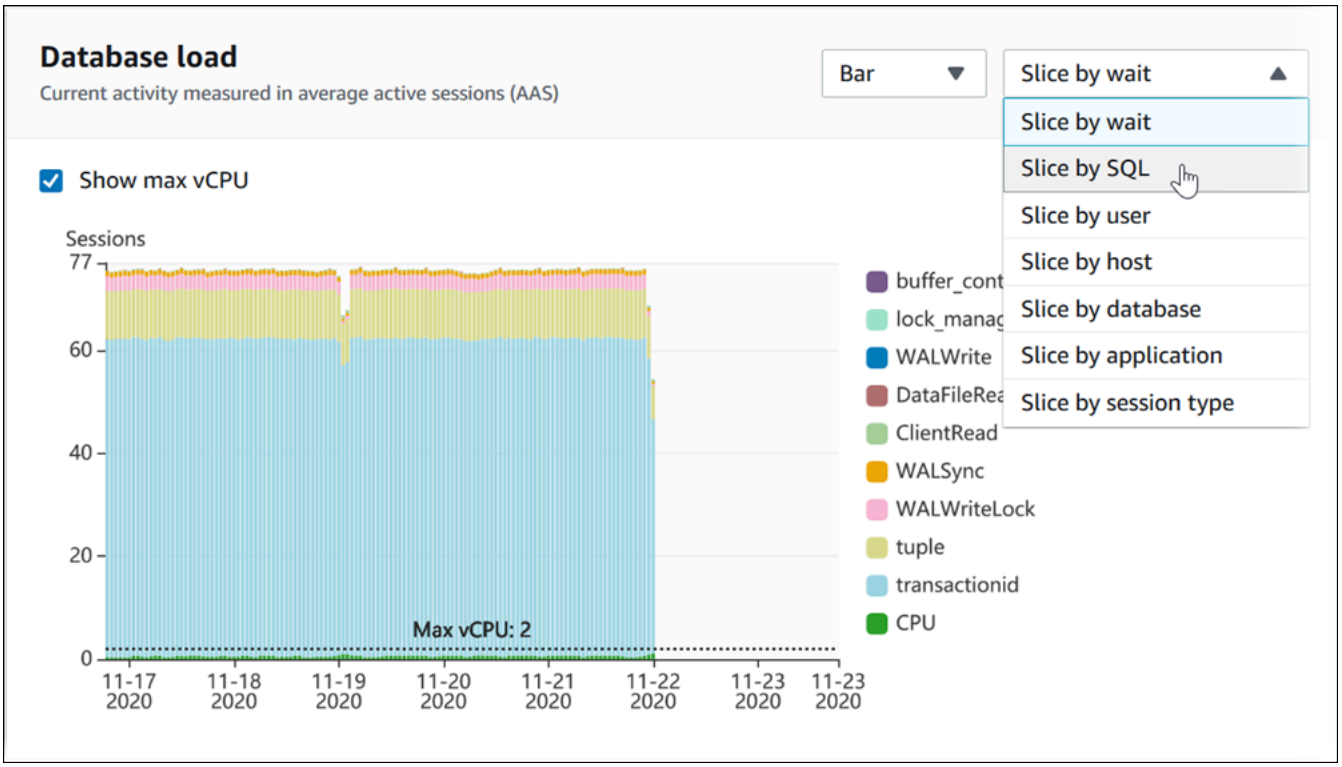



Muatan DB diiris berdasarkan dimensi

Anda dapat memilih untuk menampilkan muatan sebagai sesi aktif yang dikelompokkan berdasarkan dimensi yang didukung. Tabel berikut menunjukkan dimensi yang didukung untuk mesin yang berbeda.

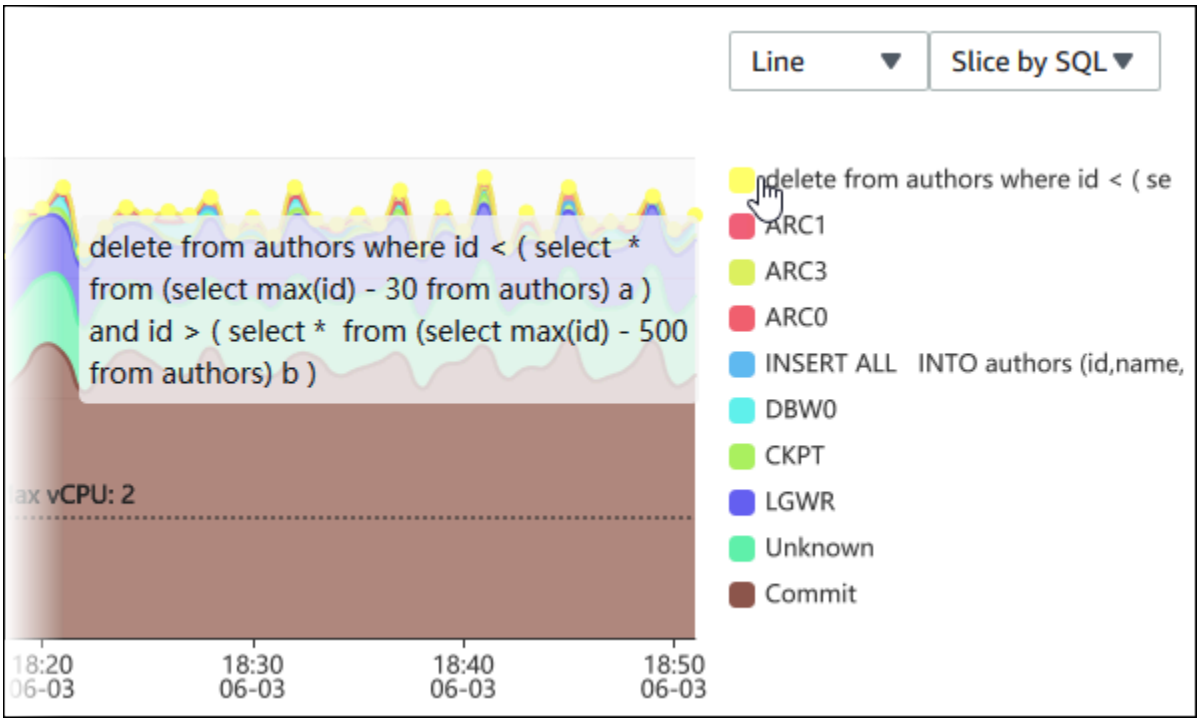
| Dimensi | Aurora Postgre SQL | Aurora Saya SQL |
|---|---|---|
| Host | Ya | Ya |
| SQL | Ya | Ya |
| Pengguna | Ya | Ya |
| Tunggu | Ya | Ya |
| Aplikasi | Ya | Tidak |
| Basis data | Ya | Ya |
| Jenis sesi | Ya | Tidak |

Gambar berikut menunjukkan dimensi untuk instance Postgre SQL DB.

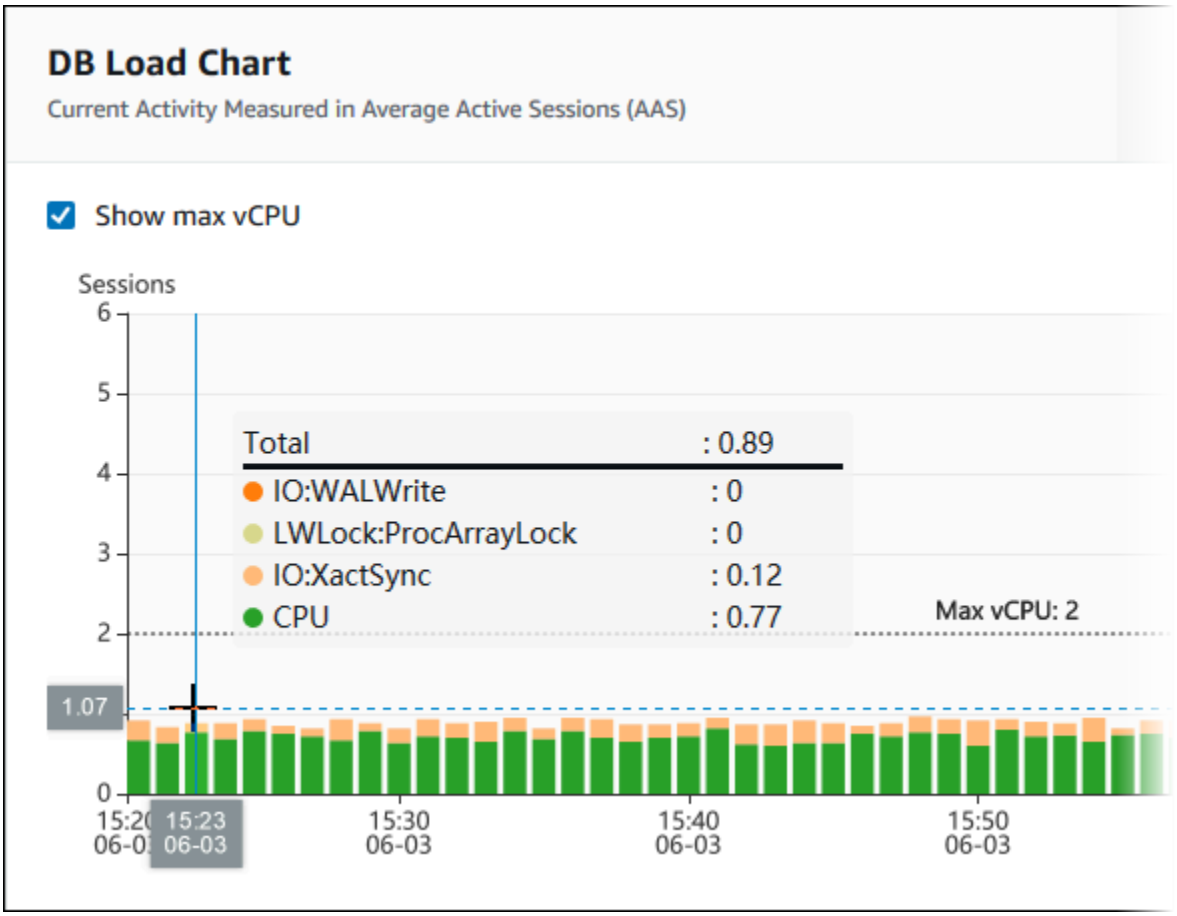



Detail muatan DB untuk item dimensi

Untuk melihat detail tentang item muatan DB dalam dimensi, arahkan kursor ke nama item. Gambar berikut menunjukkan detail untuk SQL pernyataan.

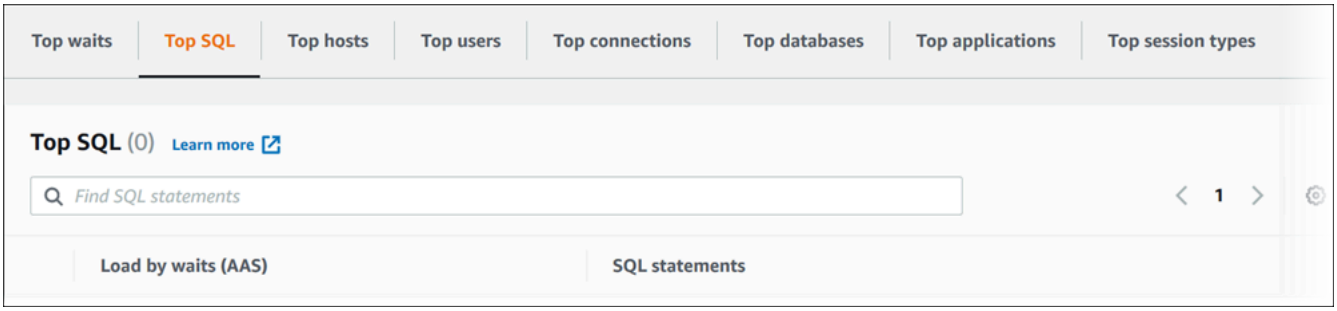


Untuk melihat detail setiap item selama periode waktu yang dipilih dalam legenda, arahkan kursor ke item tersebut.




Tabel Dimensi teratas

Tabel Dimensi teratas mengiris muatan DB dengan dimensi yang berbeda. Dimensi adalah kategori atau “irisan menurut” untuk karakteristik muatan DB yang berbeda. Jika dimensinyaSQL, Top SQL menunjukkan SQL pernyataan yang berkontribusi paling besar terhadap beban DB.




Pilih salah satu tab dimensi berikut.

| Tab | Deskripsi | Mesin yang didukung |
| --- | --- | --- |
| Teratas SQL | SQLPernyataan yang sedang berjalan | Semua |
| Tunggu teratas | Peristiwa di mana backend basis data sedang menunggu | Semua |
| Host teratas | Nama host klien yang terhubung | Semua |
| Pengguna teratas | Pengguna yang masuk ke basis data | Semua |
| Aplikasi teratas | Nama aplikasi yang terhubung ke basis data | Aurora SQL |
| Jenis sesi teratas | Jenis sesi saat ini | Hanya Aurora Postgre SQL |

Untuk mempelajari cara menganalisis kueri dengan menggunakan SQL tab Top, lihatIkhtisar SQL tab Top.

## Mengakses dasbor Wawasan Performa

Amazon RDS menyediakan tampilan konsolidasi Performance Insights CloudWatch dan metrik di dasbor Performance Insights.

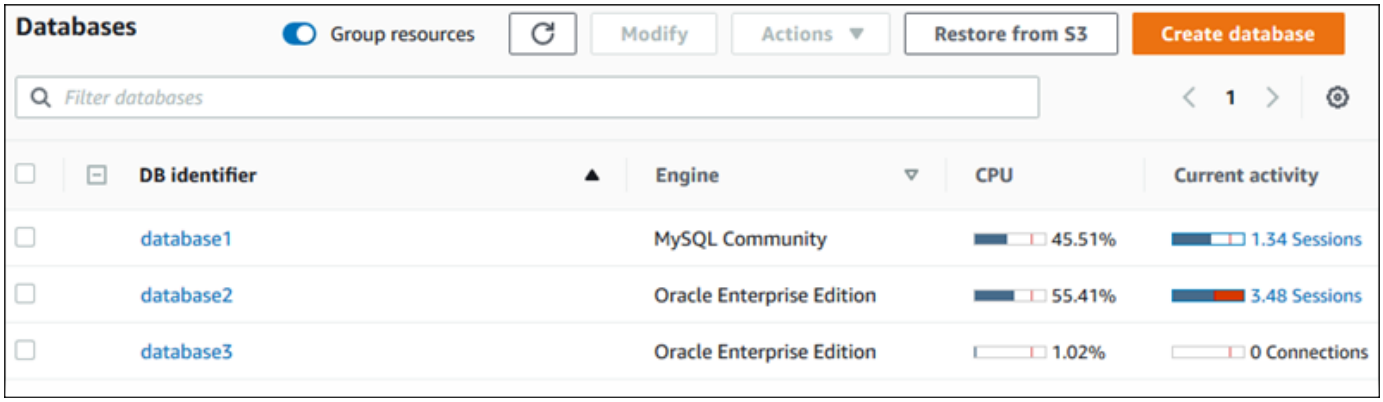
Untuk mengakses dasbor Wawasan Performa, gunakan prosedur berikut.

Untuk melihat dasbor Performance Insights di Konsol Manajemen AWS

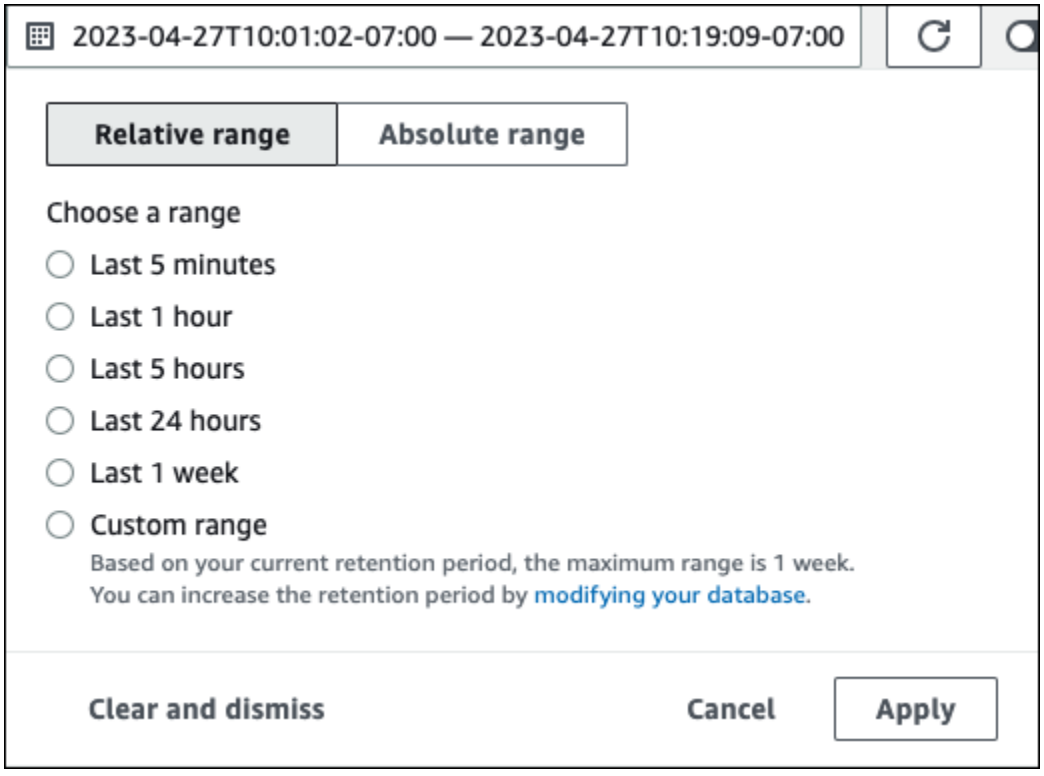
1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Untuk instans DB dengan Performance Insights diaktifkan, Anda juga dapat mengakses dasbor Performance Insights dengan memilih item Sessions dalam daftar instans DB. Di bagian Aktivitas saat ini, item Sesi menunjukkan muatan basis data dalam sesi aktif rata-rata selama lima menit terakhir. Bilah ini secara grafis menunjukkan muatan. Jika bilah kosong, berarti instans

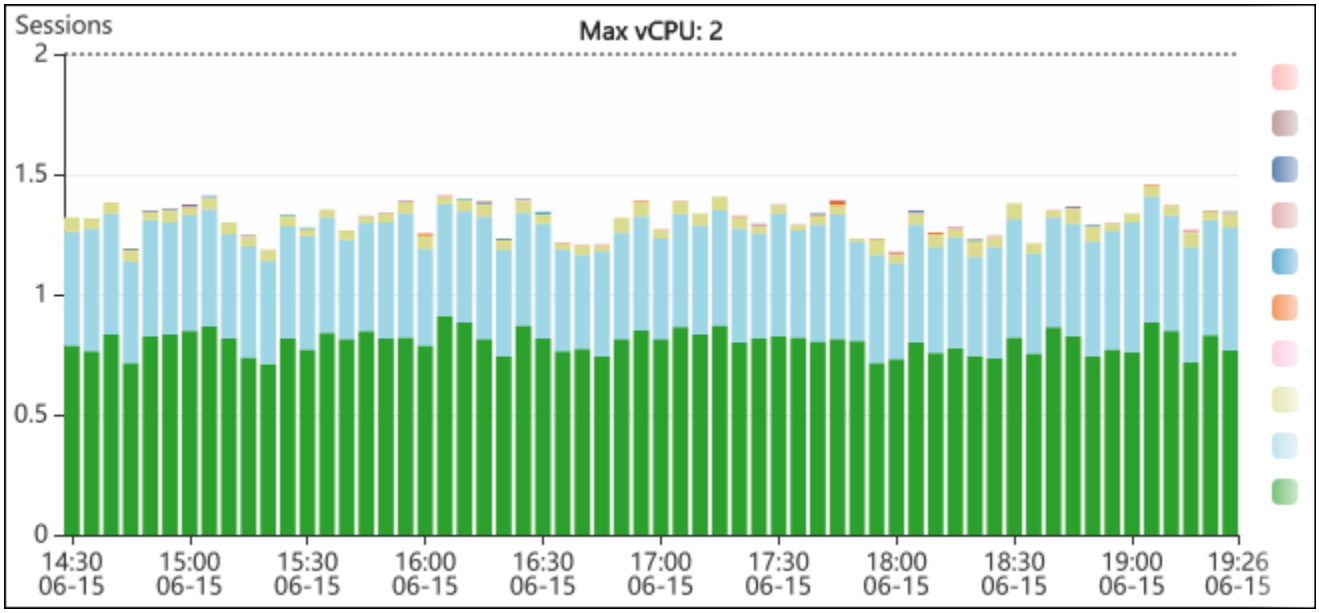
DB sedang diam. Saat muatan meningkat, bilah akan terisi dengan warna biru. Ketika beban melewati jumlah virtual CPUs (vCPUs) pada kelas instans DB, bilah berubah menjadi merah, menunjukkan potensi kemacetan.




4. (Opsional) Pilih rentang tanggal atau waktu di kanan atas dan tentukan interval waktu relatif atau absolut yang berbeda. Anda kini dapat menentukan periode waktu, dan menghasilkan laporan analisis performa basis data. Laporan ini berisi rekomendasi dan wawasan yang diidentifikasi. Untuk informasi selengkapnya, lihat Membuat laporan analisis kinerja di Performance Insights.

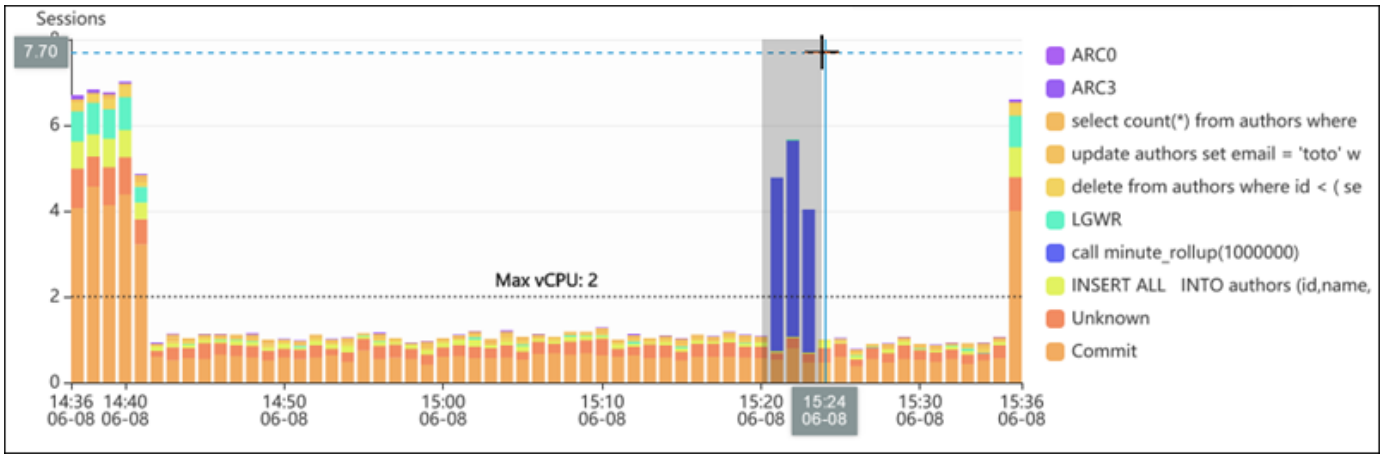



Di tangkapan layar berikut, interval muatan DB adalah 5 jam.

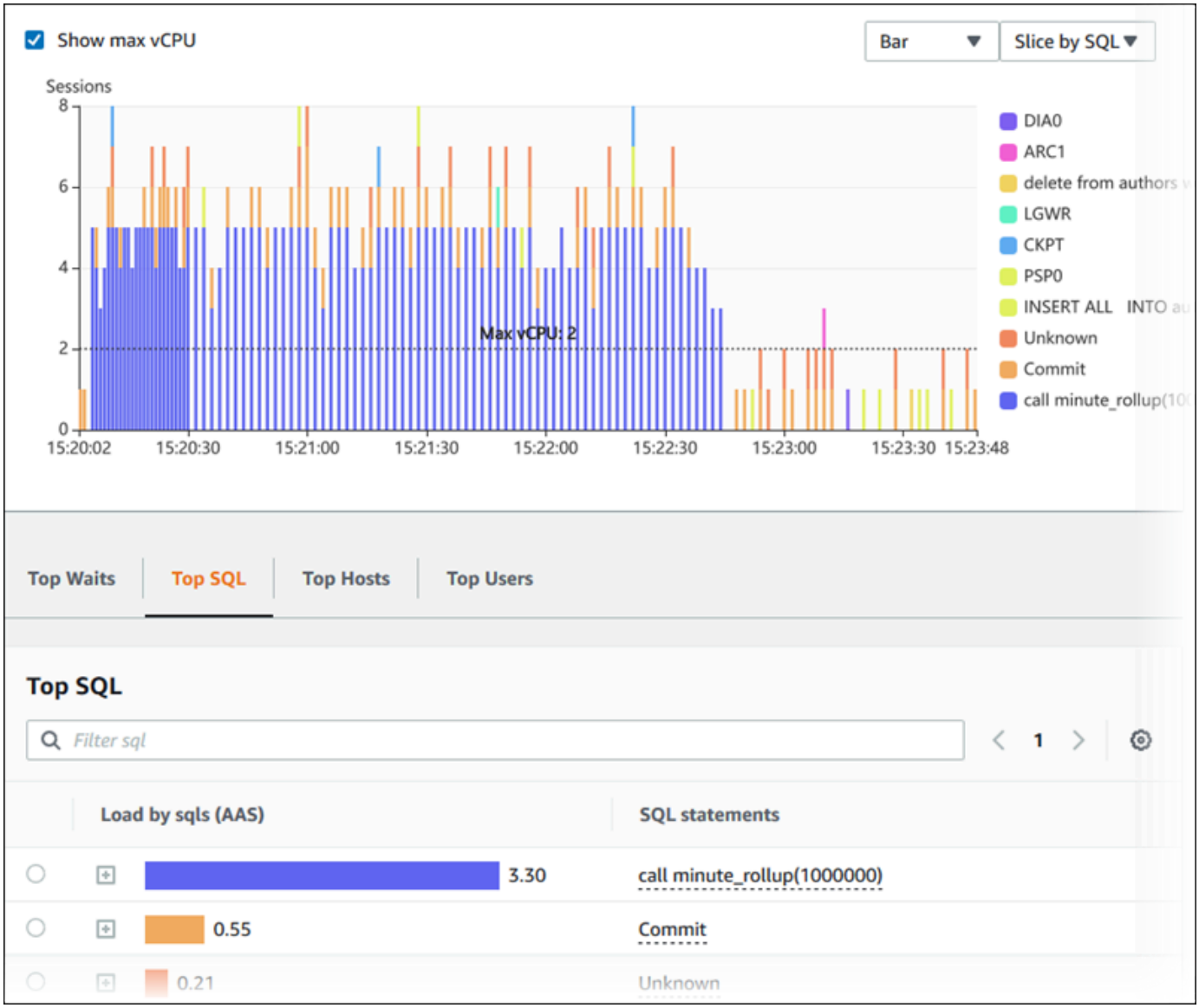



5. (Opsional) Untuk memperbesar sebagian bagan muatan DB, pilih waktu mulai dan seret ke akhir periode waktu yang diinginkan.

   Area yang dipilih disorot dalam bagan muatan DB.




   Saat Anda melepaskan mouse, bagan beban DB memperbesar AWS Wilayah yang dipilih, dan tabel Dimensi atas dihitung ulang.




6. (Opsional) Untuk menyegarkan data Anda secara otomatis, pilih Segarkan otomatis.

Dasbor Wawasan Performa secara otomatis disegarkan dengan data baru. Laju penyegaran bergantung pada jumlah data yang ditampilkan:

- Penyegaran 5 menit setiap 10 detik.
- Penyegaran 1 jam setiap 5 menit.
- Penyegaran 5 jam setiap 5 menit.
- Penyegaran 24 jam setiap 30 menit.
- Penyegaran 1 minggu setiap hari.

- Penyegaran 1 bulan setiap hari.

## Menganalisis muatan DB menurut peristiwa tunggu

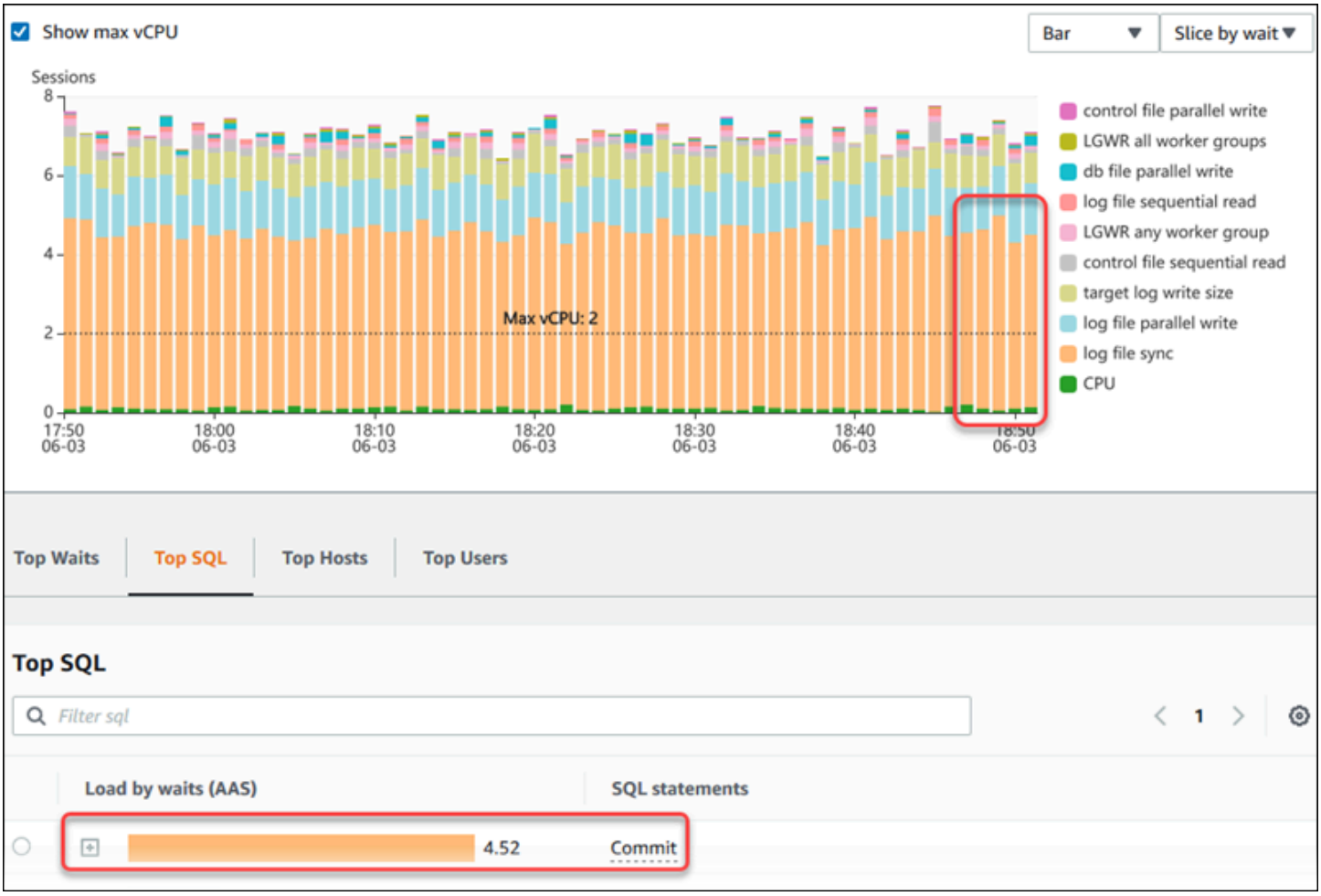
Jika bagan Muatan basis data menunjukkan kemacetan, Anda dapat mengetahui dari mana muatan tersebut berasal. Untuk melakukannya, lihat tabel item muatan teratas di bawah bagan Muatan basis data. Pilih item tertentu, seperti SQL kueri atau pengguna, untuk menelusuri item itu dan melihat detailnya.

Beban DB yang dikelompokkan berdasarkan menunggu dan SQL kueri teratas adalah tampilan dasbor Performance Insights default. Kombinasi ini biasanya memberikan wawasan paling banyak tentang masalah performa. Muatan DB yang dikelompokkan berdasarkan peristiwa tunggu menunjukkan apakah ada sumber daya atau kemacetan konkurensi dalam basis data. Dalam hal ini, SQLtab tabel item pemuatan atas menunjukkan kueri mana yang mendorong beban itu.

Alur kerja tipikal Anda untuk mendiagnosis masalah performa adalah sebagai berikut:

1. Tinjau bagan beban Database dan lihat apakah ada insiden beban database yang melebihi garis Max CPU.
2. Jika ada, lihat bagan Muatan basis data dan identifikasi satu atau beberapa status tunggu yang paling bertanggung jawab.
3. Identifikasi kueri intisari yang menyebabkan pemuatan dengan melihat kueri mana yang paling banyak disumbangkan oleh SQLtab pada tabel item pemuatan teratas pada status tunggu tersebut. Anda dapat mengidentifikasinya berdasarkan kolom Muatan DB berdasarkan peristiwa Tunggu.
4. Pilih salah satu kueri intisari ini di SQLtab untuk memperluasnya dan melihat kueri anak yang terdiri darinya.

Misalnya, di dasbor berikut, peristiwa tunggu sinkronisasi file log menyumbang sebagian besar muatan DB. LGWRSemua kelompok pekerja menunggu juga tinggi. SQLBagan teratas menunjukkan apa yang menyebabkan sinkronisasi file log menunggu: `COMMIT` pernyataan yang sering. Dalam kasus ini, eksekusi yang lebih jarang akan mengurangi muatan DB.




## Menganalisis performa basis data selama periode waktu tertentu

Menganalisis kinerja database dengan analisis on-demand dengan membuat laporan analisis kinerja untuk jangka waktu tertentu. Lihat laporan analisis kinerja untuk menemukan masalah kinerja, seperti kemacetan sumber daya atau perubahan dalam kueri di instans DB Anda. Dasbor Wawasan Performa memungkinkan Anda memilih periode waktu dan membuat laporan analisis performa. Anda juga dapat menambahkan satu tag atau lebih ke laporan.

Untuk menggunakan fitur ini, Anda harus menggunakan periode retensi tingkat berbayar. Untuk informasi selengkapnya, silakan lihat Harga dan retensi data untuk Wawasan Performa

Laporan ini dapat dipilih dan dilihat di tab Laporan analisis performa - baru. Laporan ini berisi wawasan, metrik terkait, dan rekomendasi untuk menyelesaikan masalah performa. Laporan ini dapat dilihat selama periode retensi Wawasan Performa.

Laporan dihapus jika waktu mulai periode analisis laporan berada di luar periode retensi. Anda juga dapat menghapus laporan sebelum periode retensi berakhir.

Untuk mendeteksi masalah performa dan menghasilkan laporan analisis untuk instans DB, Anda harus mengaktifkan Wawasan Performa. Untuk informasi selengkapnya tentang mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora.

Untuk informasi dukungan wilayah, mesin DB, dan kelas instans untuk fitur ini, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Di bagian berikut, Anda dapat membuat, melihat, menambahkan tag, dan menghapus laporan analisis kinerja.

Topik

- Membuat laporan analisis kinerja di Performance Insights
- Melihat laporan analisis kinerja di Performance Insights
- Menambahkan tag ke laporan analisis kinerja di Performance Insights
- Menghapus laporan analisis kinerja di Performance Insights

## Membuat laporan analisis kinerja di Performance Insights

Anda dapat membuat laporan analisis performa selama periode tertentu di dasbor Wawasan Performa. Anda dapat memilih periode waktu dan menambahkan satu tag atau lebih ke laporan analisis.

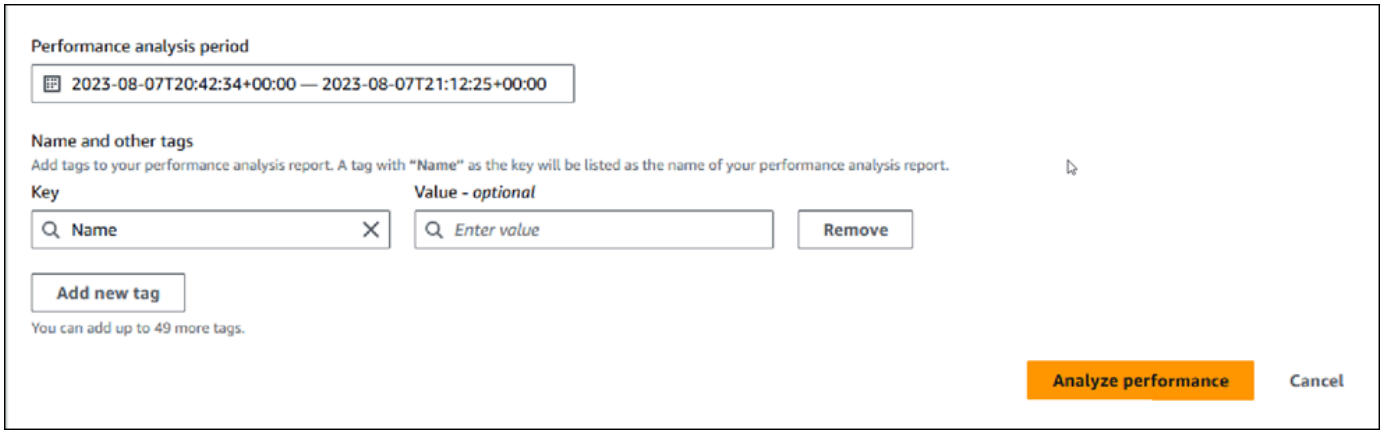
Periode analisis bisa berkisar dari 5 menit hingga 6 hari. Harus ada data performa setidaknya 24 jam sebelum waktu mulai analisis.

Untuk informasi dukungan wilayah, mesin DB, dan kelas instans untuk fitur ini, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Untuk membuat laporan analisis performa selama periode waktu tertentu

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Pilih Analisis kinerja di bagian Pemuatan database di dasbor Performance Insights.

   Kolom untuk mengatur periode waktu dan menambahkan satu atau beberapa tag ke laporan analisis performa ditampilkan.

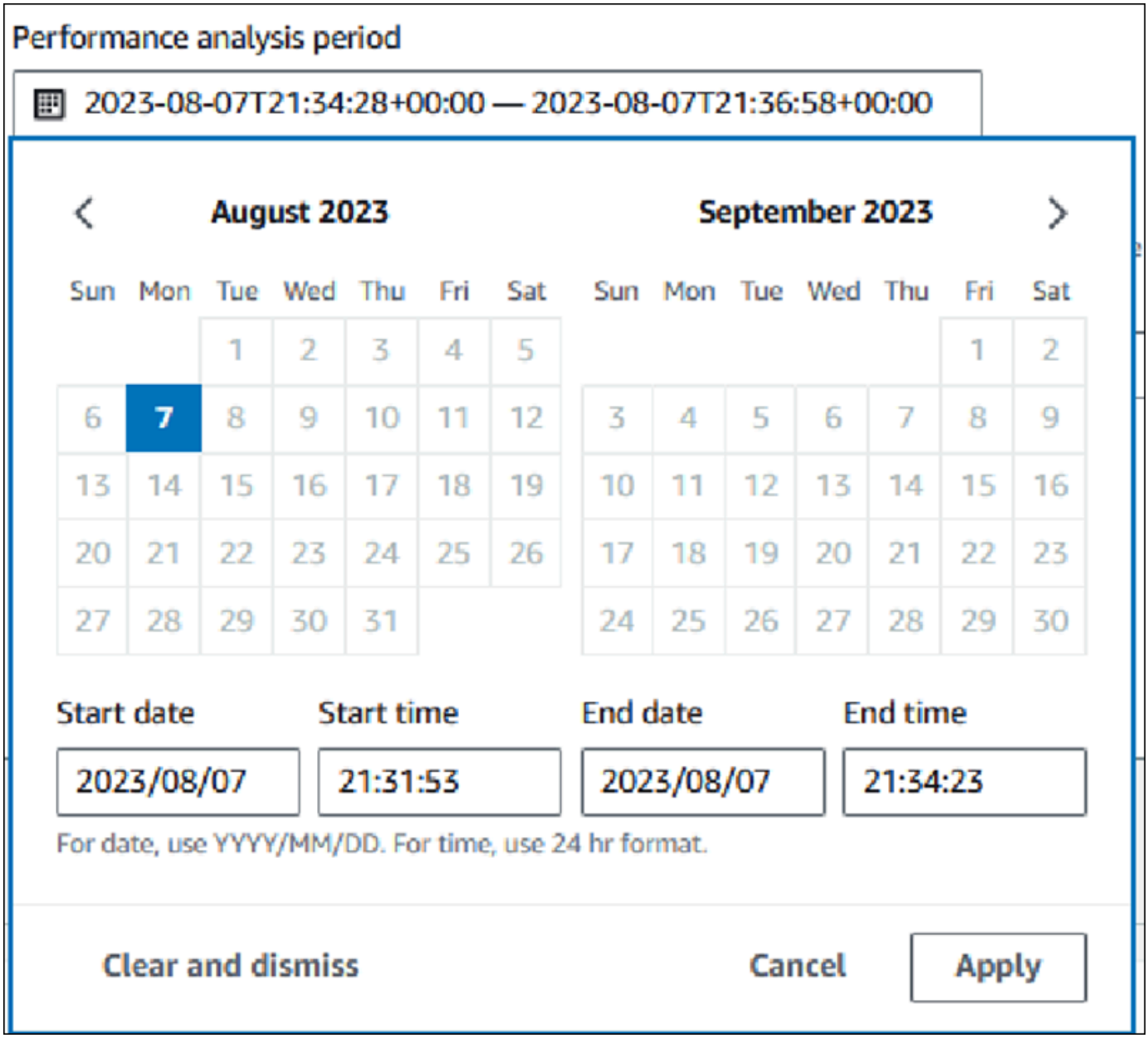



5. Pilih periode waktu. Jika menetapkan periode waktu dalam Rentang relatif atau Rentang absolut di kanan atas, Anda hanya dapat memasukkan atau memilih tanggal dan waktu laporan analisis dalam periode waktu ini. Jika Anda memilih periode analisis di luar periode waktu ini, pesan kesalahan akan muncul.

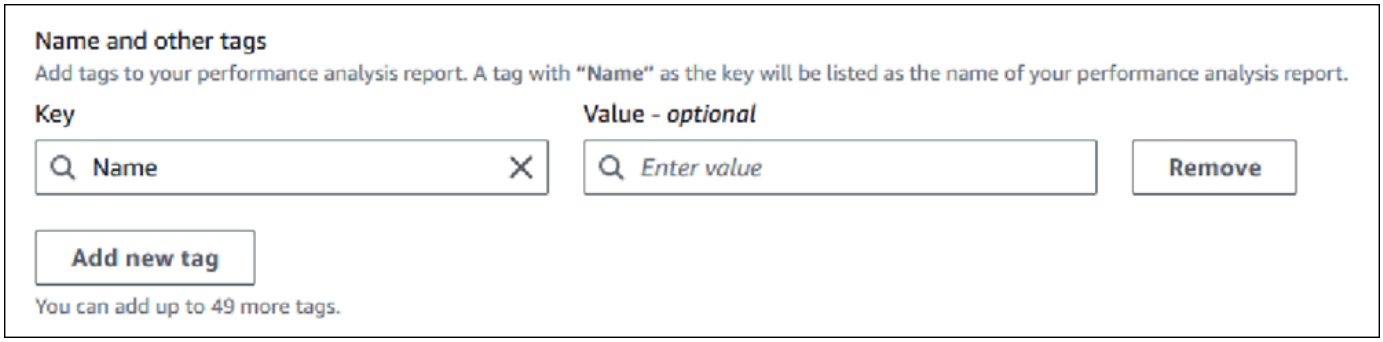
   Untuk mengatur periode waktu, Anda dapat mengikuti langkah-langkah berikut:

   - Tekan dan seret salah satu slider pada bagan muatan DB.

     Kotak Periode analisis performa menampilkan periode waktu yang dipilih dan bagan muatan DB menyoroti periode waktu yang dipilih.
   - Pilih Tanggal mulai, Waktu mulai, Tanggal akhir, dan Waktu akhir di kotak Periode analisis performa.




6. (Opsional) Masukkan Kunci dan Nilai-opsional untuk menambahkan tag untuk laporan.



7. Pilih Analisis performa.

   Spanduk menampilkan pesan apakah pembuatan laporan berhasil atau gagal. Pesan juga menyediakan tautan untuk melihat laporan.

   Contoh berikut menunjukkan spanduk dengan pesan pembuatan laporan berhasil.

   Laporan ini dapat dilihat di tab Laporan analisis performa - baru.

Anda dapat membuat laporan analisis performa menggunakan AWS CLI. Untuk contoh tentang cara membuat laporan menggunakan AWS CLI, lihatMembuat laporan analisis performa selama periode waktu tertentu.

Melihat laporan analisis kinerja di Performance Insights

Tab Laporan analisis performa - baru mencantumkan semua laporan yang dibuat untuk instans DB. Berikut ini ditampilkan untuk setiap laporan:

- ID: Pengidentifikasi unik laporan.
- Nama: Kunci tag yang ditambahkan ke laporan.
- Waktu pembuatan laporan: Waktu Anda membuat laporan.
- Waktu mulai analisis: Waktu mulai analisis dalam laporan.
- Waktu akhir analisis: Waktu akhir analisis dalam laporan.

Untuk melihat laporan analisis performa

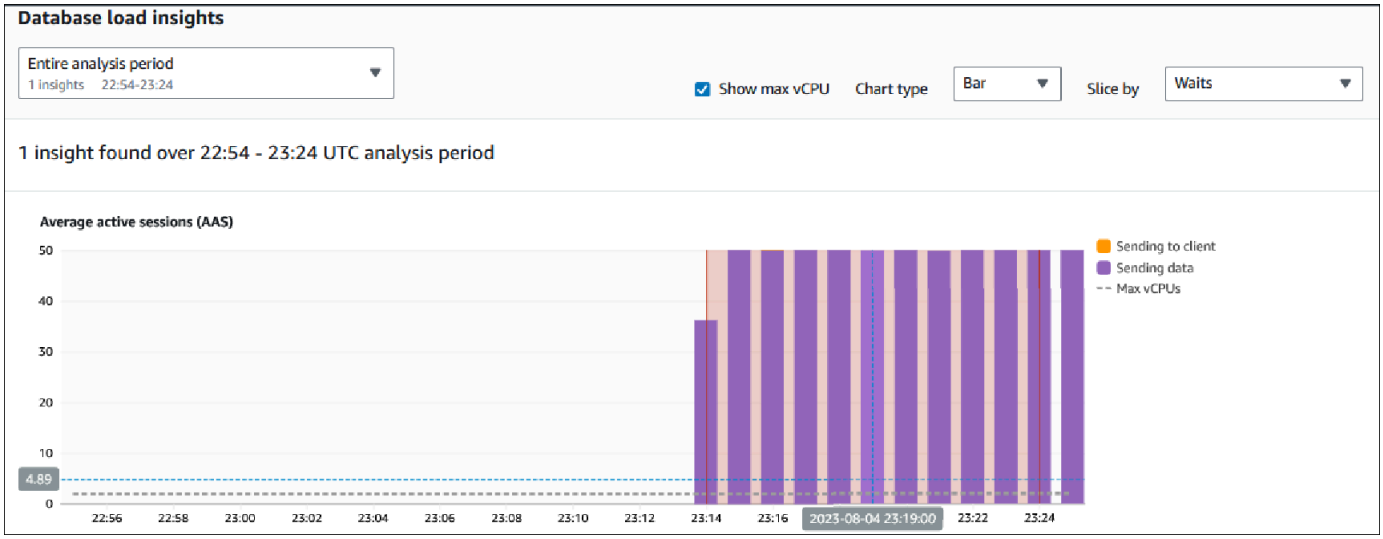
1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB yang laporan analisisnya ingin dilihat.
4. Gulir ke bawah dan pilih Laporan analisis kinerja - tab baru di dasbor Performance Insights.

   Semua laporan analisis untuk periode waktu yang berbeda ditampilkan.
5. Pilih ID laporan yang ingin dilihat.

Bagan muatan DB menampilkan seluruh periode analisis secara default jika ada lebih dari satu wawasan yang diidentifikasi. Jika laporan telah mengidentifikasi satu wawasan, bagan muatan DB akan menampilkan wawasan secara default.

Dasbor juga mencantumkan tag untuk laporan di bagian Tag.

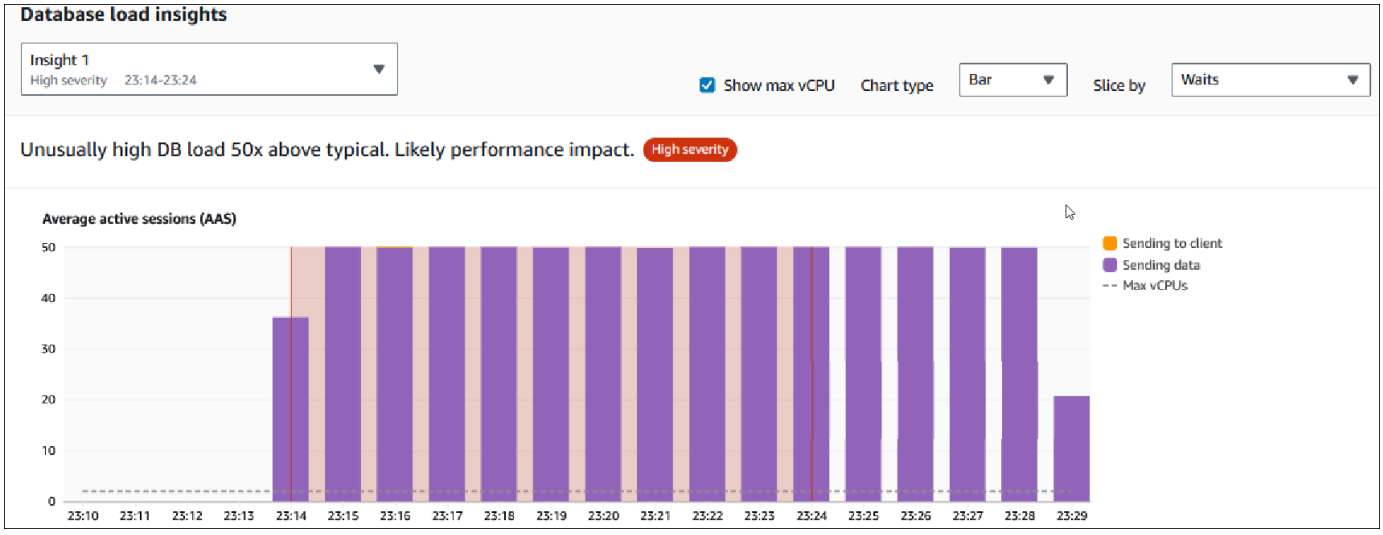
Contoh berikut menunjukkan seluruh periode analisis untuk laporan.

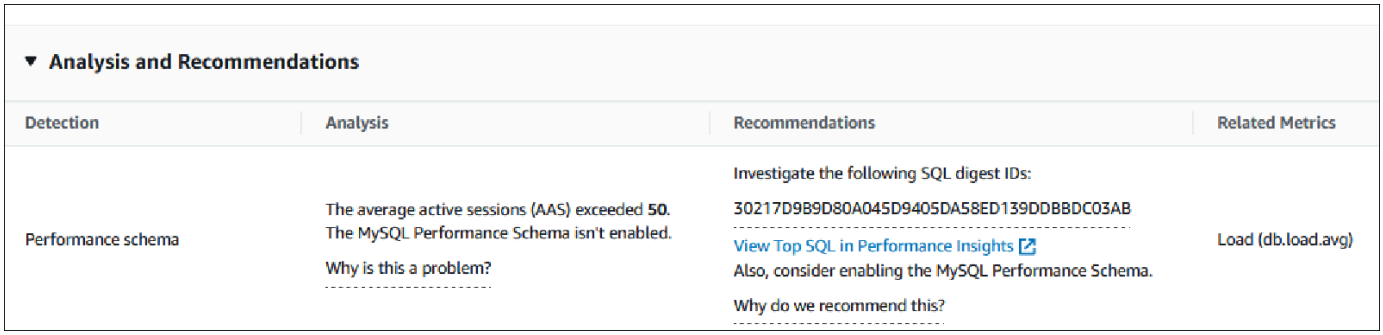



6. Pilih wawasan dalam daftar Wawasan muatan basis data yang ingin dilihat jika ada lebih dari satu wawasan yang diidentifikasi dalam laporan.

Dasbor menampilkan pesan wawasan, bagan muatan DB yang menyoroti periode waktu wawasan, analisis dan rekomendasi, serta daftar tag laporan.

Contoh berikut menunjukkan wawasan muatan DB dalam laporan.






Menambahkan tag ke laporan analisis kinerja di Performance Insights

Anda dapat menambahkan tag saat membuat atau melihat laporan. Anda dapat menambahkan hingga 50 tag untuk sebuah laporan.

Anda memerlukan izin untuk menambahkan tag. Untuk informasi selengkapnya tentang kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa

Untuk menambahkan satu atau beberapa tag saat membuat laporan, lihat langkah 6 dalam prosedur Membuat laporan analisis kinerja di Performance Insights.

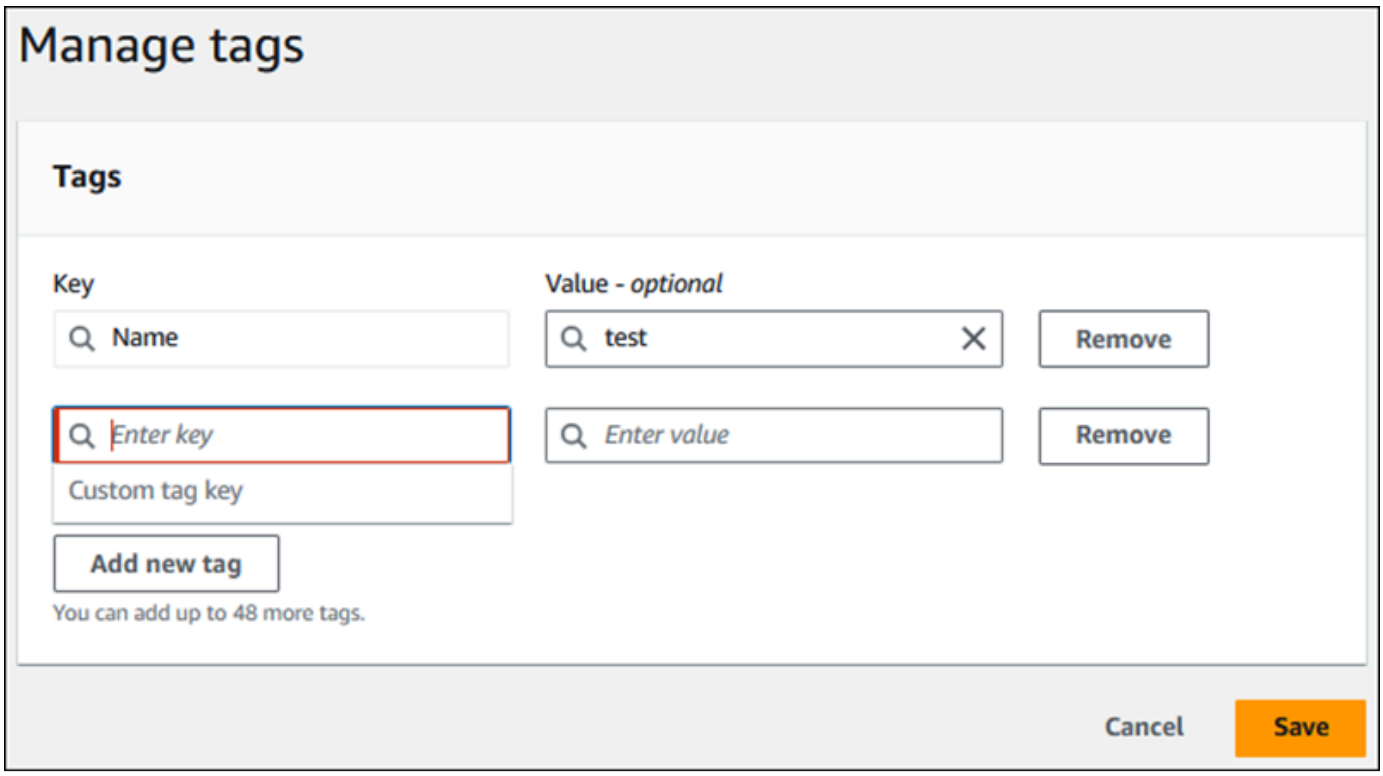
Untuk menambahkan satu tag atau lebih saat melihat laporan

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa muncul untuk instans DB.
4. Gulir ke bawah dan pilih tab Laporan analisis performa - baru.
5. Pilih laporan yang ingin diberi tag.

   Dasbor menampilkan laporan.
6. Gulir ke bawah ke Tag dan pilih Kelola tag.
7. Pilih Tambahkan tag baru.
8. Masukkan Kunci dan Nilai - opsional, dan pilih Tambahkan tag baru.

   Contoh berikut memberikan opsi untuk menambahkan tag baru untuk laporan yang dipilih.




Tag baru dibuat untuk laporan.

Daftar tag untuk laporan ditampilkan di bagian Tag pada dasbor. Jika Anda ingin menghapus tag dari laporan, pilih Hapus di samping tag.

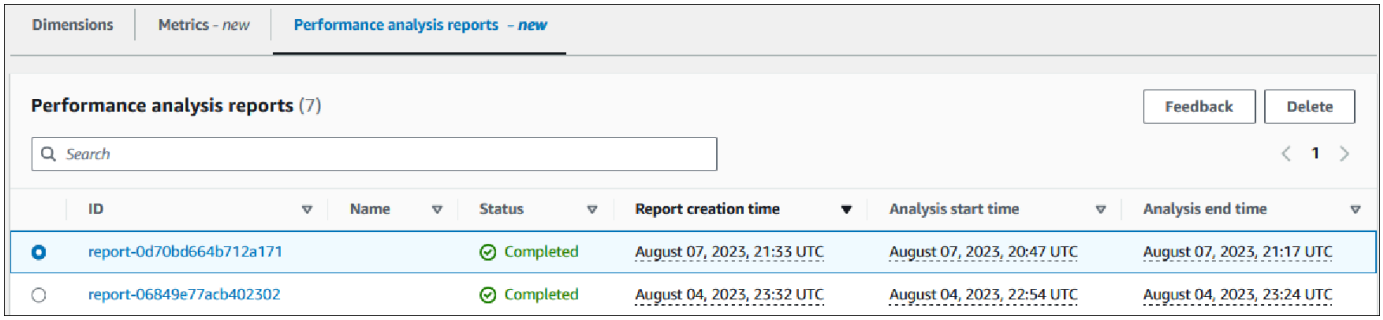
## Menghapus laporan analisis kinerja di Performance Insights

Anda dapat menghapus laporan dari daftar laporan yang ditampilkan di tab Laporan analisis performa atau saat melihat laporan.

Untuk menghapus laporan

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa muncul untuk instans DB.
4. Gulir ke bawah dan pilih tab Laporan analisis performa - baru.
5. Pilih laporan yang ingin dihapus dan pilih Hapus di kanan atas.




Jendela konfirmasi ditampilkan. Laporan akan dihapus setelah Anda memilih konfirmasi.

6. (Opsional) Pilih ID laporan yang ingin dihapus.

   Di halaman laporan, pilih Hapus di kanan atas.

   Jendela konfirmasi ditampilkan. Laporan akan dihapus setelah Anda memilih konfirmasi.

## Menganalisis kueri dengan SQL tab Teratas di Performance Insights

Di dasbor Amazon RDS Performance Insights, Anda dapat menemukan informasi tentang menjalankan dan kueri terbaru di SQL tab Atas di tabel Dimensi teratas. Anda dapat menggunakan informasi ini untuk menyetel kueri Anda.

Topik

- Ikhtisar SQL tab Top
- Mengakses lebih banyak teks SQL di dasbor Wawasan Performa
- Melihat SQL statistik di dasbor Performance Insights

### Ikhtisar SQL tab Top

Secara default, SQL tab Top menunjukkan 25 kueri yang paling berkontribusi pada pemuatan DB. Untuk membantu menyetel kueri, Anda dapat menganalisis informasi seperti teks kueri dan SQL statistik. Anda juga dapat memilih statistik yang ingin Anda tampilkan di SQL tab Atas.

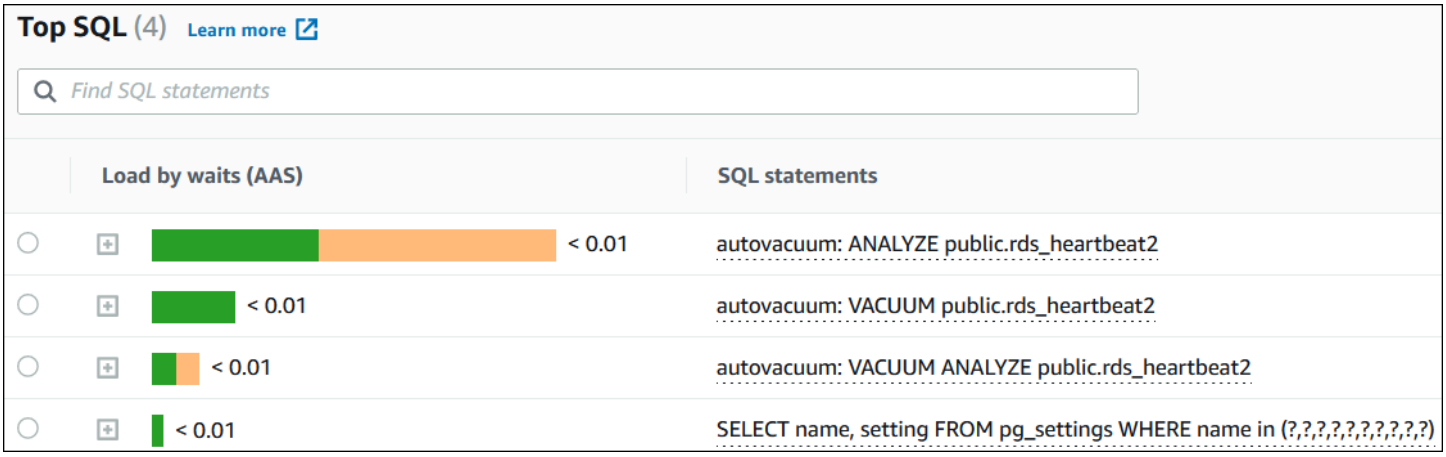
Topik

- SQLteks
- SQLstatistik
- Muat dengan menunggu () AAS
- Lihat SQL informasi

- Pilih preferensi statistik

SQLteks

Secara default, setiap baris di SQL tabel Top menunjukkan 500 byte teks untuk setiap pernyataan.

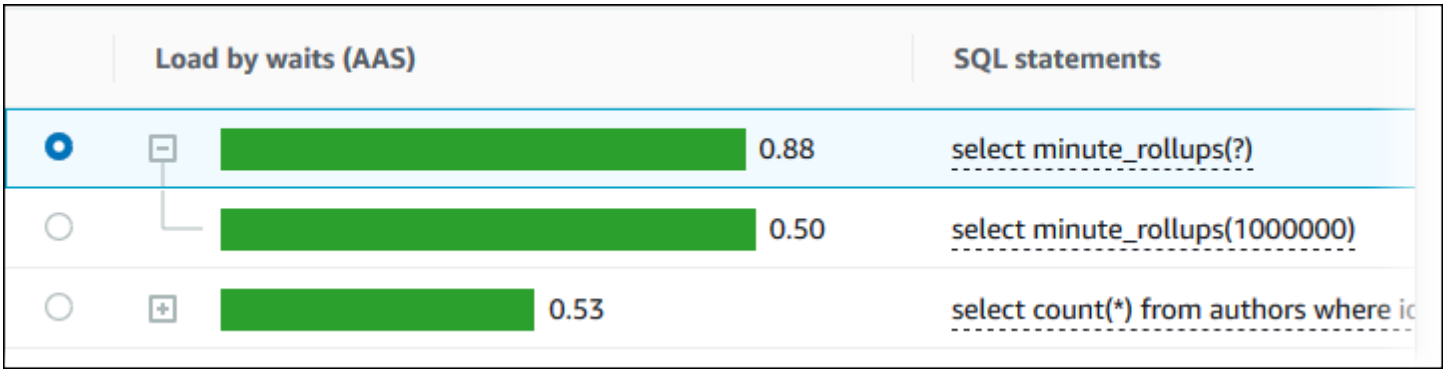



Untuk mempelajari cara melihat lebih dari 500 byte SQL teks default, lihatMengakses lebih banyak teks SQL di dasbor Wawasan Performa.

SQLIntisari adalah gabungan dari beberapa kueri aktual yang secara struktural serupa tetapi mungkin memiliki nilai literal yang berbeda. Digest menggantikan nilai berkode keras dengan tanda tanya. Misalnya, digest mungkin berupa `SELECT * FROM emp WHERE lname= ?`. Digest ini dapat mencakup kueri turunan berikut:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Untuk melihat SQL pernyataan literal dalam intisari, pilih kueri, lalu pilih simbol plus (+). Dalam contoh berikut, kueri yang dipilih adalah digest.

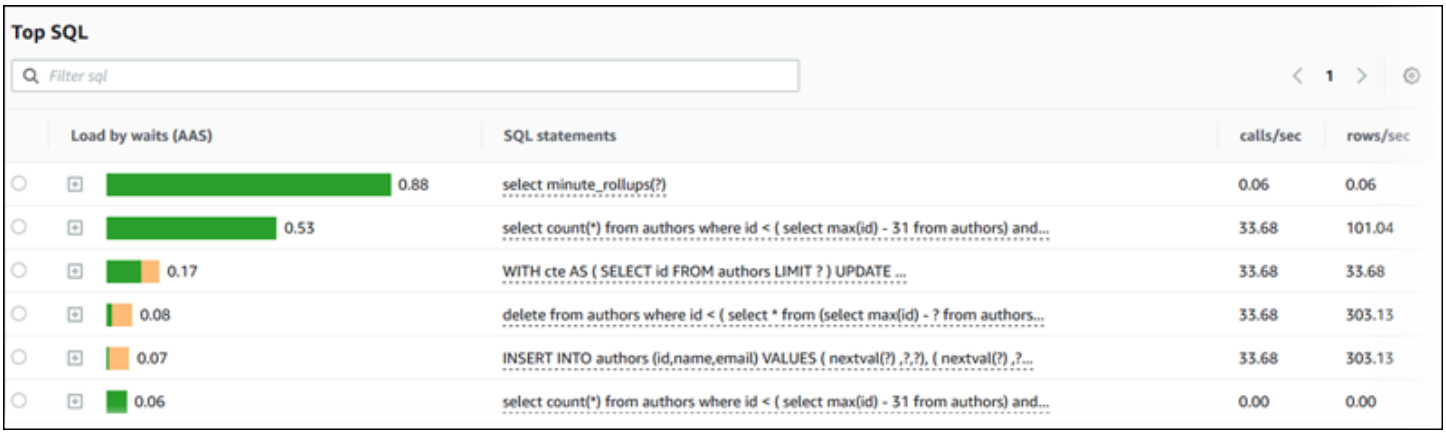


> **Note**
>
> SQLIntisari mengelompokkan SQL pernyataan serupa, tetapi tidak menyunting informasi sensitif.

### SQLstatistik

SQLstatistik adalah metrik terkait kinerja tentang kueri. SQL Misalnya, Wawasan Performa mungkin menampilkan eksekusi per detik atau baris yang diproses per detik. Wawasan Performa mengumpulkan statistik hanya untuk kueri yang paling umum. Biasanya, ini cocok dengan kueri teratas berdasarkan muatan yang ditampilkan di dasbor Wawasan Performa.

Setiap baris di SQL Tabel Top menunjukkan statistik yang relevan untuk SQL pernyataan atau intisari, seperti yang ditunjukkan pada contoh berikut.




Performance Insights dapat melaporkan `0.00` dan `-` (tidak diketahui) untuk SQL statistik. Situasi ini terjadi dalam kondisi berikut:

- Hanya ada satu sampel. Misalnya, Performance Insights menghitung tingkat perubahan untuk SQL kueri Aurora Postgre berdasarkan beberapa sampel dari tampilan. `pg_stat_statements` Ketika beban kerja berjalan untuk waktu yang singkat, Wawasan Performa mungkin hanya mengumpulkan satu sampel, yang berarti tidak dapat menghitung tingkat perubahan. Nilai yang tidak diketahui ditunjukkan dengan tanda hubung (`-`).
- Dua sampel memiliki nilai yang sama. Wawasan Performa tidak dapat menghitung tingkat perubahan karena tidak ada perubahan yang terjadi, sehingga melaporkan tingkatnya sebagai `0.00`.
- SQLPernyataan Aurora Postgre tidak memiliki pengenal yang valid. Postgre SQL membuat identifier untuk pernyataan hanya setelah parsing dan analisis. Dengan demikian, sebuah

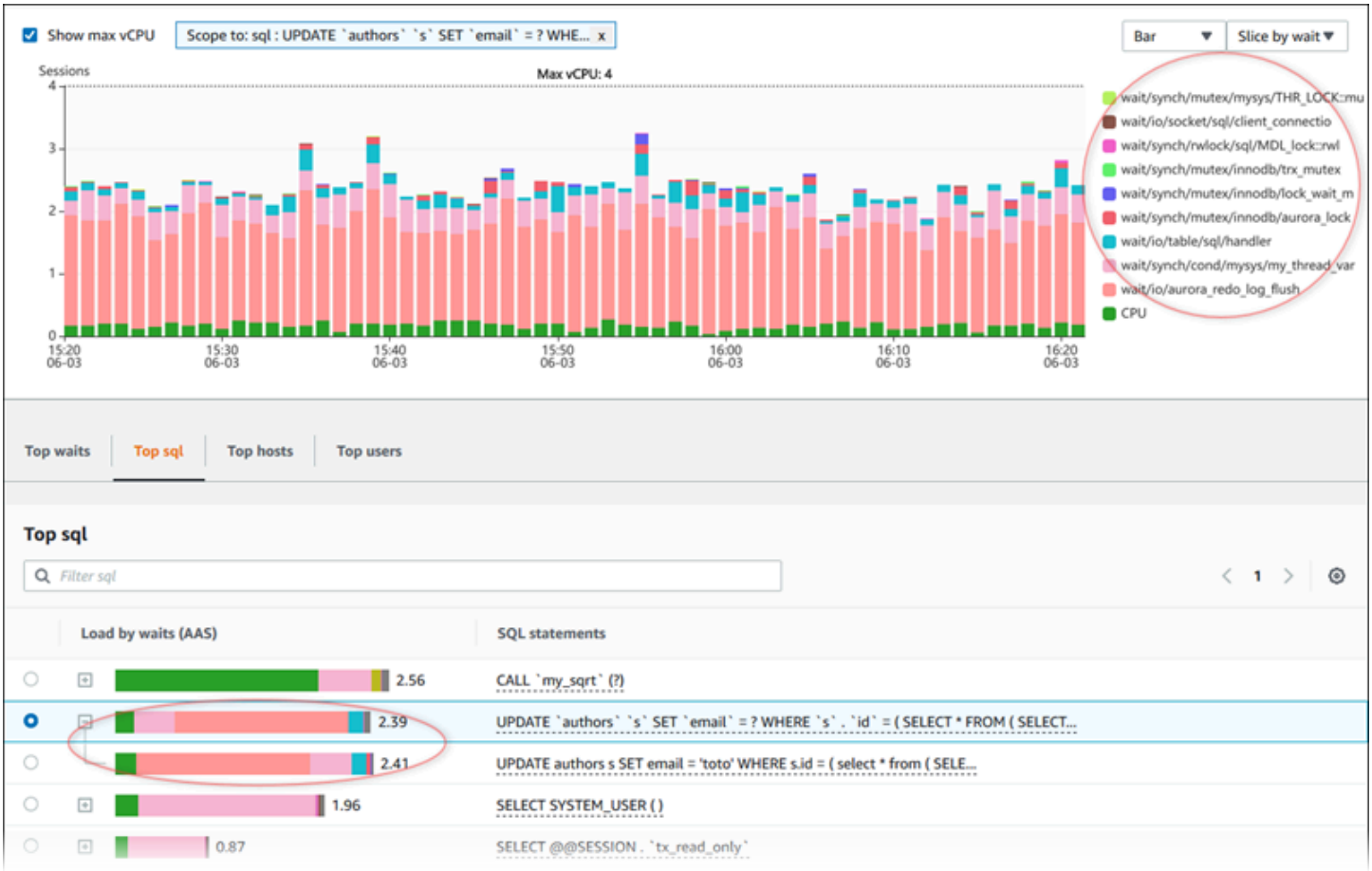
pernyataan dapat ada dalam struktur SQL internal dalam memori Postgre tanpa pengidentifikasi. Karena Wawasan Performa mengambil sampel struktur internal dalam memori sekali per detik, kueri latensi rendah mungkin muncul hanya untuk satu sampel. Jika ID kueri tidak tersedia untuk sampel ini, Wawasan Performa tidak dapat mengaitkan pernyataan ini dengan statistiknya. Nilai yang tidak diketahui ditunjukkan dengan tanda hubung (-).

Untuk deskripsi SQL statistik untuk mesin Aurora, lihat. Statistik SQL untuk Wawasan Performa

Muat dengan menunggu () AAS

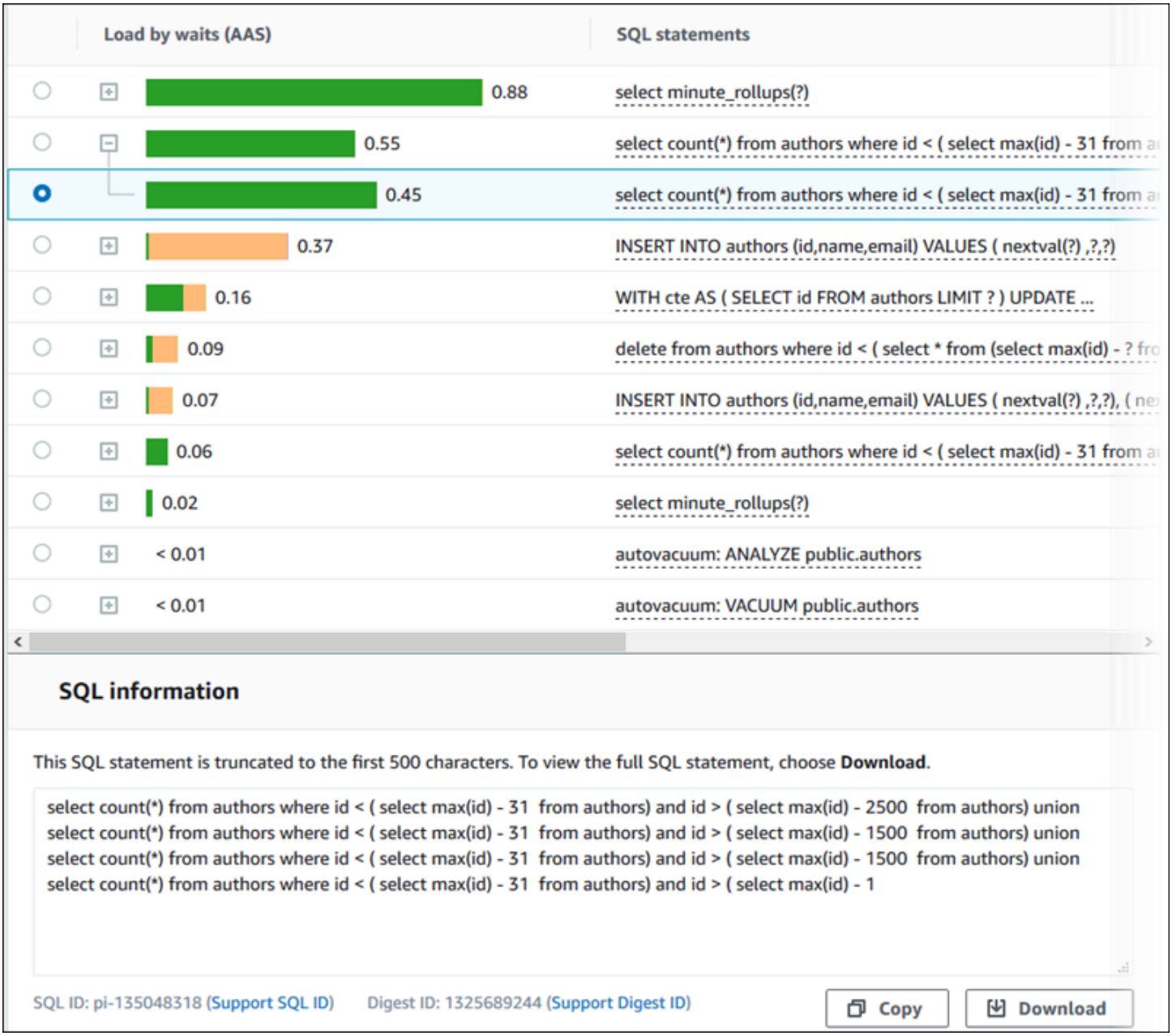
Di Atas SQL, kolom Load by waits (AAS) menggambarkan persentase beban database yang terkait dengan setiap item beban teratas. Kolom ini menunjukkan muatan untuk item tersebut berdasarkan pengelompokan apa pun yang saat ini dipilih di Bagan Muatan DB. Untuk informasi selengkapnya tentang Rata-rata sesi aktif (AAS), lihatSesi aktif rata-rata.

Misalnya, Anda mungkin mengelompokkan bagan Muatan DB berdasarkan status tunggu. Anda memeriksa SQL kueri di tabel item pemuatan teratas. Dalam kasus ini, bilah Muatan DB berdasarkan Status Tunggu diberi ukuran, disegmentasi, dan diberi kode warna untuk menunjukkan seberapa banyak status tunggu tertentu yang dikontribusikan oleh kueri. Bilah ini juga menunjukkan status tunggu yang memengaruhi kueri yang dipilih.




Lihat SQL informasi

Di SQL tabel Atas, Anda dapat membuka pernyataan untuk melihat informasinya. Informasi ini muncul di panel bawah.

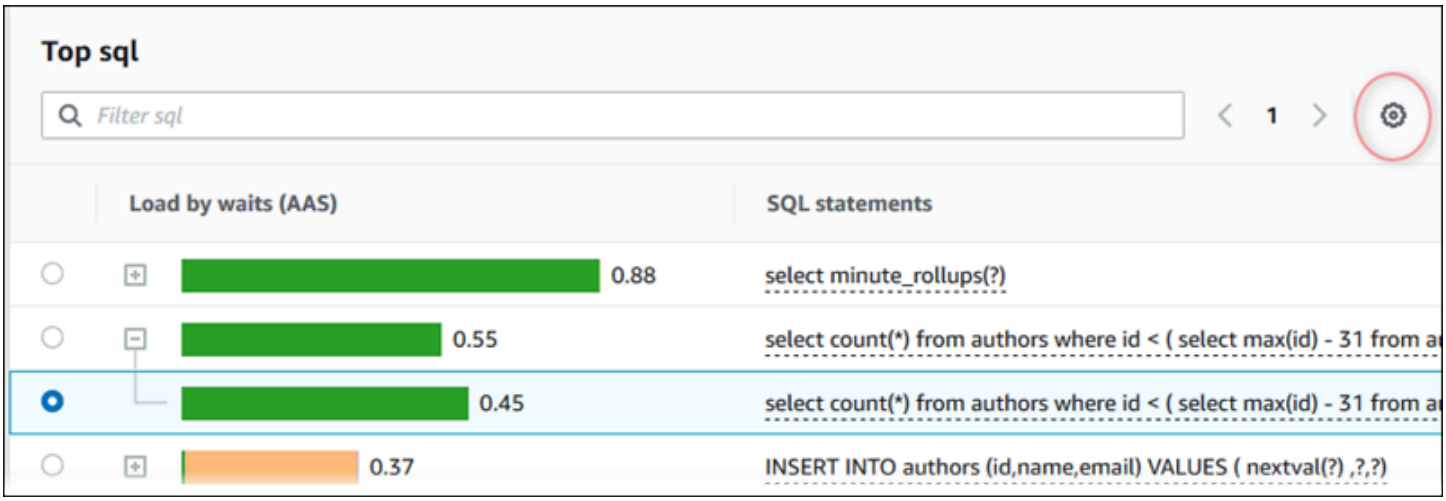



Berikut jenis identifier (IDs) yang terkait dengan SQL pernyataan:

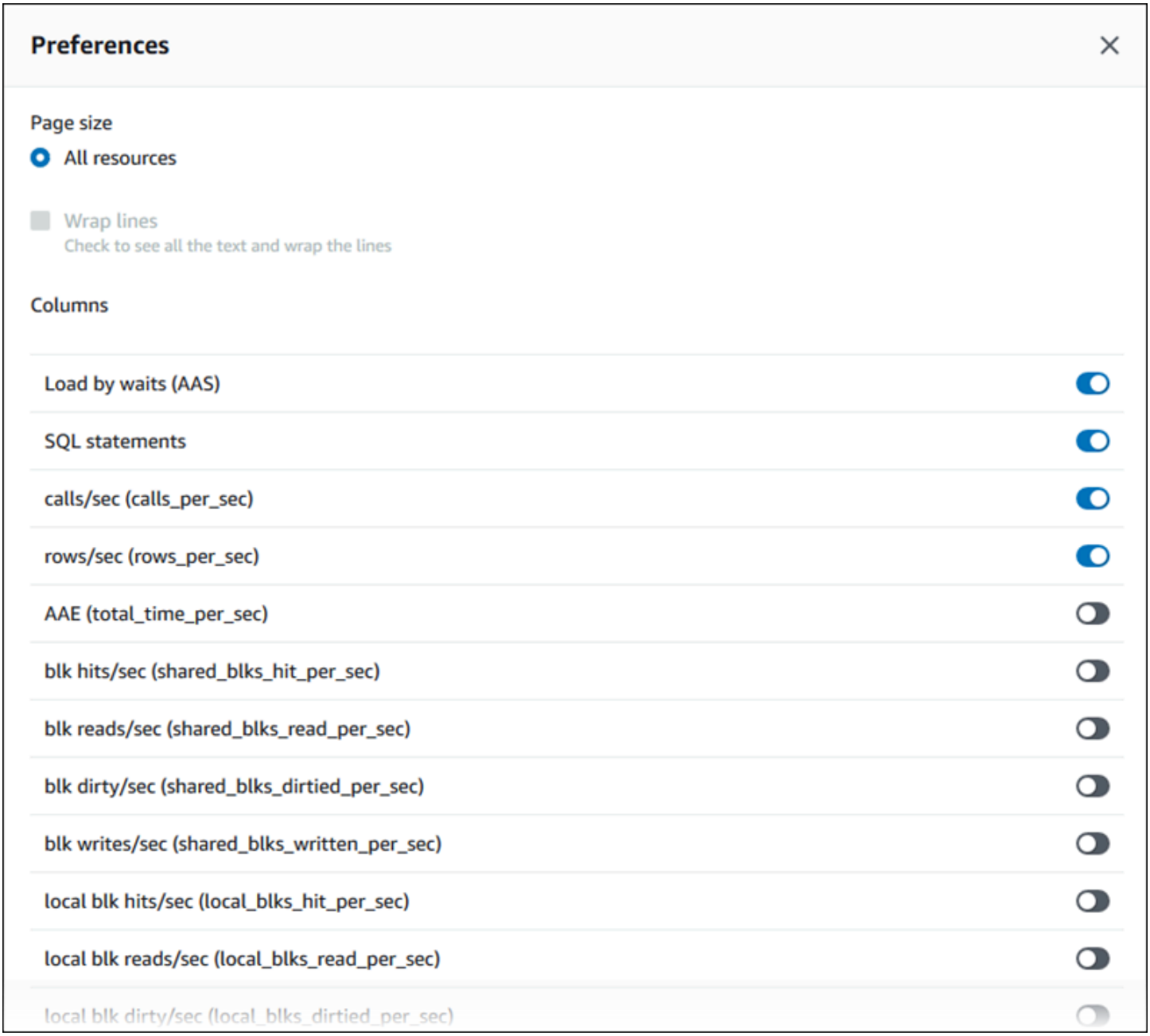
- Support SQL ID — Nilai hash dari SQL ID. Nilai ini hanya untuk mereferensikan SQL ID saat Anda bekerja dengan AWS Support. AWS Support tidak memiliki akses ke aktual SQL IDs dan SQL teks Anda.
- Support Digest ID – Nilai hash dari ID digest. Nilai ini hanya untuk mereferensikan ID intisari saat Anda bekerja dengan Support AWS . AWS Support tidak memiliki akses ke intisari IDs dan SQL teks Anda yang sebenarnya.

Pilih preferensi statistik

Anda dapat mengontrol statistik yang ditampilkan di SQL tab Atas dengan memilih ikon Preferensi.




Saat memilih ikon Preferensi, jendela Preferensi akan terbuka. Tangkapan layar berikut adalah contoh jendela Preferensi.




Untuk mengaktifkan statistik yang ingin Anda tampilkan di SQL tab Atas, gunakan mouse Anda untuk menggulir ke bagian bawah jendela, lalu pilih Lanjutkan.

Untuk informasi selengkapnya tentang statistik per detik atau per panggilan untuk mesin Amazon , lihat bagian statistik khusus mesin di SQL Statistik SQL untuk Wawasan Performa

Mengakses lebih banyak teks SQL di dasbor Wawasan Performa

Secara default, setiap baris dalam tabel SQL Teratas menunjukkan 500 byte teks SQL untuk setiap pernyataan SQL.

Saat pernyataan SQL melebihi 500 byte, Anda dapat melihat lebih banyak teks di bagian Teks SQL di bawah tabel SQL Teratas. Dalam hal ini, panjang maksimum teks yang ditampilkan dalam Teks SQL adalah 4 KB. Batas ini diperkenalkan oleh konsol dan tunduk pada batas yang ditetapkan oleh mesin basis data. Untuk menyimpan teks yang ditampilkan dalam Teks SQL, pilih Unduh.

Topik

- Batas ukuran teks untuk Aurora MySQL
- Menyetel batas SQL teks untuk DB SQL
- Melihat dan mengunduh SQL teks di dasbor Performance Insights

Batas ukuran teks untuk Aurora MySQL

Saat Anda mengunduh teks SQL, mesin basis data menentukan panjang maksimumnya. Anda dapat mengunduh teks SQL hingga batas per mesin berikut.

| Mesin DB | Panjang maksimum teks yang diunduh |
| --- | --- |
| Aurora MySQL | Panjangnya tetap pada 4.096 byte. |

Bagian Teks SQL dari konsol Wawasan Performa menampilkan hingga maksimum yang ditampilkan mesin. Misalnya, jika Aurora MySQL menampilkan paling banyak 1 KB ke Wawasan Performa, Aurora MySQL hanya dapat mengumpulkan dan menampilkan 1 KB, meskipun kueri asalnya lebih besar. Jadi, jika Anda melihat kueri dalam Teks SQL atau mengunduhnya, Wawasan Performa akan menampilkan jumlah byte yang sama.

Jika Anda menggunakan AWS CLI atau API, Performance Insights tidak memiliki batas 4 KB yang diberlakukan oleh konsol. `DescribeDimensionKeys`dan `GetResourceMetrics` kembali paling banyak 500 byte.

> **Note**
>
> `GetDimensionKeyDetails`mengembalikan kueri lengkap, tetapi ukurannya tunduk pada batas mesin.

Menyetel batas SQL teks untuk DB SQL

Postgre menangani teks secara berbeda. Anda dapat mengatur batas ukuran teks dengan parameter instans DB `track_activity_query_size`. Parameter ini memiliki karakteristik sebagai berikut:

Ukuran teks default

Di Postgre SQL versi 9.6, pengaturan default untuk parameter adalah 1.024 byte. `track_activity_query_size` Di Postgre SQL versi 10 atau lebih tinggi, defaultnya adalah 4.096 byte.

Ukuran teks maksimum

Batasnya `track_activity_query_size` adalah 102.400 byte untuk dan lebih rendah. SQL Ukuran maksimumnya adalah 1 MB untuk versi 13 dan yang lebih baru.

Jika mesin menampilkan 1 MB ke Wawasan Performa, konsol hanya akan menampilkan 4 KB pertama. Jika mengunduh kueri, Anda akan mendapatkan 1 MB penuh. Dalam hal ini, melihat dan mengunduh menampilkan jumlah byte yang berbeda. Untuk informasi selengkapnya tentang parameter instans `track_activity_query_size` DB, lihat Statistik Run-time dalam dokumentasi SQL Postgre.

Untuk menambah ukuran SQL teks, tingkatkan `track_activity_query_size` batasnya. Untuk memodifikasi parameter, ubah pengaturan parameter dalam grup parameter yang terkait dengan instans DB . SQL

Untuk mengubah pengaturan saat instans menggunakan grup parameter default

1. Buat grup parameter instans DB baru untuk mesin DB dan versi mesin DB yang sesuai.
2. Tetapkan parameter di grup parameter baru.
3. Hubungkan grup parameter baru dengan instans DB.

Untuk informasi tentang cara mengatur parameter instans DB, lihat Memodifikasi parameter dalam grup parameter DB di .

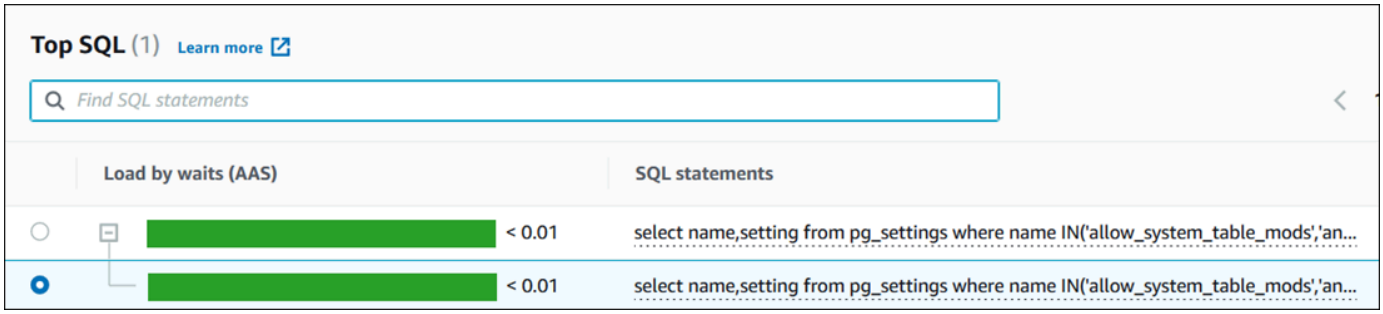
Melihat dan mengunduh SQL teks di dasbor Performance Insights

Di dasbor Performance Insights, Anda dapat melihat atau mengunduh SQL teks.

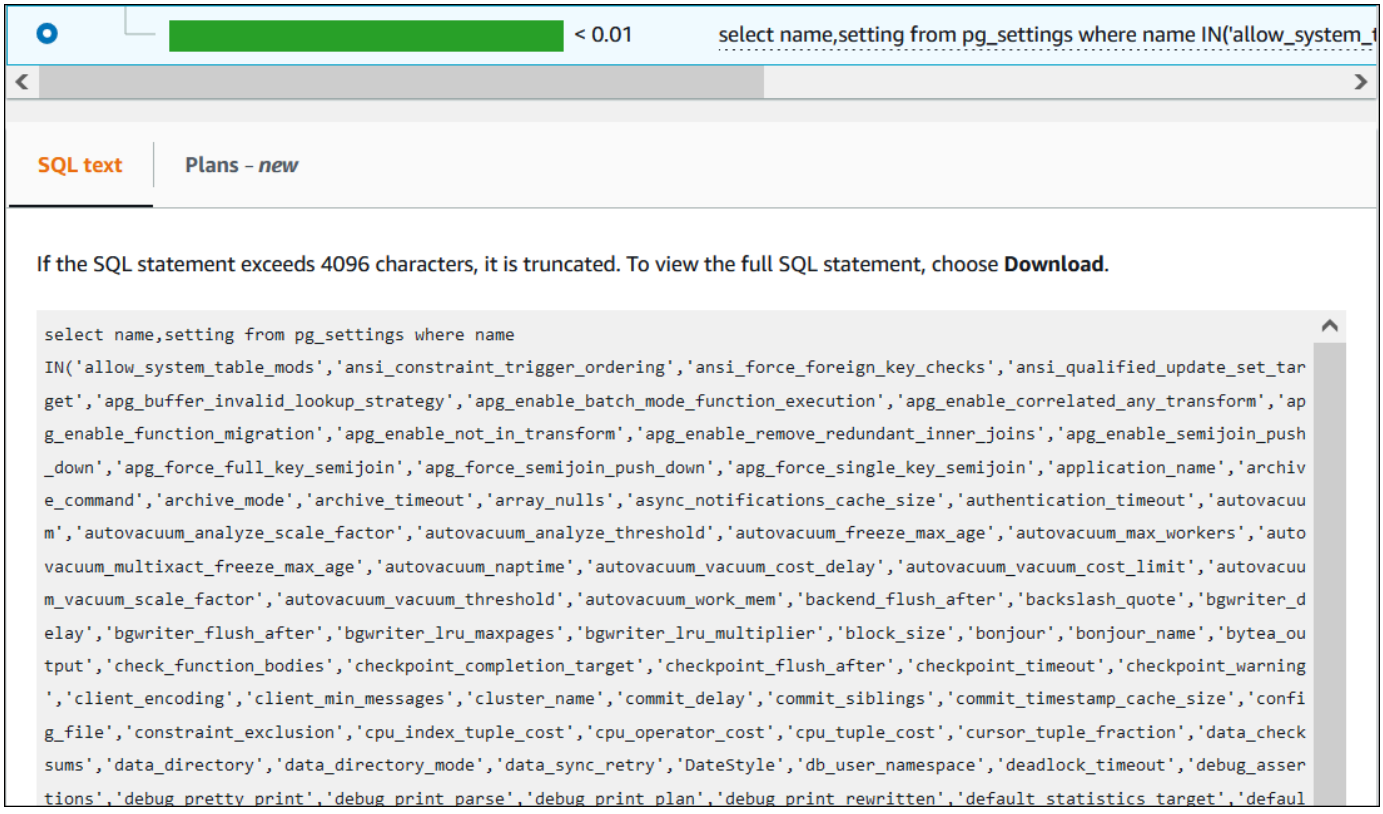
Untuk melihat SQL teks lainnya di dasbor Performance Insights

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB.
4. Gulir ke bawah ke SQL tab Top di dasbor Performance Insights.
5. Pilih tanda plus untuk memperluas SQL intisari dan pilih salah satu kueri turunan intisari.

   SQLpernyataan dengan teks yang lebih besar dari 500 byte terlihat mirip dengan gambar berikut.




6. Gulir ke bawah ke tab SQLteks.



Dasbor Performance Insights dapat menampilkan hingga 4.096 byte untuk setiap pernyataan. SQL

7. (Opsional) Pilih Salin untuk menyalin SQL pernyataan yang ditampilkan, atau pilih Unduh untuk mengunduh SQL pernyataan untuk melihat SQL teks hingga batas mesin DB.

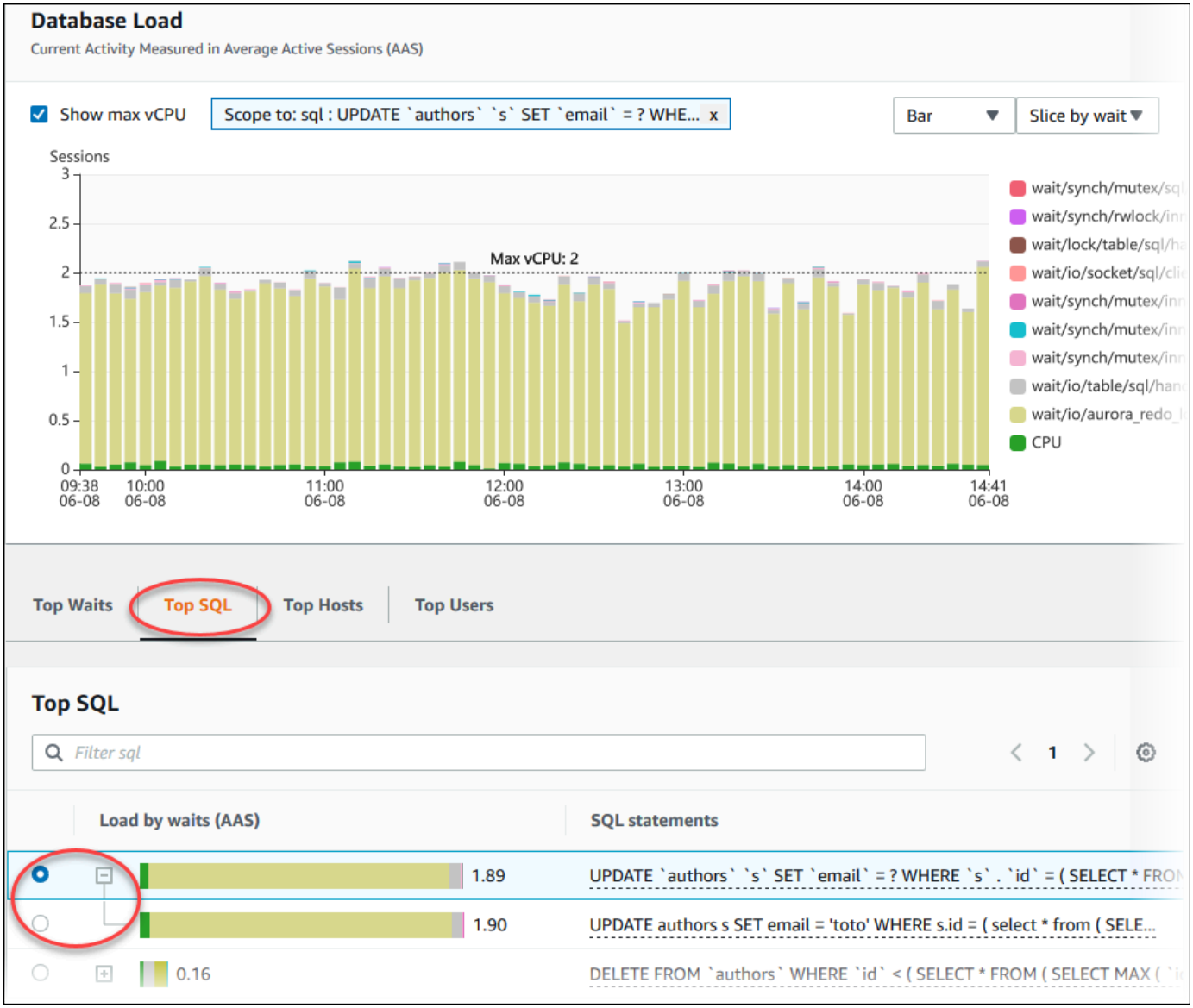
> **Note**
>
> Untuk menyalin atau mengunduh SQL pernyataan, nonaktifkan pemblokir pop-up.

### Melihat SQL statistik di dasbor Performance Insights

Di dasbor Performance Insights, SQL statistik tersedia di SQL tab Atas bagan pemuatan Database.

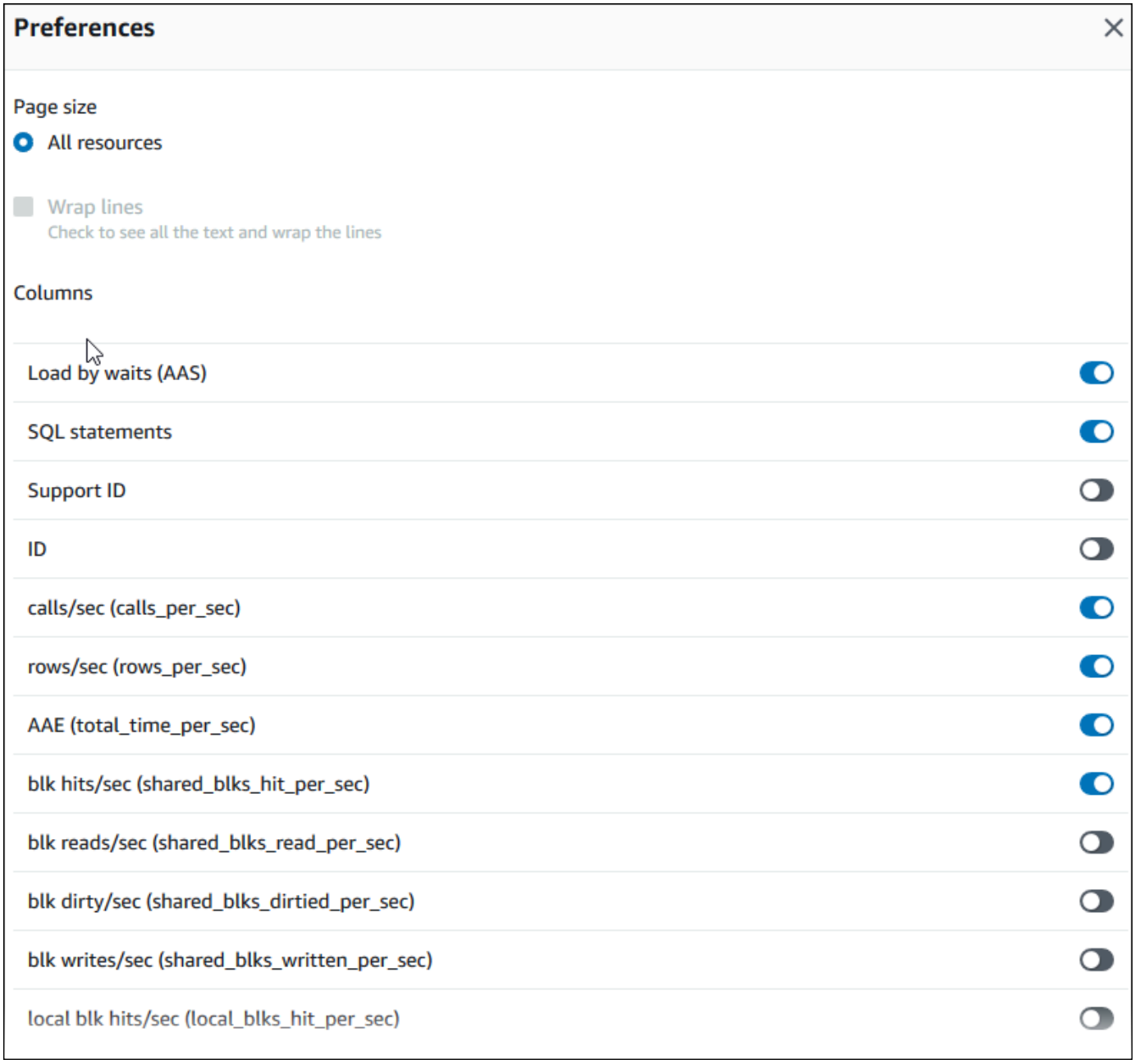
Untuk melihat SQL statistik

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Di bagian atas halaman, pilih database yang SQL statistiknya ingin Anda lihat.
4. Gulir ke bagian bawah halaman dan pilih SQL tab Atas.
5. Pilih pernyataan individual (Aurora My SQL only) atau kueri intisari.

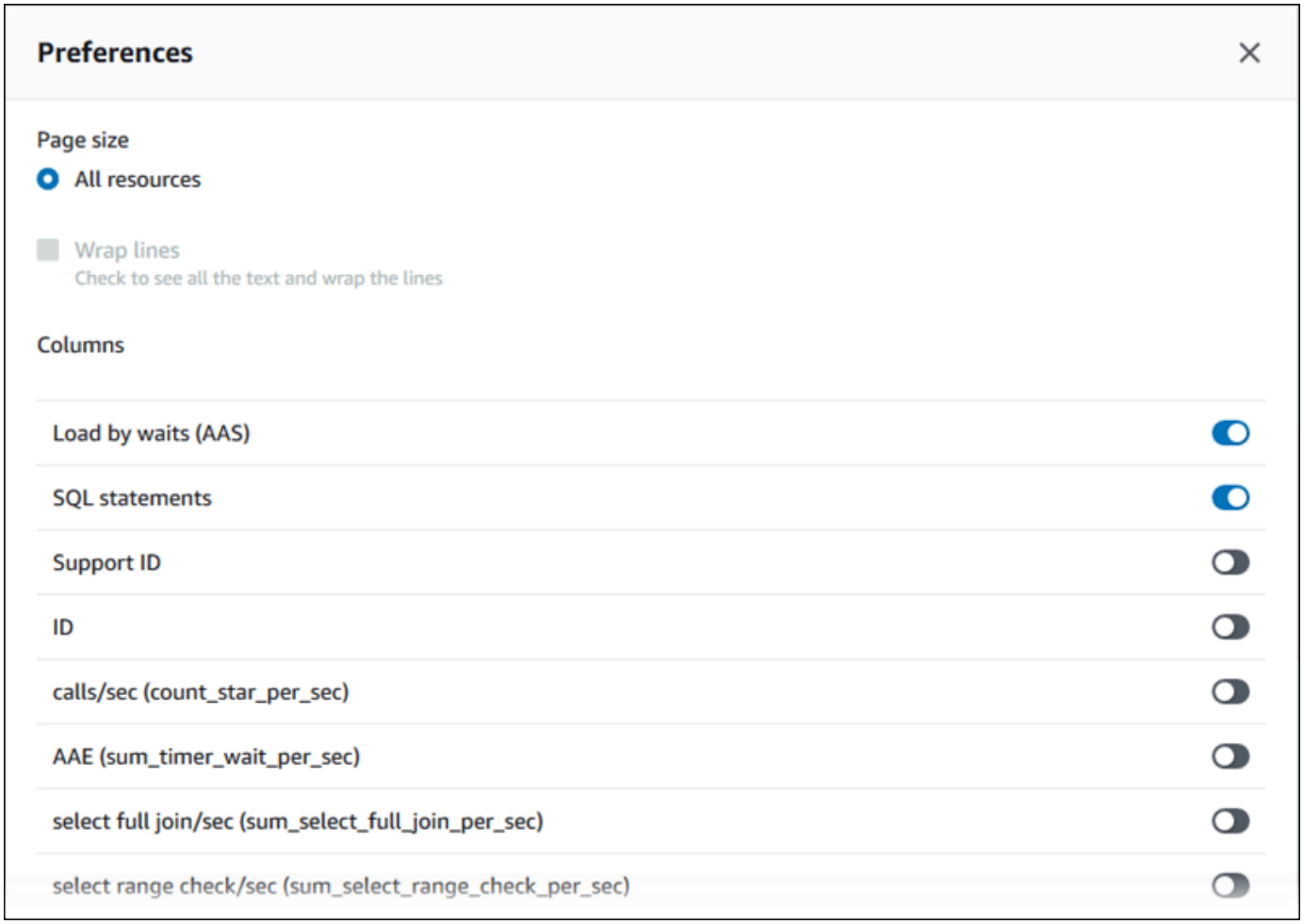



6. Pilih statistik yang akan ditampilkan dengan memilih ikon roda gigi di sudut kanan atas bagan. Untuk deskripsi SQL statistik untuk mesin Amazon RDS Aurora, lihat. Statistik SQL untuk Wawasan Performa

   Contoh berikut menunjukkan preferensi untuk Aurora Postgre. SQL




Contoh berikut menunjukkan preferensi untuk instance DB. SQL




7. Pilih Simpan untuk menyimpan preferensi Anda.

   SQLMeja atas menyegarkan.

## Melihat rekomendasi proaktif Performance Insights

Amazon RDS Performance Insights memantau metrik tertentu dan secara otomatis membuat ambang batas dengan menganalisis level apa yang mungkin berpotensi bermasalah untuk sumber daya tertentu. Ketika nilai metrik baru melewati ambang batas yang telah ditentukan selama periode waktu tertentu, Performance Insights menghasilkan rekomendasi proaktif. Rekomendasi ini membantu mencegah dampak kinerja database future. Untuk menerima rekomendasi proaktif ini, Anda harus mengaktifkan Performance Insights dengan periode retensi tingkat berbayar.

Untuk informasi selengkapnya tentang mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora. Untuk informasi tentang harga dan retensi data untuk Performance Insights, lihat. Harga dan retensi data untuk Wawasan Performa

Untuk mengetahui wilayah, mesin DB, dan kelas instance yang didukung untuk rekomendasi proaktif, lihatDukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa.

Anda dapat melihat analisis terperinci dan investigasi rekomendasi proaktif yang direkomendasikan di halaman detail rekomendasi.

Untuk informasi lebih lanjut tentang rekomendasi, lihat.

Untuk melihat analisis rinci dari rekomendasi proaktif

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi.

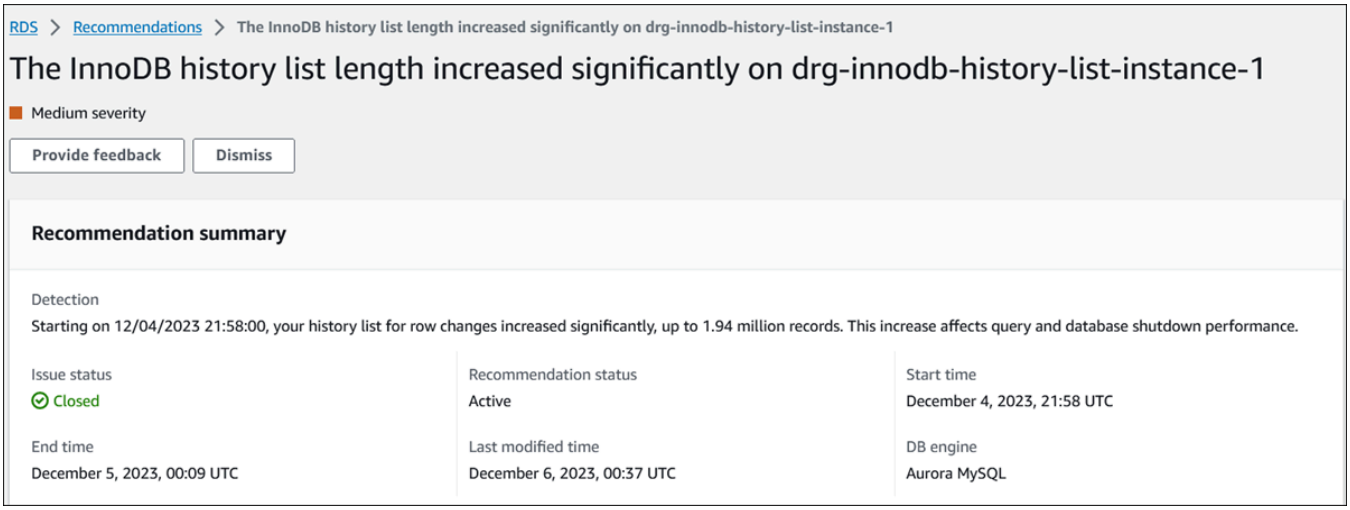
     Halaman Rekomendasi menampilkan daftar rekomendasi yang diurutkan berdasarkan tingkat keparahan semua sumber daya di akun Anda.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Tab Rekomendasi menampilkan rekomendasi dan detailnya untuk sumber daya yang dipilih.
3. Temukan rekomendasi proaktif dan pilih Lihat detail.

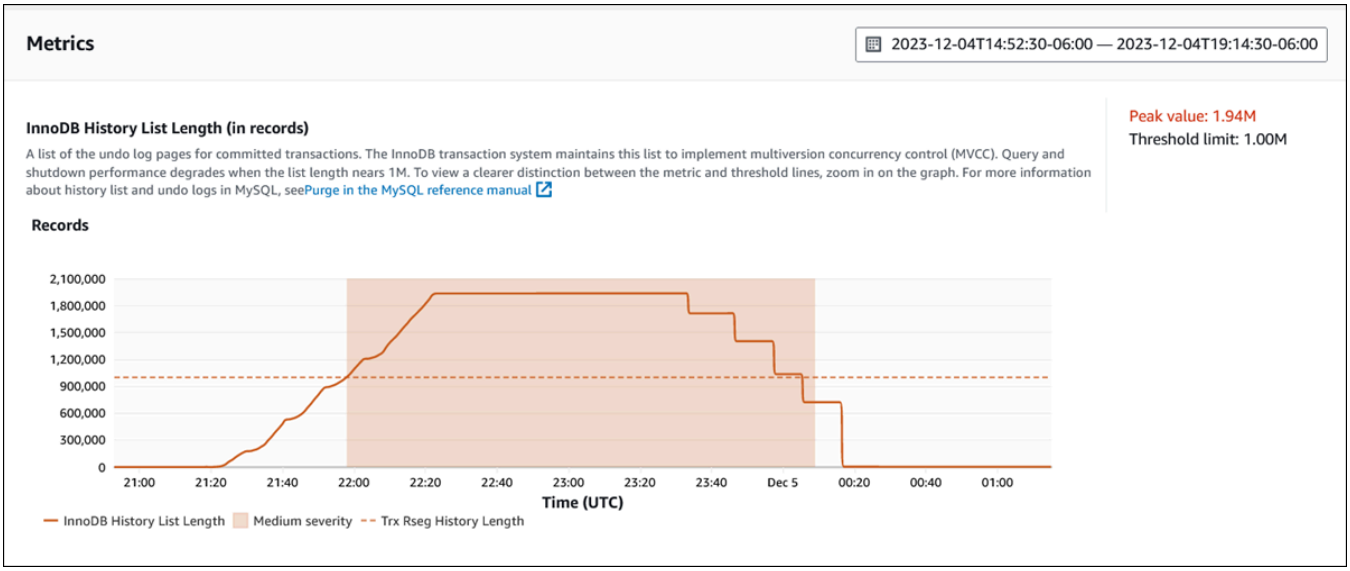
   Halaman detail rekomendasi muncul. Judul memberikan nama sumber daya yang terpengaruh dengan masalah yang terdeteksi dan tingkat keparahannya.

   Berikut ini adalah komponen pada halaman detail rekomendasi:

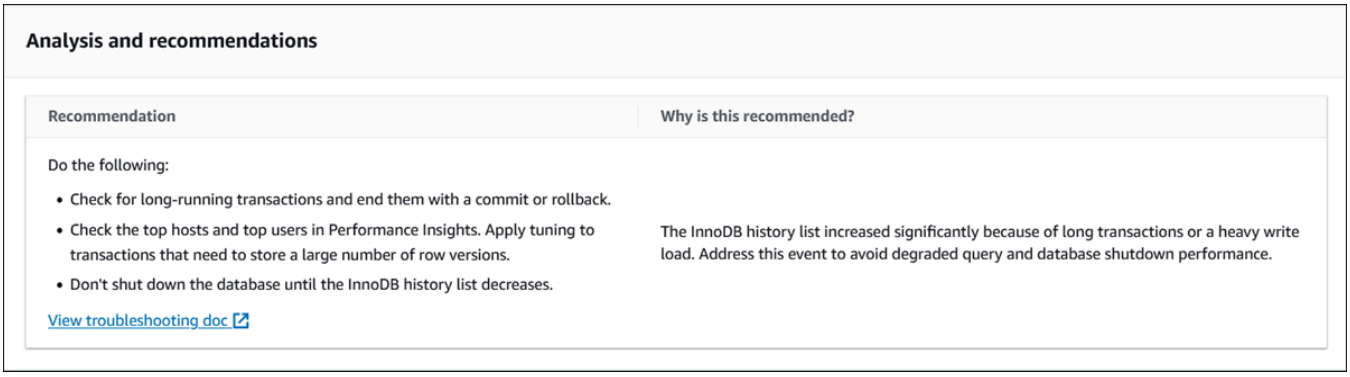
   - Ringkasan rekomendasi — Masalah yang terdeteksi, status rekomendasi dan masalah, waktu mulai dan berakhir masalah, waktu modifikasi rekomendasi, dan jenis mesin.




- Metrik — Grafik masalah yang terdeteksi. Setiap grafik menampilkan ambang batas yang ditentukan oleh perilaku dasar sumber daya dan data metrik yang dilaporkan dari waktu mulai masalah.




- Analisis dan rekomendasi — Rekomendasi dan alasan rekomendasi yang disarankan.



Anda dapat meninjau penyebab masalah dan kemudian melakukan tindakan yang disarankan untuk memperbaiki masalah, atau memilih Singkirkan di kanan atas untuk mengabaikan rekomendasi.

## Mengambil metrik dengan Performance API Insights untuk Aurora

Saat Performance Insights diaktifkan, maka akan API memberikan visibilitas ke performa instans. Amazon CloudWatch Logs menyediakan sumber otoritatif untuk metrik pemantauan terjual untuk layanan. AWS

Performance Insights menawarkan tampilan khusus domain dari beban database yang diukur sebagai sesi aktif rata-rata (). AAS Metrik ini muncul bagi API konsumen sebagai kumpulan data deret waktu dua dimensi. Dimensi waktu data menyediakan data muatan DB untuk setiap titik waktu dalam rentang waktu yang dikueri. Setiap titik waktu menguraikan keseluruhan muatan dalam hubungannya dengan dimensi yang diminta, seperti `SQL`, `Wait-event`, `User`, atau `Host`, yang diukur pada titik waktu tersebut.

Amazon RDS Performance Insights memantau cluster Amazon Aurora sehingga Anda dapat menganalisis dan memecahkan masalah kinerja database. Salah satu cara untuk menampilkan data Wawasan Performa dapat ditemukan di AWS Management Console. Performance Insights juga menyediakan publik API sehingga Anda dapat melakukan kueri data Anda sendiri. Anda dapat menggunakan API untuk melakukan hal berikut:

- Membongkar data ke dalam basis data
- Menambahkan data Wawasan Performa ke dasbor pemantauan yang ada
- Merancang alat pemantauan

Untuk menggunakan Performance InsightsAPI, aktifkan Performance Insights di salah satu instans Amazon RDS DB Anda. Untuk informasi tentang cara mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora. Untuk informasi selengkapnya tentang Performance InsightsAPI, lihat Referensi Performance RDS API Insights Amazon.

Performance Insights API menyediakan operasi berikut.

| Tindakan Wawasan Performa | AWS CLI perintah | Deskripsi |
| --- | --- | --- |
| CreatePerformanceAnalysisReport | aws pi create-performance-analysis-report | Membuat laporan analisis performa selama periode waktu tertentu untuk instans DB. Hasilnya adalah `AnalysisReportId` , yakni ID unik laporan. |
| DeletePerformanceAnalysisReport | aws pi delete-performance-analysis-report | Menghapus laporan analisis performa. |
| DescribeDimensionKeys | aws pi describe-dimension-keys | Mengambil kunci dimensi N teratas untuk metrik selama periode waktu tertentu. |
| GetDimensionKeyDetails | aws pi get-dimension-key-details | Mengambil atribut dari grup dimensi tertentu untuk instans DB atau sumber data. Misalnya, jika Anda menentukan SQL ID, dan jika detail dimensi tersedia, `GetDimensionKeyDetails` mengambil teks lengkap dari dimensi yang `db.sql.statement` terkait dengan ID ini. Operasi ini berguna karena `GetResourceMetrics` dan `DescribeDimensionKeys` tidak mendukung pengambilan teks SQL pernyataan besar. |

| Tindakan Wawasan Performa | AWS CLI perintah | Deskripsi |
| --- | --- | --- |
| GetPerformanceAnalysisReport | aws pi get-performance-analysis-report | Mengambil laporan yang mencakup wawasan untuk laporan. Hasilnya mencakup status laporan, ID laporan, detail waktu laporan, wawasan, dan rekomendasi. |
| GetResourceMetadata | aws pi get-resource-metadata | Mengambil metadata untuk fitur yang berbeda. Misalnya, metadata mungkin menunjukkan bahwa fitur diaktifkan atau dinonaktifkan pada instans DB tertentu. |
| GetResourceMetrics | aws pi get-resource-metrics | Mengambil metrik Wawasan Performa untuk sekumpulan sumber data, selama periode waktu tertentu. Anda dapat menyediakan grup dimensi dan dimensi tertentu, serta memberikan kriteria penggabungan dan pemfilteran untuk setiap grup. |
| ListAvailableResourceDimensions | aws pi list-available-resource-dimensions | Mengambil dimensi yang dapat dikueri untuk setiap jenis metrik yang ditentukan pada instans tertentu. |
| ListAvailableResourceMetrics | aws pi list-available-resource-metrics | Mengambil semua metrik jenis metrik tertentu yang tersedia yang dapat dikueri untuk instans DB tertentu. |

| Tindakan Wawasan Performa | AWS CLI perintah | Deskripsi |
| --- | --- | --- |
| ListPerformanceAnalysisReports | aws pi list-performance-analysis-reports | Mengambil semua laporan analisis yang tersedia untuk instans DB. Laporan dicantumkan berdasarkan waktu mulai setiap laporan. |
| ListTagsForResource | aws pi list-tags-for-resource | Daftar semua tag metadata yang ditambahkan ke sumber daya. Daftar ini mencakup nama dan nilai tag. |
| TagResource | aws pi tag-resource | Menambahkan tag metadata ke sumber daya AmazonRDS . Tag ini termasuk nama dan nilai. |
| UntagResource | aws pi untag-resource | Menghapus tag metadata dari sumber daya. |

Untuk informasi selengkapnya tentang mengambil metrik deret waktu dan AWS CLI contoh untuk Performance Insights, lihat topik berikut.

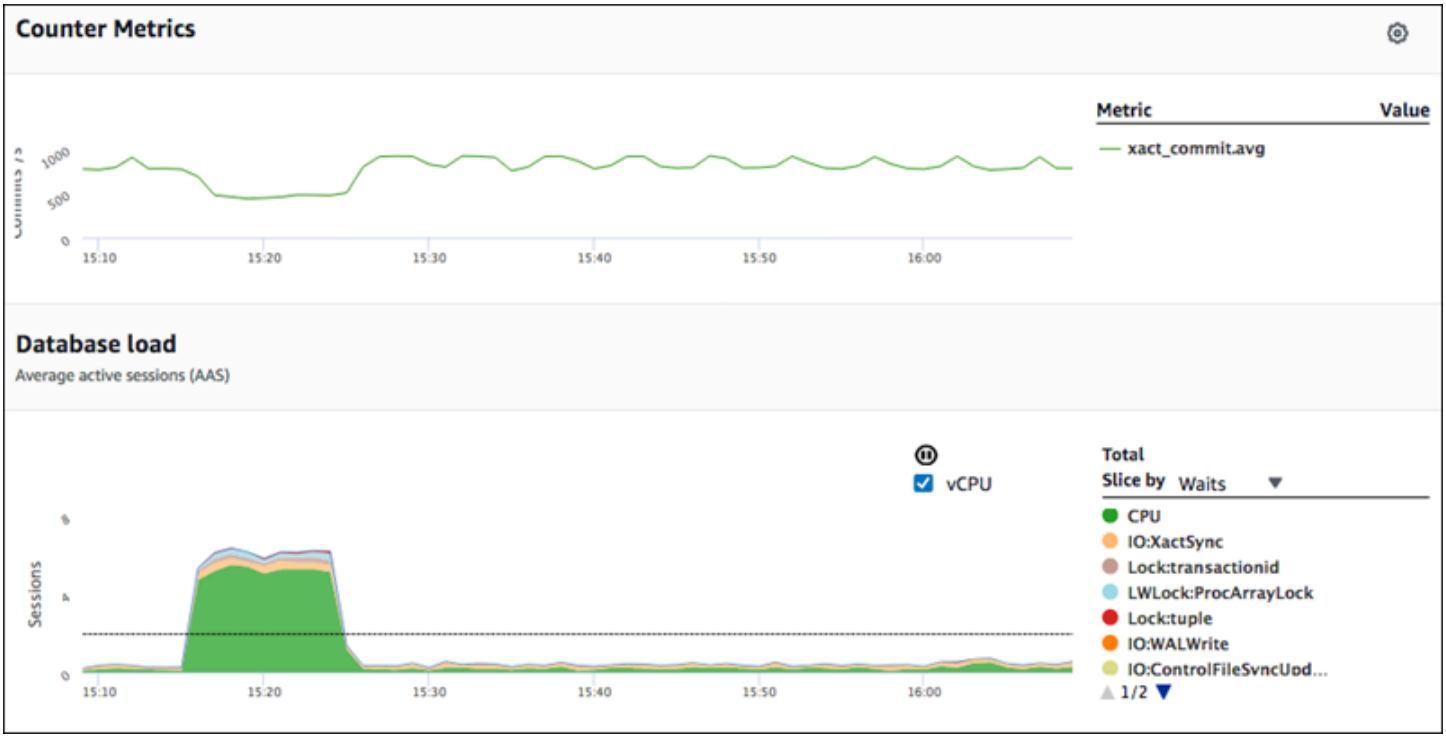
Topik

- Mengambil metrik deret waktu untuk Performance Insights
- AWS CLI contoh untuk Performance Insights

## Mengambil metrik deret waktu untuk Performance Insights

Operasi `GetResourceMetrics` mengambil satu atau beberapa metrik deret waktu dari data Wawasan Performa. `GetResourceMetrics` memerlukan metrik dan periode waktu, dan menampilkan respons dengan daftar poin data.

Misalnya, AWS Management Console penggunaan `GetResourceMetrics` untuk mengisi bagan Metrik Penghitung dan bagan Pemuatan Database, seperti yang terlihat pada gambar berikut.




Semua metrik yang ditampilkan oleh `GetResourceMetrics` adalah metrik deret waktu standar, dengan pengecualian `db.load`. Metrik ini ditampilkan dalam bagan Muatan Basis Data. Metrik `db.load` berbeda dengan metrik deret waktu lainnya karena Anda dapat membaginya menjadi beberapa sub-komponen yang disebut dimensi. Di gambar sebelumnya, `db.load` dibagi dan dikelompokkan berdasarkan status tunggu yang membentuk `db.load`.

> **Note**
>
> `GetResourceMetrics` juga dapat menampilkan metrik `db.sampleload`, tetapi metrik `db.load` sesuai di sebagian besar kasus.

Untuk informasi tentang metrik penghitung yang ditampilkan oleh `GetResourceMetrics`, lihat Metrik penghitung Wawasan Performa.

Penghitungan berikut didukung untuk metrik:

- Rata-rata – Nilai rata-rata untuk metrik selama periode waktu tertentu. Tambahkan `.avg` ke nama metrik.
- Minimum – Nilai minimum metrik selama periode waktu tertentu. Tambahkan `.min` ke nama metrik.

- Maksimum – Nilai maksimum metrik selama periode waktu tertentu. Tambahkan `.max` ke nama metrik.
- Jumlah – Jumlah nilai metrik selama periode waktu tertentu. Tambahkan `.sum` ke nama metrik.
- Jumlah sampel – Frekuensi pengumpulan metrik selama periode waktu tertentu. Tambahkan `.sample_count` ke nama metrik.

Sebagai contoh, misalkan sebuah metrik dikumpulkan selama 300 detik (5 menit), dan metrik tersebut dikumpulkan satu kali setiap menit. Nilai untuk setiap menit adalah 1, 2, 3, 4, dan 5. Dalam kasus ini, penghitungan berikut ditampilkan:

- Rata-rata – 3
- Minimum – 1
- Maksimum – 5
- Jumlah – 15
- Jumlah sampel – 5

Untuk informasi tentang menggunakan `get-resource-metrics` AWS CLI perintah, lihat `get-resource-metrics`.

Untuk opsi `--metric-queries`, tentukan satu atau beberapa kueri yang diinginkan untuk mendapatkan hasil. Setiap kueri terdiri dari `Metric` wajib dan `GroupBy` opsional serta parameter `Filter`. Berikut ini adalah contoh spesifikasi opsi `--metric-queries`.

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

## AWS CLI contoh untuk Performance Insights

Di bagian berikut, pelajari lebih lanjut tentang AWS Command Line Interface (AWS CLI) untuk Performance Insights dan gunakan AWS CLI contoh.

Topik


- Bantuan bawaan AWS CLI untuk Performance Insights
- Mengambil metrik penghitung
- Mengambil rata-rata muatan DB untuk peristiwa tunggu teratas
- Mengambil rata-rata beban DB untuk bagian atas SQL
- Mengambil rata-rata beban DB yang disaring oleh SQL
- Mengambil teks lengkap dari sebuah pernyataan SQL
- Membuat laporan analisis performa selama periode waktu tertentu
- Mengambil laporan analisis performa
- Daftar semua laporan analisis performa untuk instans DB
- Menghapus laporan analisis performa
- Menambahkan tag ke laporan analisis performa
- Mencantumkan semua tag untuk laporan analisis performa
- Menghapus tag dari laporan analisis performa


### Bantuan bawaan AWS CLI untuk Performance Insights

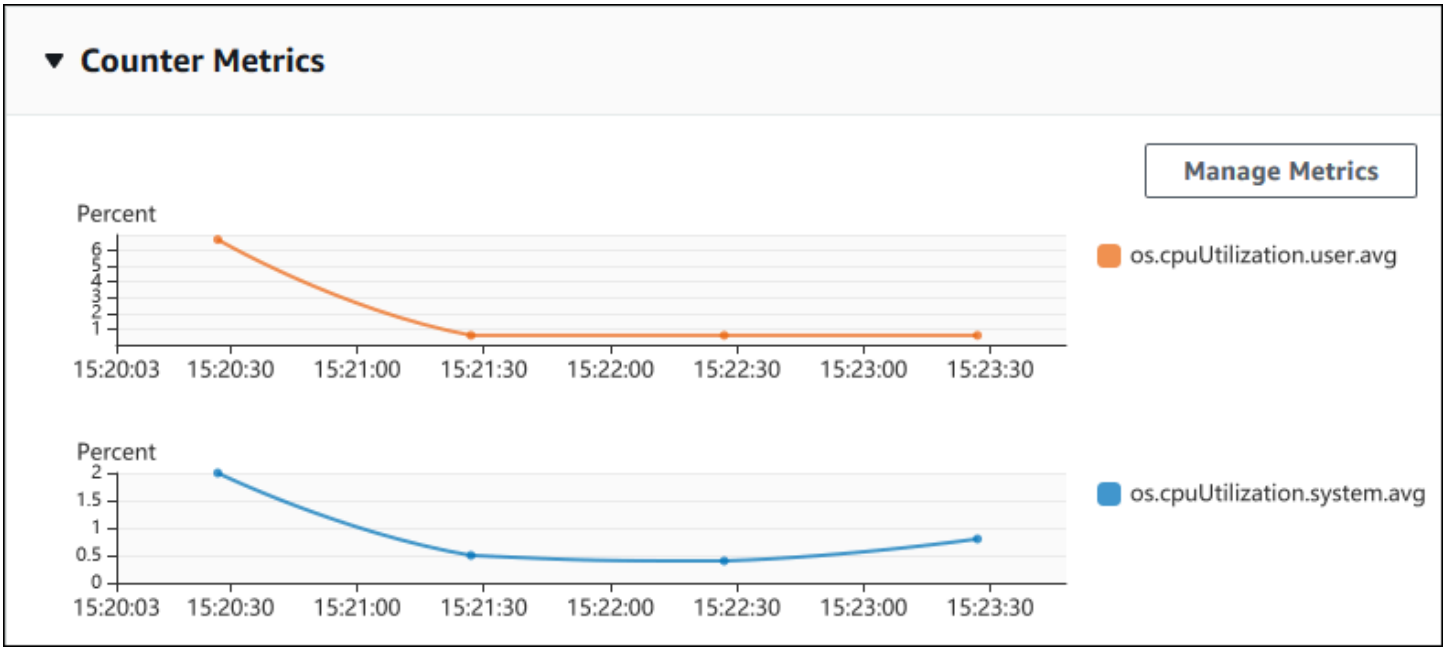
Anda dapat melihat data Wawasan Performa menggunakan AWS CLI. Anda dapat melihat bantuan untuk AWS CLI perintah untuk Performance Insights dengan memasukkan yang berikut ini pada baris perintah.

```
aws pi help
```

Jika Anda belum menginstal, lihat Menginstal AWS CLI di Panduan AWS CLI Pengguna untuk informasi tentang menginstalnya. AWS CLI

### Mengambil metrik penghitung

Tangkapan layar berikut menunjukkan dua bagan metrik penghitung dalam AWS Management Console.




Contoh berikut menunjukkan cara mengumpulkan data yang sama yang AWS Management Console digunakan untuk menghasilkan dua bagan metrik penghitung.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Anda juga dapat membaca perintah dengan lebih mudah dengan menentukan file untuk opsi `--metrics-query`. Contoh berikut menggunakan file yang disebut query.json untuk opsi tersebut. File memiliki konten berikut.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

Jalankan perintah berikut untuk menggunakan file.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Contoh sebelumnya menentukan nilai-nilai berikut untuk opsi:

- `--service-type`— RDS untuk Amazon RDS
- `--identifier` – ID sumber daya untuk instans DB
- `--start-time`dan `--end-time` — `DateTime` Nilai ISO 8601 untuk periode kueri, dengan beberapa format yang didukung

Ini akan dikueri selama rentang waktu satu jam:

- `--period-in-seconds` – `60` untuk kueri per menit
- `--metric-queries` – Rangkaian dua kueri, masing-masing hanya untuk satu metrik.

  Nama metrik menggunakan titik untuk mengklasifikasikan metrik dalam kategori yang berguna, dengan elemen terakhir sebagai fungsi. Dalam contoh, fungsinya adalah `avg` untuk setiap kueri. Seperti halnya Amazon CloudWatch, fungsi yang didukung adalah `min`,`max`,`total`, dan`avg`.

Responsnya terlihat seperti berikut.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
```

```
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

Respons ini memiliki `Identifier`, `AlignedStartTime`, dan `AlignedEndTime`. B nilai `--period-in-seconds` adalah `60`, waktu mulai dan akhir telah disesuaikan dengan menit. Jika `--period-in-seconds` adalah `3600`, waktu mulai dan akhir akan disesuaikan dengan jam.

`MetricList` dalam respons memiliki sejumlah entri, masing-masing dengan entri `Key` dan `DataPoints`. Masing-masing `DataPoint` memiliki `Timestamp` dan `Value`. Masing-masing daftar `Datapoints` memiliki 60 poin data karena kueri tersebut adalah untuk data per menit selama satu jam, dengan `Timestamp1/Minute1`, `Timestamp2/Minute2`, dan seterusnya, hingga `Timestamp60/Minute60`.

Karena kueri tersebut adalah untuk dua metrik penghitung yang berbeda, ada dua elemen dalam respons `MetricList`.

Mengambil rata-rata muatan DB untuk peristiwa tunggu teratas

Contoh berikut adalah query yang sama yang AWS Management Console digunakan untuk menghasilkan grafik garis area bertumpuk. Contoh ini mengambil `db.load.avg` selama satu jam terakhir dengan muatan yang dibagi berdasarkan tujuh peristiwa tunggu teratas. Perintah ini sama dengan perintah dalam Mengambil metrik penghitung. Namun, file query.json berisi konten berikut.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

Jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Contoh ini menentukan metrik `db.load.avg` dan `GroupBy` dari tujuh peristiwa tunggu teratas. Untuk detail tentang nilai yang valid untuk contoh ini, lihat DimensionGroupdi Referensi Performance Insights API.

Responsnya terlihat seperti berikut.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
```

```
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

Dalam respons ini, ada delapan entri dalam `MetricList`. Ada satu entri untuk total `db.load.avg`, dan tujuh entri masing-masing untuk `db.load.avg` yang dibagi berdasarkan salah satu dari tujuh peristiwa tunggu teratas. Tidak seperti di contoh pertama, karena ada dimensi pengelompokan, pasti ada satu kunci untuk setiap pengelompokan metrik. Tidak boleh hanya ada satu kunci untuk setiap metrik, seperti dalam kasus penggunaan metrik penghitung dasar.

Mengambil rata-rata beban DB untuk bagian atas SQL

Contoh berikut dikelompokkan `db.wait_events` oleh 10 SQL pernyataan teratas. Ada dua kelompok berbeda untuk SQL pernyataan:

- `db.sql`SQLPernyataan lengkap, seperti `select * from customers where customer_id = 123`
- `db.sql_tokenized`— SQL Pernyataan tokenized, seperti `select * from customers where customer_id = ?`

Saat menganalisis kinerja database, akan berguna untuk mempertimbangkan SQL pernyataan yang hanya berbeda dengan parameternya sebagai satu item logika. Jadi, Anda dapat menggunakan `db.sql_tokenized` saat melakukan kueri. Namun, terutama ketika Anda tertarik untuk menjelaskan rencana, terkadang lebih berguna untuk memeriksa SQL pernyataan lengkap dengan parameter, dan pengelompokan kueri berdasarkan. `db.sql` Ada hubungan orang tua-anak antara tokenized dan fullSQL, dengan beberapa penuh SQL (anak) dikelompokkan di bawah token yang sama (induk). SQL

Perintah dalam contoh ini terlihat seperti perintah dalam Mengambil rata-rata muatan DB untuk peristiwa tunggu teratas. Namun, file query.json berisi konten berikut.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

Contoh berikut menggunakan `db.sql_tokenized`.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
   --end-time   2018-10-30T00:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

Contoh ini menanyakan lebih dari 24 jam, dengan satu jam period-in-seconds.

Contoh ini menentukan metrik `db.load.avg` dan `GroupBy` dari tujuh peristiwa tunggu teratas. Untuk detail tentang nilai yang valid untuk contoh ini, lihat DimensionGroupdi Referensi Performance Insights API.

Responsnya terlihat seperti berikut.

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
```

```
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

Respons ini memiliki 11 entri dalam `MetricList` (total 1, 10 tokenized teratasSQL), dengan setiap entri memiliki 24 per jam. `DataPoints`

Untuk tokenizedSQL, ada tiga entri di setiap daftar dimensi:

- `db.sql_tokenized.statement`— Pernyataan tokenizedSQL.
- `db.sql_tokenized.db_id` — Baik ID database asli yang digunakan untuk merujuk keSQL, atau ID sintetis yang dihasilkan Performance Insights untuk Anda jika ID database asli tidak tersedia. Contoh ini menampilkan ID sintetis `pi-2372568224`.
- `db.sql_tokenized.id` – ID kueri di dalam Wawasan Performa.

  Dalam AWS Management Console, ID ini disebut Support ID. Ini dinamakan ini karena ID adalah data yang AWS Support dapat memeriksa untuk membantu Anda memecahkan masalah dengan database Anda. AWS menjaga keamanan dan privasi data Anda dengan sangat serius, dan hampir semua data disimpan terenkripsi dengan kunci Anda AWS KMS . Oleh karena itu, tidak ada orang di dalam yang AWS dapat melihat data ini. Di contoh sebelumnya, baik `tokenized.statement` maupun `tokenized.db_id` disimpan dengan enkripsi. Jika Anda memiliki masalah dengan database Anda, AWS Support dapat membantu Anda dengan mereferensikan ID Dukungan.

Ketika melakukan kueri, mungkin lebih mudah untuk menentukan `Group` dalam `GroupBy`. Namun, untuk kontrol lebih mendetail atas data yang ditampilkan, tentukan daftar dimensi. Misalnya, jika yang dibutuhkan hanya `db.sql_tokenized.statement`, atribut `Dimensions` dapat ditambahkan ke file query.json.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
```

```
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

Mengambil rata-rata beban DB yang disaring oleh SQL

Gambar sebelumnya menunjukkan bahwa kueri tertentu dipilih, dan grafik baris area bertumpuk sesi aktif rata-rata teratas dicakup ke kueri tersebut. Meskipun kueri masih diperuntukkan bagi tujuh peristiwa tunggu teratas secara keseluruhan, nilai respons-nya akan difilter. Filter menyebabkannya hanya memperhitungkan sesi yang cocok untuk filter tertentu.

APIKueri yang sesuai dalam contoh ini mirip dengan perintah di[Mengambil rata-rata beban DB untuk bagian atas SQL](). Namun, file query.json berisi konten berikut.

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
```

```
    }
]
```

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Responsnya terlihat seperti berikut.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
```

```
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
```

```
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
```

```
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

Dalam tanggapan ini, semua nilai difilter sesuai dengan kontribusi token yang SQL AKIAIOSFODNN7EXAMPLE ditentukan dalam file query.json. Kunci juga mungkin mengikuti urutan yang berbeda dari kueri tanpa filter, karena itu adalah lima peristiwa tunggu teratas yang memengaruhi filterSQL.

Mengambil teks lengkap dari sebuah pernyataan SQL

Contoh berikut mengambil teks lengkap SQL pernyataan untuk contoh `db-10BCD2EFGHIJ3KL4M5NO6PQRS5` DB. `--group` adalah `db.sql`, dan `--group-identifier` adalah `db.sql.id`. Dalam contoh ini, *`my-sql-id`* mewakili SQL ID diambil dengan memanggil `pi get-resource-metrics` atau. `pi describe-dimension-keys`

Jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

Untuk Windows:

```
aws pi get-dimension-key-details ^
   --service-type RDS ^
```

```
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
   --group db.sql ^
   --group-identifier my-sql-id ^
   --requested-dimensions statement
```

Dalam contoh ini, detail dimensinya tersedia. Dengan demikian, Performance Insights mengambil teks lengkap SQL pernyataan, tanpa memotongnya.

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

Membuat laporan analisis performa selama periode waktu tertentu

Contoh berikut membuat laporan analisis performa dengan waktu mulai `1682969503` dan waktu akhir `1682979503` untuk basis data `db-loadtest-0`.

```
aws pi create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --region us-west-2
```

Responsnya adalah ID unik `report-0234d3ed98e28fb17` untuk laporan tersebut.

```
{
   "AnalysisReportId": "report-0234d3ed98e28fb17"
}
```

Mengambil laporan analisis performa

Contoh berikut mengambil detail laporan analisis untuk laporan `report-0d99cc91c4422ee61`.

```
aws pi get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

Respons memberikan status laporan, ID, detail waktu, dan wawasan.

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

Daftar semua laporan analisis performa untuk instans DB

Contoh berikut mencantumkan semua laporan analisis performa yang tersedia untuk basis data `db-loadtest-0`.

```
aws pi list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--region us-west-2
```

Respons ini mencantumkan semua laporan dengan ID laporan, status, dan detail periode waktu.

```
{
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
```

```
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

Menghapus laporan analisis performa

Contoh berikut menghapus laporan analisis untuk basis data `db-loadtest-0`.

```
aws pi delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

Menambahkan tag ke laporan analisis performa

Contoh berikut menambahkan tag dengan kunci `name` dan nilai `test-tag` ke laporan `report-01ad15f9b88bcbd56`.

```
aws pi tag-resource \
```

```
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
--region us-west-2
```

Mencantumkan semua tag untuk laporan analisis performa

Contoh berikut mencantumkan semua tag untuk laporan `report-01ad15f9b88bcbd56`.

```
aws pi list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--region us-west-2
```

Respons ini mencantumkan nilai dan kunci untuk semua tag yang ditambahkan ke laporan:

```
{
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

Menghapus tag dari laporan analisis performa

Contoh berikut menghapus tag `name` dari laporan `report-01ad15f9b88bcbd56`.

```
aws pi untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--region us-west-2
```

Setelah tag dihapus, memanggil `list-tags-for-resource` API tidak mencantumkan tag ini.

# Pencatatan panggilan Performance Insights menggunakan AWS CloudTrail

Performance Insights berjalan dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Performance Insights. CloudTrailmenangkap semua API panggilan untuk Performance Insights sebagai acara. Tangkapan ini mencakup panggilan dari RDS konsol Amazon dan dari panggilan kode ke operasi Performance InsightsAPI.

Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman acara secara berkelanjutan ke bucket Amazon S3, termasuk CloudTrail peristiwa untuk Performance Insights. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan data yang dikumpulkan oleh CloudTrail, Anda dapat menentukan informasi tertentu. Informasi ini mencakup permintaan yang dibuat untuk Wawasan Performa, alamat IP asal permintaan, siapa yang membuat permintaan, dan kapan permintaan tersebut dibuat. Informasi ini juga mencakup detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

## Bekerja dengan informasi Performance Insights di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Saat aktivitas terjadi di Performance Insights, aktivitas tersebut direkam dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya di CloudTrail konsol dalam riwayat Peristiwa. Anda dapat melihat, mencari, dan mengunduh acara terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat Melihat Acara dengan Riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk Performance Insights, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua AWS Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna AWS CloudTrail :

- Ikhtisar Pembuatan Jejak
- CloudTrail Layanan dan Integrasi yang Didukung
- Mengkonfigurasi SNS Pemberitahuan Amazon untuk CloudTrail

- Menerima File CloudTrail Log dari Beberapa Wilayah dan Menerima File CloudTrail Log dari Beberapa Akun

Semua operasi Performance Insights dicatat oleh CloudTrail dan didokumentasikan dalam Referensi Performance API Insights. Misalnya, panggilan ke `DescribeDimensionKeys` dan `GetResourceMetrics` operasi menghasilkan entri dalam file CloudTrail log.

Setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau IAM.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna terfederasi.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi selengkapnya, lihat CloudTrail userIdentityElemen.

## Entri file log Wawasan Performa

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa menunjukkan satu permintaan dari sumber mana pun. Setiap acara mencakup informasi tentang operasi yang diminta, tanggal dan waktu operasi, parameter permintaan, dan sebagainya. CloudTrailfile log bukanlah jejak tumpukan yang diurutkan dari API panggilan publik, sehingga tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `GetResourceMetrics` operasi.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
```

```
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


## Performance Insights API dan titik VPC akhir antarmuka ()AWS PrivateLink

Anda dapat menggunakannya AWS PrivateLink untuk membuat koneksi pribadi antara RDS Performance Insights Anda VPC dan Amazon. Anda dapat mengakses Performance Insights seolah-olah berada di dalamVPC, tanpa menggunakan gateway internet, NAT perangkat, VPN koneksi, atau AWS Direct Connect koneksi. Instans di Anda VPC tidak memerlukan alamat IP publik untuk mengakses Performance Insights.

Anda membuat koneksi pribadi ini dengan membuat titik akhir antarmuka, yang didukung oleh AWS PrivateLink. Kami membuat antarmuka jaringan endpoint di setiap subnet yang Anda aktifkan untuk titik akhir antarmuka. Ini adalah antarmuka jaringan yang dikelola pemohon yang berfungsi sebagai titik masuk untuk lalu lintas yang ditujukan untuk Performance Insights.

Untuk informasi selengkapnya, lihat Akses Layanan AWS melalui AWS PrivateLink di AWS PrivateLink Panduan.

## Pertimbangan untuk Performance Insights

Sebelum menyiapkan titik akhir antarmuka untuk Performance Insights, tinjau Pertimbangan dalam Panduan.AWS PrivateLink

Performance Insights mendukung panggilan ke semua API tindakannya melalui titik akhir antarmuka.

Secara default, akses penuh ke Performance Insights diizinkan melalui titik akhir antarmuka. Untuk mengontrol lalu lintas ke Performance Insights melalui titik akhir antarmuka, kaitkan grup keamanan dengan antarmuka jaringan titik akhir.

## Ketersediaan

Performance Insights API saat ini mendukung VPC titik akhir Wilayah AWS yang mendukung Performance Insights. Untuk informasi tentang ketersediaan Performance Insights, lihat. Daerah yang Didukung dan engine Aurora DB untuk Performance Insights

## Membuat titik akhir antarmuka untuk Performance Insights

Anda dapat membuat titik akhir antarmuka untuk Performance Insights menggunakan konsol VPC Amazon atau AWS Command Line Interface ().AWS CLI Untuk informasi selengkapnya, lihat Membuat titik akhir antarmuka di AWS PrivateLink Panduan.

Membuat titik akhir antarmuka untuk Performance Insights menggunakan nama layanan berikut:

Jika Anda mengaktifkan private DNS untuk titik akhir antarmuka, Anda dapat membuat API permintaan ke Performance Insights menggunakan nama Regional DNS defaultnya. Misalnya, `pi.us-east-1.amazonaws.com`.

## Membuat kebijakan VPC endpoint untuk Performance Insights API

Kebijakan endpoint adalah IAM sumber daya yang dapat Anda lampirkan ke titik akhir antarmuka. Kebijakan endpoint default memungkinkan akses penuh ke Performance Insights melalui titik akhir antarmuka. Untuk mengontrol akses yang diizinkan ke Performance Insights dari AndaVPC, lampirkan kebijakan titik akhir kustom ke titik akhir antarmuka.

kebijakan titik akhir mencantumkan informasi berikut:

- Prinsipal yang dapat melakukan tindakan (Akun AWS, IAM pengguna, dan IAM peran).
- Tindakan yang dapat dilakukan.

- Sumber daya untuk melakukan tindakan.

Untuk informasi selengkapnya, lihat Mengontrol akses ke layanan menggunakan kebijakan titik akhir di Panduan AWS PrivateLink .

Contoh: kebijakan VPC endpoint untuk tindakan Performance Insights

Berikut ini adalah contoh kebijakan endpoint kustom. Saat Anda melampirkan kebijakan ini ke titik akhir antarmuka, kebijakan ini akan memberikan akses ke tindakan Performance Insights yang terdaftar untuk semua prinsipal di semua sumber daya.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreatePerformanceAnalysisReport",
            "rds:DeletePerformanceAnalysisReport",
            "rds:GetPerformanceAnalysisReport"
         ],
         "Resource":"*"
      }
   ]
}
```

Contoh: kebijakan VPC endpoint yang menolak semua akses dari akun tertentu AWS

Kebijakan VPC endpoint berikut menolak 123456789012 semua akses AWS akun ke sumber daya menggunakan endpoint. Kebijakan ini mengizinkan semua tindakan dari akun lainnya.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
```

```
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

## Pengalamatan IP untuk Performance Insights

Alamat IP memungkinkan sumber daya Anda VPC untuk berkomunikasi satu sama lain, dan dengan sumber daya melalui internet. Performance Insights mendukung protokol pengalamatan IPv4 dan IPv6 pengalamatan. Secara default, Performance Insights dan Amazon VPC menggunakan protokol IPv4 pengalamatan. Anda tidak dapat menonaktifkan perilaku ini. Saat Anda membuatVPC, pastikan untuk menentukan IPv4 CIDR blok (rentang IPv4 alamat pribadi).

Anda dapat secara opsional menetapkan IPv6 CIDR blok ke subnet VPC dan Anda, dan menetapkan IPv6 alamat dari blok itu ke RDS sumber daya di subnet Anda. Support untuk IPv6 protokol memperluas jumlah alamat IP yang didukung. Dengan menggunakan IPv6 protokol, Anda memastikan bahwa Anda memiliki alamat yang cukup tersedia untuk pertumbuhan internet di masa depan. RDSSumber daya baru dan yang sudah ada dapat digunakan IPv4 dan IPv6 alamat di dalam AndaVPC. Mengonfigurasi, mengamankan, dan menerjemahkan lalu lintas jaringan di antara dua protokol yang digunakan di berbagai bagian aplikasi dapat menimbulkan overhead operasional. Anda dapat melakukan standarisasi pada IPv6 protokol untuk RDS sumber daya Amazon untuk menyederhanakan konfigurasi jaringan Anda. Untuk informasi selengkapnya tentang titik akhir dan kuota layanan, lihat titik akhir dan kuota Amazon Relational Database Service.

# Menganalisis anomali kinerja Aurora dengan Amazon Guru untuk Amazon DevOps RDS

Amazon DevOps Guru adalah layanan operasi yang dikelola sepenuhnya yang membantu pengembang dan operator meningkatkan kinerja dan ketersediaan aplikasi mereka. DevOpsGuru menurunkan tugas yang terkait dengan mengidentifikasi masalah operasional sehingga Anda dapat dengan cepat menerapkan rekomendasi untuk meningkatkan aplikasi Anda. Untuk informasi selengkapnya, lihat Apa itu Amazon DevOps Guru? di Panduan Pengguna Amazon DevOps Guru.

DevOpsGuru mendeteksi, menganalisis, dan membuat rekomendasi untuk masalah operasional yang ada untuk semua mesin Amazon RDS DB. DevOps Fitur pemantauan ini memungkinkan DevOps Guru RDS untuk mendeteksi dan mendiagnosis kemacetan kinerja dan merekomendasikan tindakan korektif tertentu. DevOpsGuru for juga RDS dapat mendeteksi kondisi bermasalah di database Aurora Anda RDS untuk database sebelum terjadi.

Anda sekarang dapat melihat rekomendasi ini di RDS konsol. Untuk informasi selengkapnya, lihat .

> **Important**
>
> Amazon DevOps Guru tidak tersedia untuk Aurora Serverless klaster DB.

Video berikut adalah ikhtisar DevOps Guru untukRDS.

Untuk menyelam lebih dalam tentang hal ini, lihat Amazon DevOps Guru untuk RDS di bawah tenda.

Topik

- Manfaat DevOps Guru untuk RDS
- Bagaimana DevOps Guru RDS bekerja
- Menyiapkan DevOps Guru untuk RDS

## Manfaat DevOps Guru untuk RDS

Jika Anda bertanggung jawab atas SQL database Amazon Aurora RDS , Anda mungkin tidak tahu bahwa peristiwa atau regresi yang memengaruhi basis data tersebut sedang terjadi. Ketika mengetahui masalah ini, Anda mungkin tidak tahu alasannya terjadi atau apa yang harus dilakukan

terhadapnya. Daripada beralih ke administrator database (DBA) untuk bantuan atau mengandalkan alat pihak ketiga, Anda dapat mengikuti rekomendasi dari DevOps Guru untukRDS.

Anda mendapatkan keuntungan berikut dari analisis terperinci DevOps Guru untukRDS:

Diagnosis cepat

DevOpsGuru untuk RDS terus memantau dan menganalisis telemetri database. DevOpsGuru untuk RDS menggunakan teknik statistik dan pembelajaran mesin untuk menambang data ini dan mendeteksi anomali. Untuk mempelajari lebih lanjut tentang data telemetri, lihat Memantau pemuatan DB dengan Performance Insights di Amazon Aurora dan metrik Pemantauan OS dengan Pemantauan yang Ditingkatkan di Panduan Pengguna Amazon Aurora Memantau beban DB dengan Performance Insights di Amazon RDS dan Metrik Pemantauan . RDS

Resolusi cepat

Setiap anomali mengidentifikasi masalah performa dan menyarankan alur investigasi atau tindakan korektif. Misalnya, DevOps Guru for RDS mungkin menyarankan agar Anda menyelidiki peristiwa menunggu tertentu. Atau mungkin menyarankan agar Anda menyetel pengaturan kumpulan aplikasi Anda untuk membatasi jumlah koneksi basis data. Berdasarkan rekomendasi ini, Anda dapat menyelesaikan masalah performa lebih cepat daripada dengan memecahkan masalah secara manual.

Wawasan proaktif

DevOpsGuru untuk RDS menggunakan metrik dari sumber daya Anda untuk mendeteksi perilaku yang berpotensi bermasalah sebelum menjadi masalah yang lebih besar. Misalnya, fitur ini dapat mendeteksi ketika basis data Anda menggunakan makin banyak tabel sementara pada disk, yang dapat mulai memengaruhi performa. DevOpsGuru kemudian memberikan rekomendasi untuk membantu Anda mengatasi masalah sebelum menjadi masalah yang lebih besar.

Pengetahuan mendalam insinyur Amazon dan machine learning

Untuk mendeteksi masalah kinerja dan membantu Anda mengatasi kemacetan, DevOps Guru RDS mengandalkan pembelajaran mesin (ML) dan rumus matematika tingkat lanjut. Insinyur basis data Amazon berkontribusi pada pengembangan DevOps Guru untuk RDS temuan, yang merangkum bertahun-tahun mengelola ratusan ribu database. Dengan memanfaatkan pengetahuan kolektif ini, DevOps Guru for RDS dapat mengajari Anda praktik terbaik.

# Bagaimana DevOps Guru RDS bekerja

DevOpsGuru for RDS mengumpulkan data tentang database RDSAurora Postgre Anda dari Amazon Performance Insights. RDS Metrik yang paling penting adalah`DBLoad`. DevOpsGuru untuk RDS menggunakan metrik Performance Insights, menganalisisnya dengan pembelajaran mesin, dan menerbitkan wawasan ke dasbor.

Wawasan adalah kumpulan anomali terkait yang terdeteksi oleh DevOps Guru.

## Wawasan proaktif

Wawasan proaktif memberi tahu Anda tentang perilaku bermasalah sebelum menimbulkan masalah. Ini berisi anomali dengan rekomendasi dan metrik terkait untuk membantu Anda mengatasi masalah di database Amazon Aurora RDS Anda sebelum menjadi masalah yang lebih besar. Wawasan ini dipublikasikan di dasbor DevOps Guru.

Misalnya, DevOps Guru mungkin mendeteksi bahwa SQL database Aurora Postgre Anda membuat banyak tabel sementara di disk. Jika tidak ditangani, tren ini dapat menyebabkan masalah performa. Setiap wawasan proaktif mencakup rekomendasi untuk perilaku korektif dan ditautkan ke topik yang relevan di Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru atau Menyetel  dengan wawasan proaktif Amazon Guru DevOps. Untuk informasi selengkapnya, lihat Bekerja dengan wawasan di DevOps Guru di Panduan Pengguna Amazon DevOps Guru.

## Wawasan reaktif

Wawasan reaktif mengidentifikasi perilaku anomali saat terjadi. Jika DevOps Guru for RDS menemukan masalah kinerja di Amazon Aurora RDS Anda, Guru akan menerbitkan wawasan reaktif di dasbor Guru. DevOps Untuk informasi selengkapnya, lihat Bekerja dengan wawasan di DevOps Guru di Panduan Pengguna Amazon DevOps Guru.

Anomali kausal

Anomali kausal adalah anomali tingkat puncak dalam wawasan reaktif. Beban basis data (beban DB) adalah anomali kausal untuk Guru untuk DevOps. RDS

Anomali mengukur dampak performa dengan menetapkan tingkat keparahan Tinggi, Sedang, atau Rendah. Untuk mempelajari lebih lanjut, lihat Konsep kunci untuk DevOps Guru RDS di Panduan Pengguna Amazon DevOps Guru.

Jika DevOps Guru mendeteksi anomali saat ini pada instans DB Anda, Anda akan diberi tahu di halaman Database konsol. RDS Konsol juga memperingatkan Anda tentang anomali yang terjadi dalam 24 jam terakhir. Untuk pergi ke halaman anomali dari RDS konsol, pilih tautan dalam pesan peringatan. RDSKonsol juga memberi tahu Anda di halaman untuk DB.

Anomali kontekstual

Anomali kontekstual adalah temuan dalam Beban basis data (Beban DB) yang terkait dengan wawasan reaktif. Setiap anomali kontekstual menjelaskan Amazon Aurora spesifik yang memerlukan penyelidikan. Misalnya, DevOps Guru for RDS mungkin menyarankan Anda mempertimbangkan untuk meningkatkan CPU kapasitas atau menyelidiki peristiwa menunggu yang berkontribusi terhadap beban DB.

> **Important**
>
> Sebaiknya uji setiap perubahan pada instans uji sebelum diterapkan pada instans produksi. Dengan cara ini, Anda memahami dampak perubahan.

Untuk mempelajari lebih lanjut, lihat Menganalisis anomali RDS di Amazon di Panduan Pengguna Amazon DevOps Guru.

## Menyiapkan DevOps Guru untuk RDS

Untuk memungkinkan DevOps Guru untuk Amazon mempublikasikan wawasan RDS untuk database Amazon Aurora RDS Postgre, selesaikan tugas-tugas berikut.

Topik

- Mengkonfigurasi kebijakan IAM akses untuk DevOps Guru for RDS
- 
- Mengaktifkan DevOps Guru dan menentukan cakupan sumber daya

### Mengkonfigurasi kebijakan IAM akses untuk DevOps Guru for RDS

Untuk melihat peringatan dari DevOps Guru di RDS konsol, AWS Identity and Access Management (IAM) pengguna atau peran harus memiliki salah satu dari kebijakan berikut:

- Bagian AWS kebijakan terkelola `AmazonDevOpsGuruConsoleFullAccess`

- Bagian AWS kebijakan terkelola `AmazonDevOpsGuruConsoleReadOnlyAccess` dan salah satu dari kebijakan berikut:
  - Bagian AWS kebijakan terkelola `AmazonRDSFullAccess`
  - Kebijakan terkelola pelanggan yang mencakup `pi:GetResourceMetrics` dan `pi:DescribeDimensionKeys`

Untuk informasi selengkapnya, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa.

DevOpsGuru RDS mengandalkan Performance Insights untuk datanya. Tanpa Performance Insights, DevOps Guru menerbitkan anomali, tetapi tidak menyertakan analisis dan rekomendasi terperinci.

Saat membuat klaster basis data Aurora atau mengubah instans klaster, Anda dapat mengaktifkan Wawasan Performa. Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan Performance Insights untuk Aurora.

## Mengaktifkan DevOps Guru dan menentukan cakupan sumber daya

Anda dapat mengaktifkan DevOps Guru untuk memonitor Amazon Aurora Anda dengan salah satu cara berikut.

Topik

- Menghidupkan DevOps Guru di RDS konsol
- Menambahkan Aurora RDS di konsol Guru DevOps
- Menambahkan Aurora RDSAWS CloudFormation

### Menghidupkan DevOps Guru di RDS konsol

Anda dapat mengambil beberapa jalur di RDS konsol Amazon untuk mengaktifkan DevOps Guru.

Topik

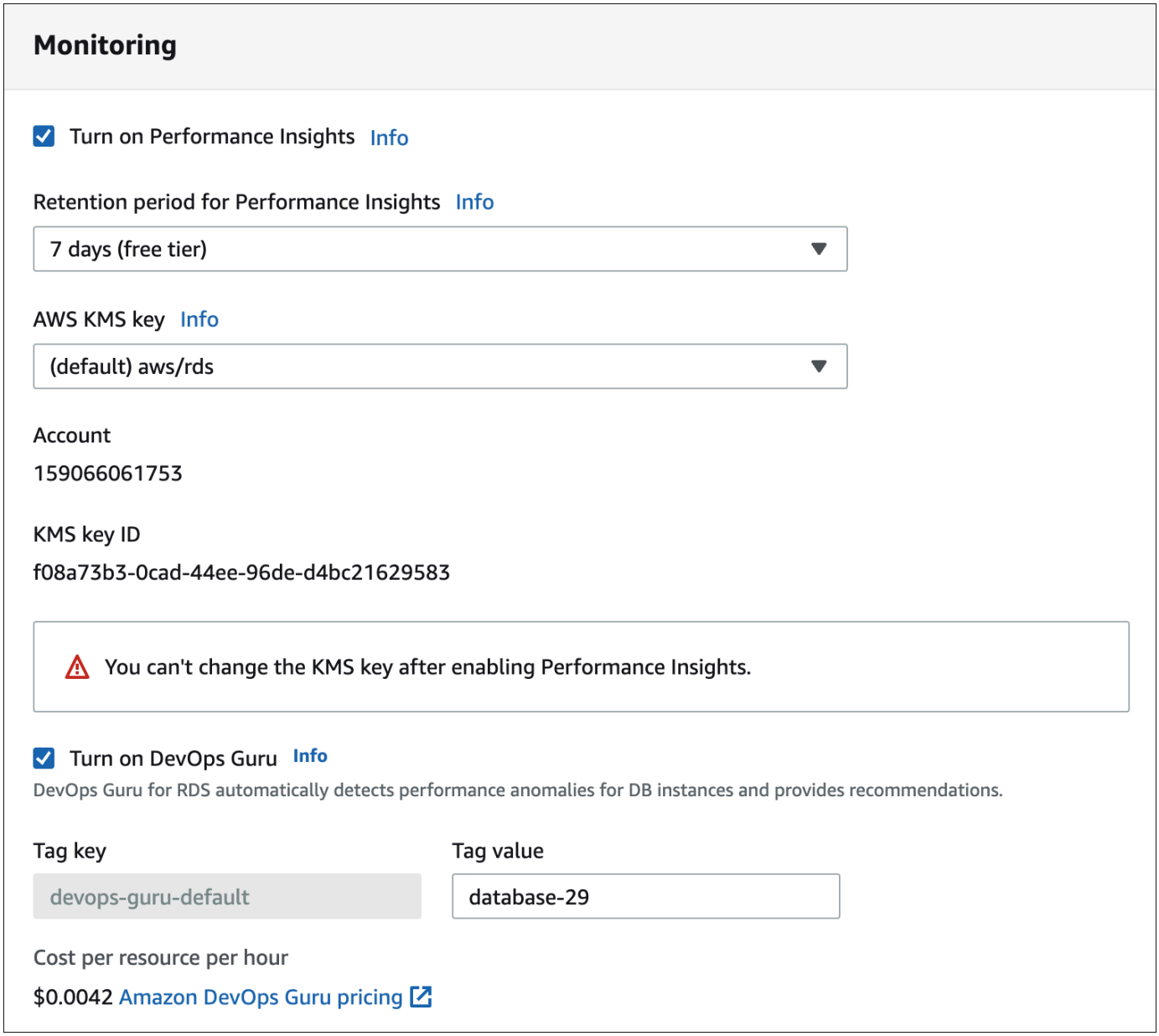
- Menghidupkan DevOps Guru saat Anda membuat Aurora RDS database Postgre SQL
- Menghidupkan DevOps Guru dari spanduk notifikasi
- Menanggapi kesalahan izin saat Anda mengaktifkan Guru DevOps

### Menghidupkan DevOps Guru saat Anda membuat Aurora RDS database Postgre SQL

Alur kerja pembuatan mencakup pengaturan yang mengaktifkan cakupan DevOps Guru untuk database Anda. Pengaturan ini diaktifkan secara default saat Anda memilih templat Produksi.

Untuk mengaktifkan DevOps Guru saat Anda membuat Aurora RDS database Postgre SQL

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Ikuti langkah-langkah di Membuat klaster DB, sampai tetapi tidak meliputi langkah ketika Anda memilih setelan pemantauan.
3. Di Pemantauan, pilih Aktifkan Wawasan Performa. RDSAgar DevOps Guru dapat memberikan analisis terperinci tentang anomali kinerja, Performance Insights harus diaktifkan.
4. Pilih Aktifkan DevOps Guru.




5. Buat tag untuk database Anda sehingga DevOps Guru dapat memantaunya. Lakukan hal-hal berikut:

   - Di bidang teks untuk Kunci tag, masukkan nama yang dimulai dengan **`Devops-Guru-`**.
   - Di bidang teks untuk Nilai tag, masukkan nilai apa pun. Misalnya, jika Anda memasukkan nama Aurora RDS Anda **`rds-database-1`** untuk SQL database , Anda juga dapat memasukkan **`rds-database-1`** sebagai nilai tag.

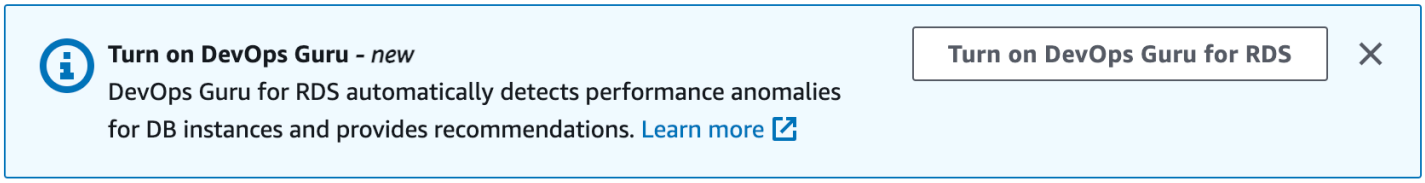
   Untuk informasi selengkapnya tentang tag, lihat "Menggunakan tag untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda" di Panduan Pengguna Amazon DevOps Guru.

6. Selesaikan langkah-langkah selebihnya di Membuat klaster DB.

Menghidupkan DevOps Guru dari spanduk notifikasi

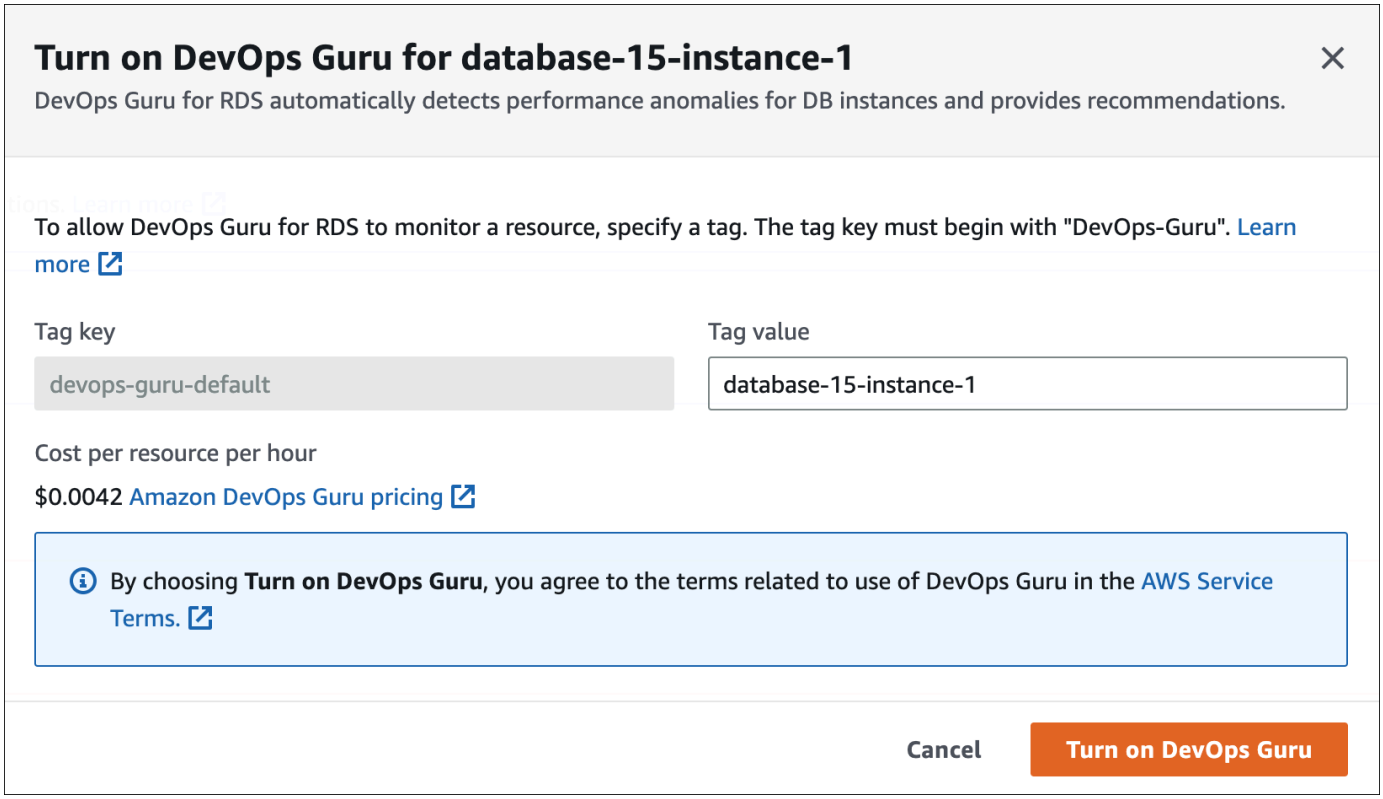
Jika sumber daya Anda tidak dicakup oleh DevOps Guru, Amazon RDS akan memberi tahu Anda dengan spanduk di lokasi berikut:

- Tab Pemantauan instans klaster basis data
- Dasbor Wawasan Performa




Untuk mengaktifkan DevOps Guru untuk Aurora Anda untuk database

1. Di spanduk, pilih Aktifkan DevOps Guru untuk RDS.
2. Masukkan nama kunci dan nilai tag. Untuk informasi selengkapnya tentang tag, lihat "Menggunakan tag untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda" di Panduan Pengguna Amazon DevOps Guru.




3. Pilih Aktifkan DevOps Guru.

Menanggapi kesalahan izin saat Anda mengaktifkan Guru DevOps

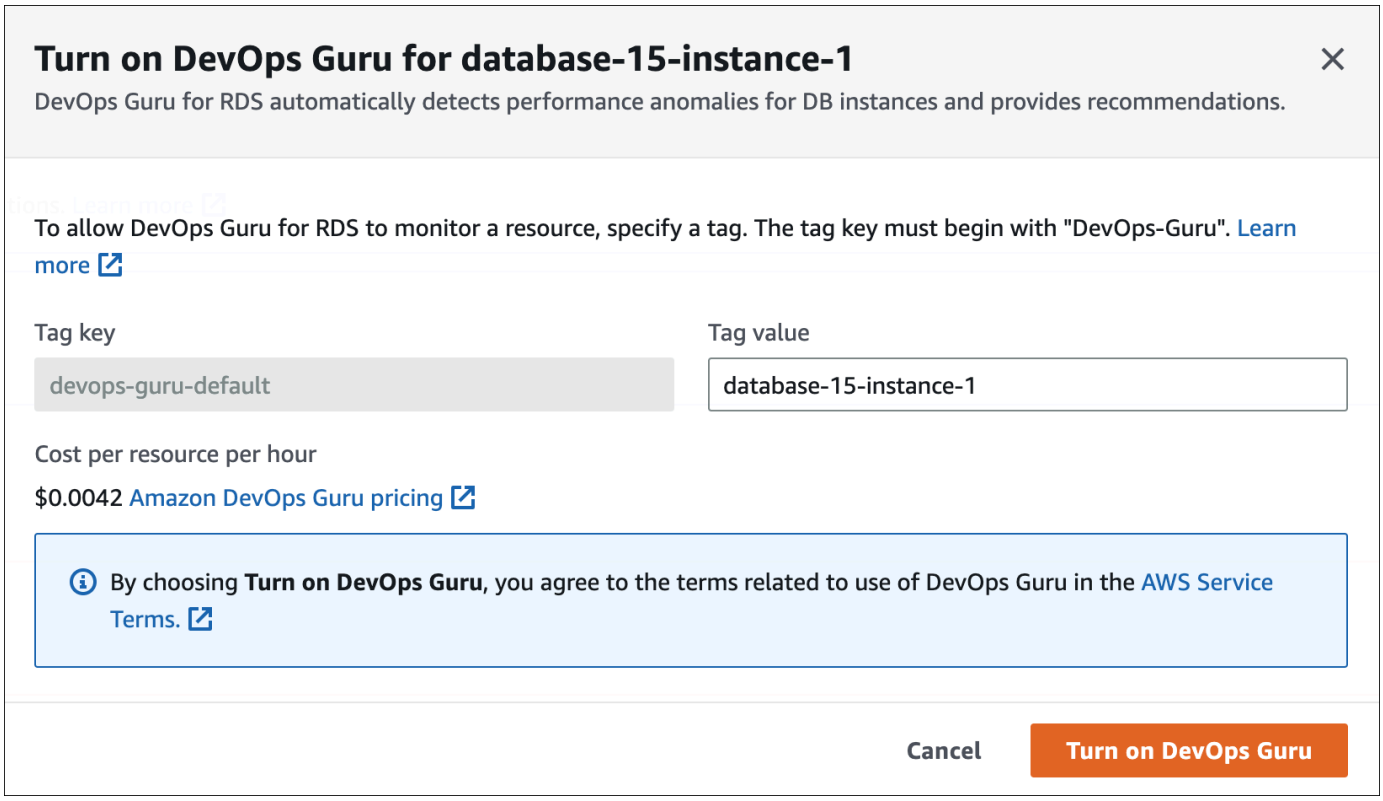
Jika Anda mengaktifkan DevOps Guru dari RDS konsol saat membuat database, RDS mungkin akan menampilkan spanduk berikut tentang izin yang hilang.

Untuk menanggapi kesalahan izin

1. Berikan IAM pengguna atau peran Anda peran yang dikelola penggunaAmazonDevOpsGuruConsoleFullAccess. Untuk informasi selengkapnya, lihat Mengkonfigurasi kebijakan IAM akses untuk DevOps Guru for RDS.
2. Buka RDS konsol.
3. Di panel navigasi, pilih Wawasan Performa.
4. Pilih instans basis data di klaster yang baru saja Anda buat.
5. Pilih sakelar untuk mengaktifkan DevOpsGuru RDS.

DevOps Guru for RDS

6. Pilih nilai tag. Untuk informasi selengkapnya, lihat "Menggunakan tag untuk mengidentifikasi sumber daya di aplikasi DevOps Guru Anda" di Panduan Pengguna Amazon DevOps Guru.




7. Pilih Aktifkan DevOps Guru.

Menambahkan Aurora RDS di konsol Guru DevOps

Anda dapat menentukan cakupan sumber daya DevOps Guru Anda di konsol DevOps Guru. Ikuti langkah yang dijelaskan dalam Tentukan cakupan sumber daya DevOps Guru Anda di Panduan Pengguna Amazon DevOps Guru. Saat Anda mengedit sumber daya yang dianalisis, pilih salah satu opsi berikut:

- Pilih Semua sumber daya akun untuk menganalisis semua sumber daya yang didukung, termasuk database Aurora RDS , di Akun AWS dan Wilayah.
- Pilih CloudFormation tumpukan untuk menganalisis RDSAurora database SQL Postgre yang ada di tumpukan yang Anda pilih. Untuk informasi selengkapnya, lihat Menggunakan AWS CloudFormation tumpukan untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda di Panduan Pengguna Amazon DevOps Guru.

- Pilih Tag untuk menganalisis Aurora RDS yang telah Anda tag. Untuk informasi selengkapnya, lihat Menggunakan tag untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda di Panduan Pengguna Amazon DevOps Guru.

Untuk informasi selengkapnya, lihat Aktifkan DevOps DevOps Guru di Panduan Pengguna Amazon Guru.

Menambahkan Aurora RDSAWS CloudFormation

Anda dapat menggunakan tag untuk menambahkan cakupan untuk Aurora RDS Anda untuk SQL sumber daya ke template Anda. CloudFormation Prosedur berikut mengasumsikan bahwa Anda memiliki CloudFormation template baik untuk  Anda untuk instans SQL Postgre DB dan tumpukan Guru. DevOps

Untuk menentukan Aurora RDS DB menggunakan tag CloudFormation

1. Dalam CloudFormation template untuk instans DB Anda, tentukan tag menggunakan pasangan kunci/nilai.

   Contoh berikut menetapkan nilai `my-aurora-db-instance1` untuk `Devops-guru-cfn-default` bagi instans basis data Aurora.

   ```
   MyAuroraDBInstance1:
     Type: "AWS::RDS::DBInstance"
     Properties:
       DBClusterIdentifier: my-aurora-db-cluster
       DBInstanceIdentifier: my-aurora-db-instance1
       Tags:
         - Key: Devops-guru-cfn-default
           Value: devopsguru-my-aurora-db-instance1
   ```

2. Dalam CloudFormation template untuk tumpukan DevOps Guru Anda, tentukan tag yang sama di filter pengumpulan sumber daya Anda.

   Contoh berikut mengkonfigurasi DevOps Guru untuk menyediakan cakupan untuk sumber daya dengan nilai `my-aurora-db-instance1` tag.

   ```
   DevOpsGuruResourceCollection:
     Type: AWS::DevOpsGuru::ResourceCollection
     Properties:
       ResourceCollectionFilter:
   ```

```
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "devopsguru-my-aurora-db-instance1"
```

Contoh berikut menyediakan cakupan untuk semua sumber daya dalam batas aplikasi `Devops-guru-cfn-default`.

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

Untuk informasi lebih lanjut, lihat AWS:DevOpsGuru::AWS: ResourceCollection dan:RDS::: DBInstance di AWS CloudFormation Panduan Pengguna.

# Memantau metrik OS dengan Pemantauan yang Ditingkatkan

Dengan Pemantauan yang Ditingkatkan, Anda dapat memantau sistem operasi instans DB Anda secara real-time. Metrik Pemantauan Ditingkatkan berguna ketika Anda ingin melihat bagaimana proses atau thread yang berbeda pada instans DB menggunakan CPU.

Topik


- Gambaran Umum Pemantauan Ditingkatkan
- Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan
- Melihat metrik OS di konsol RDS
- Melihat metrik OS menggunakan CloudWatch Log


## Gambaran Umum Pemantauan Ditingkatkan

Amazon RDS menyediakan metrik secara real-time untuk sistem operasi (OS) tempat instans DB Anda dijalankan. Anda dapat melihat semua metrik sistem dan memproses informasi instans DB RDS Anda di konsol. Anda dapat mengelola metrik yang ingin dipantau untuk setiap instans dan menyesuaikan dasbor sesuai kebutuhan Anda. Untuk deskripsi metrik Pemantauan yang Ditingkatkan, lihat Metrik OS dalam Pemantauan yang Ditingkatkan.

RDS mengirimkan metrik dari Enhanced Monitoring ke akun Amazon CloudWatch Logs Anda. Anda dapat membuat filter metrik CloudWatch dari CloudWatch Log dan menampilkan grafik di dasbor. CloudWatch Anda dapat menggunakan output Enhanced Monitoring JSON dari CloudWatch Log dalam sistem pemantauan pilihan Anda. Untuk informasi selengkapnya, lihat Pemantauan yang Ditingkatkan di Amazon RDS. FAQs

Topik


- Perbedaan antara CloudWatch dan metrik Pemantauan yang Ditingkatkan
- Retensi metrik Pemantauan yang Ditingkatkan
- Biaya Pemantauan yang Ditingkatkan


### Perbedaan antara CloudWatch dan metrik Pemantauan yang Ditingkatkan

Hypervisor membuat dan menjalankan mesin virtual ()VMs. Menggunakan hypervisor, sebuah instance dapat mendukung beberapa tamu VMs dengan berbagi memori dan CPU secara virtual.

CloudWatch mengumpulkan metrik tentang pemanfaatan CPU dari hypervisor untuk instance DB. Sebaliknya, Pemantauan yang Ditingkatkan mengumpulkan metrik dari agen di instans DB.

Anda mungkin menemukan perbedaan antara pengukuran CloudWatch dan Enhanced Monitoring, karena lapisan hypervisor melakukan sejumlah kecil pekerjaan. Perbedaan ini bisa lebih menonjol jika instans DB Anda menggunakan kelas instans yang lebih kecil. Dalam skenario ini, lebih banyak mesin virtual (VMs) mungkin dikelola oleh lapisan hypervisor pada satu instance fisik.

Untuk deskripsi metrik Pemantauan yang Ditingkatkan, lihat Metrik OS dalam Pemantauan yang Ditingkatkan. Untuk informasi selengkapnya tentang CloudWatch metrik, lihat Panduan CloudWatch Pengguna Amazon.

## Retensi metrik Pemantauan yang Ditingkatkan

Secara default, metrik Pemantauan yang Ditingkatkan disimpan selama 30 hari di CloudWatch Log. Periode retensi ini berbeda dari CloudWatch metrik biasa.

Untuk mengubah jumlah waktu metrik disimpan di CloudWatch Log, ubah retensi untuk grup `RDSOSMetrics` log di CloudWatch konsol. Untuk informasi selengkapnya, lihat Mengubah penyimpanan data CloudWatch log di log di Panduan Pengguna CloudWatch Log Amazon.

## Biaya Pemantauan yang Ditingkatkan

Metrik Pemantauan yang Ditingkatkan disimpan di CloudWatch Log, bukan dalam CloudWatch metrik. Biaya Pemantauan yang Ditingkatkan ditentukan oleh faktor-faktor berikut:

- Anda dikenakan biaya untuk Enhanced Monitoring hanya jika Anda melebihi tingkat gratis yang disediakan oleh Amazon CloudWatch Logs. Biaya didasarkan pada transfer data CloudWatch Log dan tingkat penyimpanan.
- Jumlah informasi yang ditransfer untuk instans RDS berbanding lurus dengan perincian yang ditentukan untuk fitur Pemantauan yang Ditingkatkan. Interval pemantauan yang lebih kecil menghasilkan pelaporan metrik OS yang lebih sering dan meningkatkan biaya pemantauan. Untuk mengelola biaya, atur perincian yang berbeda untuk instans yang berbeda di akun Anda.
- Biaya penggunaan Pemantauan yang Ditingkatkan diterapkan untuk setiap instans DB yang fitur Pemantauan yang Ditingkatkannya diaktifkan. Pemantauan instans DB dalam jumlah akan lebih mahal dibandingkan dengan pemantauan dalam jumlah sedikit.
- Instans DB yang mendukung beban kerja komputasi yang lebih berat memiliki lebih banyak aktivitas proses OS yang perlu dan biaya untuk Pemantauan yang Ditingkatkan lebih tinggi.

Untuk informasi selengkapnya tentang harga, lihat CloudWatch harga Amazon.

## Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan

Untuk menggunakan Pemantauan yang Ditingkatkan, Anda harus membuat peran IAM, lalu mengaktifkan Pemantauan yang Ditingkatkan.

Topik

- Membuat peran IAM untuk Pemantauan yang Ditingkatkan
- Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan
- Melindungi dari masalah confused deputy

### Membuat peran IAM untuk Pemantauan yang Ditingkatkan

Pemantauan yang Ditingkatkan memerlukan izin untuk bertindak atas nama Anda untuk mengirim informasi metrik OS ke CloudWatch Log. Anda memberikan izin Pemantauan yang Ditingkatkan menggunakan peran AWS Identity and Access Management (IAM). Anda dapat membuat peran ini saat mengaktifkan Pemantauan yang Ditingkatkan atau membuatnya terlebih dahulu.

Topik

- Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan
- Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

#### Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

Jika Anda mengaktifkan Pemantauan yang Ditingkatkan di konsol RDS, Amazon RDS dapat membuat peran IAM yang diperlukan untuk Anda. Peran ini bernama `rds-monitoring-role`. RDS menggunakan peran ini untuk instans DB, replika baca, atau klaster DB Multi-AZ tertentu.

Untuk membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

1. Ikuti langkah-langkah di Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan.
2. Atur Peran Pemantauan ke Default pada langkah tempat Anda memilih peran.

### Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

Anda dapat membuat peran yang diperlukan sebelum mengaktifkan Pemantauan yang Ditingkatkan. Jika Anda mengaktifkan Pemantauan yang Ditingkatkan, tentukan nama peran baru Anda. Anda harus membuat peran yang diperlukan ini jika Anda mengaktifkan Pemantauan yang Ditingkatkan menggunakan AWS CLI atau API RDS.

Pengguna yang mengaktifkan Pemantauan yang Ditingkatkan harus diberi izin `PassRole`. Untuk informasi selengkapnya, lihat Contoh 2 dalam Memberikan izin pengguna untuk meneruskan peran ke AWS layanan di Panduan Pengguna IAM.

Untuk membuat peran IAM untuk pemantauan yang ditingkatkan Amazon RDS

1. Buka konsol IAM di https://console.aws.amazon.com.
2. Di panel navigasi, pilih Peran.
3. Pilih Buat peran.
4. Pilih tab Layanan AWS , lalu pilih RDS dari daftar layanan.
5. Pilih RDS - Pemantauan yang Ditingkatkan, lalu pilih Berikutnya.
6. Pastikan kebijakan Izin menampilkan Amazon RDSEnhanced MonitoringRole, lalu pilih Berikutnya.
7. Untuk Nama peran, masukkan nama peran Anda. Misalnya, masukkan **`emaccess`**.

   Entitas tepercaya untuk peran Anda adalah AWS layanan monitoring.rds.amazonaws.com.
8. Pilih Buat peran.

## Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan

Anda dapat mengelola Enhanced Monitoring menggunakan AWS Management Console, AWS CLI, atau RDS API. Anda dapat mengatur granularitas yang berbeda untuk pengumpulan metrik pada setiap instans DB cluster DB . Anda juga dapat mengaktifkan Enhanced Monitoring untuk cluster DB yang ada dari konsol.

### Konsol

Anda dapat mengaktifkan Enhanced Monitoring saat membuat atau replika baca, atau saat Anda memodifikasi . Jika Anda memodifikasi instans atau cluster DB untuk mengaktifkan Enhanced Monitoring, Anda tidak perlu me-reboot instans DB Anda agar perubahan diterapkan.

Anda dapat mengaktifkan Pemantauan yang Ditingkatkan di konsol RDS saat Anda melakukan salah satu tindakan berikut di halaman Basis data:

- Buat klaster – Pilih Buat basis data.
- Buat replika baca – Pilih Tindakan, lalu Buat replika baca.
- Ubah instans DB atau cluster DB atau cluster — Pilih Ubah.

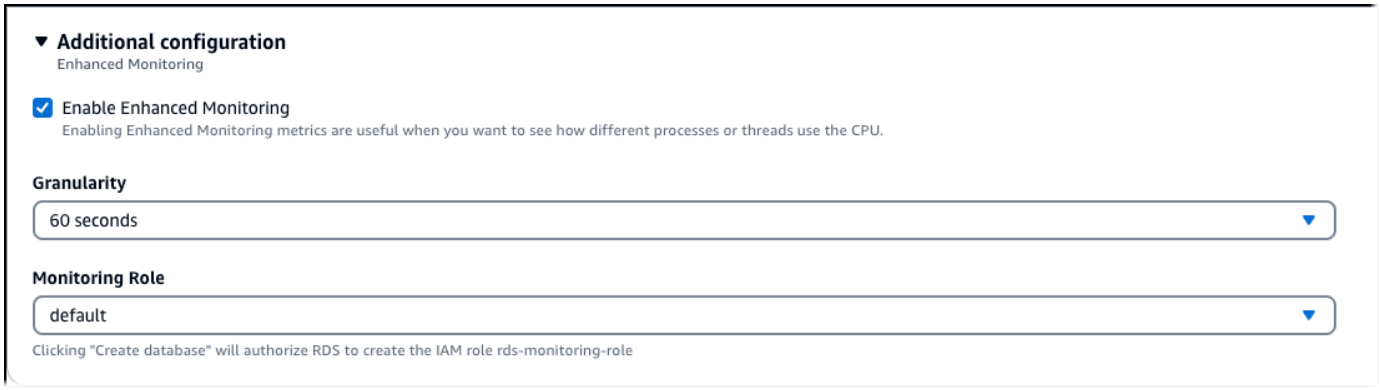
> **Note**
>
> Saat mengaktifkan Enhanced Monitoring untuk klaster DB, Anda tidak dapat mengelola Enhanced Monitoring untuk instans DB individual di dalam cluster.

Jika Anda mengelola Enhanced Monitoring untuk instans DB individual dalam cluster DB dan granularitas diatur ke nilai yang berbeda untuk instans yang berbeda, cluster DB Anda heterogen sehubungan dengan Enhanced Monitoring. Dalam kasus seperti itu, Anda tidak dapat memodifikasi cluster DB untuk mengelola Enhanced Monitoring di tingkat cluster.

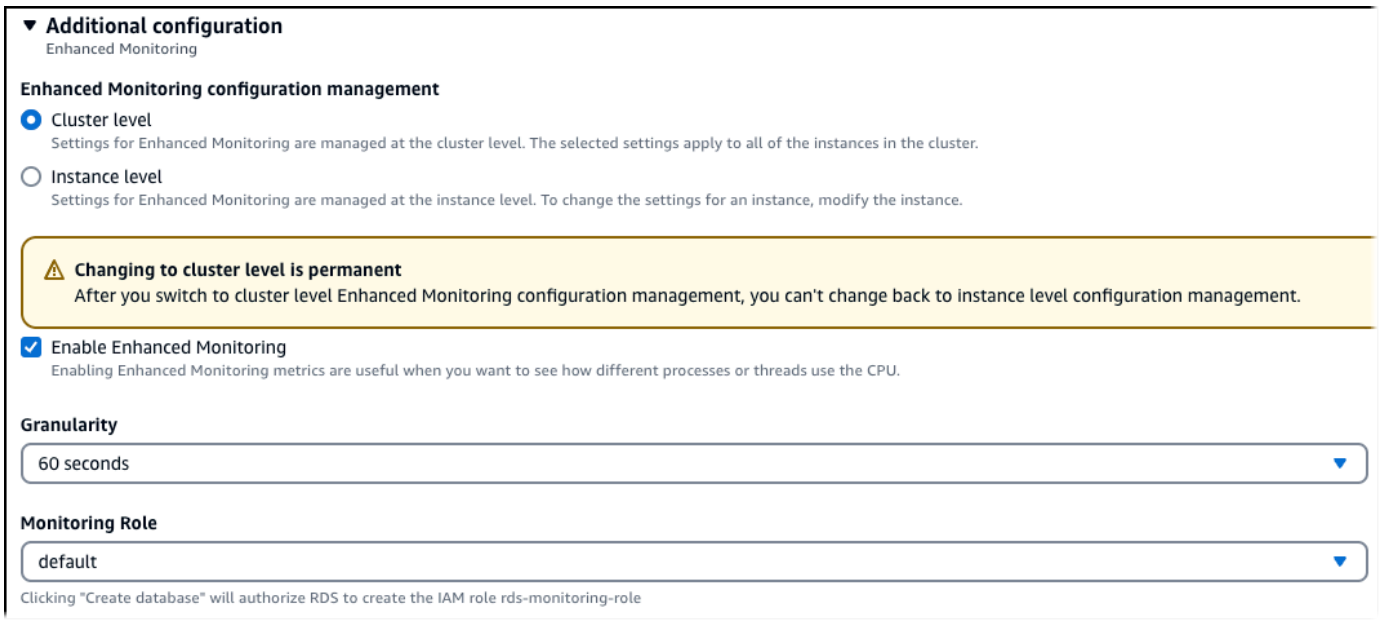
Untuk mengaktifkan atau menonaktifkan Pemantauan yang Ditingkatkan di konsol RDS

1. Gulir ke bagian Konfigurasi tambahan.
2. Di Monitoring, pilih Aktifkan Pemantauan yang Ditingkatkan untuk DB, cluster, atau replika baca Anda. Mengaktifkan Enhanced Monitoring di tingkat cluster memungkinkan Anda mengelola pengaturan dan opsi Enhanced Monitoring di tingkat cluster. Pengaturan tingkat cluster berlaku untuk semua instance DB cluster.Hapus pilihan untuk menonaktifkan Enhanced Monitoring di tingkat cluster. Anda nantinya dapat memodifikasi pengaturan Enhanced Monitoring untuk instans DB individual di cluster.

   Di halaman Buat database, Anda dapat memilih untuk mengaktifkan Enhanced Monitoring di tingkat cluster.




Jika Anda tidak mengaktifkan Enhanced Monitoring saat membuat cluster, Anda dapat memodifikasi cluster di halaman cluster Modify DB.




> **Note**
>
> Anda tidak dapat mengelola Enhanced Monitoring untuk instans DB individual di cluster DB yang sudah memiliki Enhanced Monitoring yang dikelola di tingkat cluster.

3. Anda tidak dapat memilih untuk mengelola Enhanced Monitoring di tingkat cluster jika klaster heterogen sehubungan dengan Enhanced Monitoring. Untuk mengelola Enhanced Monitoring di tingkat cluster, ubah pengaturan Enhanced Monitoring untuk setiap instance agar cocok. Sekarang Anda dapat memilih untuk mengelola Enhanced Monitoring di tingkat cluster saat Anda memodifikasi klaster Anda.

4. Setel properti Peran Pemantauan ke peran IAM yang Anda buat untuk mengizinkan Amazon RDS berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default agar RDS membuat peran untuk Anda namai. `rds-monitoring-role`
5. Atur properti Granularity ke interval, dalam hitungan detik, di antara titik saat metrik dikumpulkan untuk instans DB, cluster DB, atau replika baca Anda. Properti Granularitas dapat diatur ke salah satu nilai berikut: `1`, `5`, `10`, `15`, `30`, atau `60`.

   Waktu yang paling cepat di mana konsol RDS disegarkan adalah setiap 5 detik. Jika Anda mengatur granularitas ke 1 detik di konsol RDS, Anda masih dapat melihat metrik yang diperbarui hanya setiap 5 detik. Anda dapat mengambil pembaruan metrik 1 detik dengan menggunakan CloudWatch Log.

AWS CLI

Untuk mengaktifkan Enhanced Monitoring menggunakan AWS CLI, dalam perintah berikut, atur `--monitoring-interval` opsi ke nilai selain `0` dan atur `--monitoring-role-arn` opsi ke peran yang Anda buatMembuat peran IAM untuk Pemantauan yang Ditingkatkan.

- create-db-cluster
- modify-db-cluster
- create-db-instance
- create-db-instance-read-replika
- modify-db-instance

Opsi `--monitoring-interval` menentukan interval, dalam detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan. Nilai yang valid untuk opsi ini adalah `0`, `1`, `5`, `10`, `15`, `30`, dan `60`.

Untuk mematikan Enhanced Monitoring menggunakan AWS CLI, atur `--monitoring-interval` opsi ke `0` dalam perintah ini.

Example

Contoh berikut mengaktifkan Pemantauan yang Ditingkatkan untuk instans DB:

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
```

```
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

Contoh berikut mengaktifkan Enhanced Monitoring untuk cluster DB:

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

Contoh berikut mengaktifkan Pemantauan yang Ditingkatkan untuk klaster DB Multi-AZ:

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

API RDS

Untuk mengaktifkan Pemantauan yang Ditingkatkan menggunakan RDS API, atur parameter `MonitoringInterval` ke nilai selain `0` dan atur parameter `MonitoringRoleArn` ke peran yang Anda buat di Membuat peran IAM untuk Pemantauan yang Ditingkatkan. Tetapkan parameter ini dalam tindakan berikut:

- Buat DBCluster
- Memodifikasi DBCluster
- Buat DBInstance
- Buat DBInstance ReadReplica
- Memodifikasi DBInstance

Parameter `MonitoringInterval` menentukan interval, dalam detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan. Nilai yang valid adalah `0`, `1`, `5`, `10`, `15`, `30`, dan `60`.

Untuk menonaktifkan Pemantauan yang Ditingkatkan menggunakan API RDS, atur `MonitoringInterval` ke `0`.

## Melindungi dari masalah confused deputy

Masalah deputi yang bingung adalah masalah keamanan di mana entitas yang tidak memiliki izin untuk melakukan tindakan dapat memaksa entitas yang lebih istimewa untuk melakukan tindakan. Pada tahun AWS, peniruan lintas layanan dapat mengakibatkan masalah wakil yang membingungkan. Peniruan identitas lintas layanan dapat terjadi ketika satu layanan (layanan yang dipanggil) memanggil layanan lain (layanan yang dipanggil). Layanan pemanggilan dapat dimanipulasi menggunakan izinnya untuk bertindak pada sumber daya pelanggan lain dengan cara yang seharusnya tidak dilakukannya kecuali bila memiliki izin untuk mengakses. Untuk mencegah hal ini, AWS menyediakan alat yang membantu Anda melindungi data untuk semua layanan dengan pengguna utama layanan yang telah diberi akses ke sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Masalah confused deputy.

Untuk membatasi izin ke sumber daya yang dapat diberikan Amazon RDS kepada layanan lain, sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan kepercayaan untuk peran Pemantauan yang Ditingkatkan. Jika Anda menggunakan kedua kunci konteks kondisi global, keduanya harus menggunakan ID akun yang sama.

Cara paling efektif untuk melindungi dari masalah confused deputy adalah dengan menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN lengkap sumber daya. Untuk Amazon RDS, atur `aws:SourceArn` ke `arn:aws:rds:`*`Region`*`:`*`my-account-id`*`:db:`*`dbname`*.

Contoh berikut menggunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan kepercayaan untuk mencegah masalah deputi yang bingung.

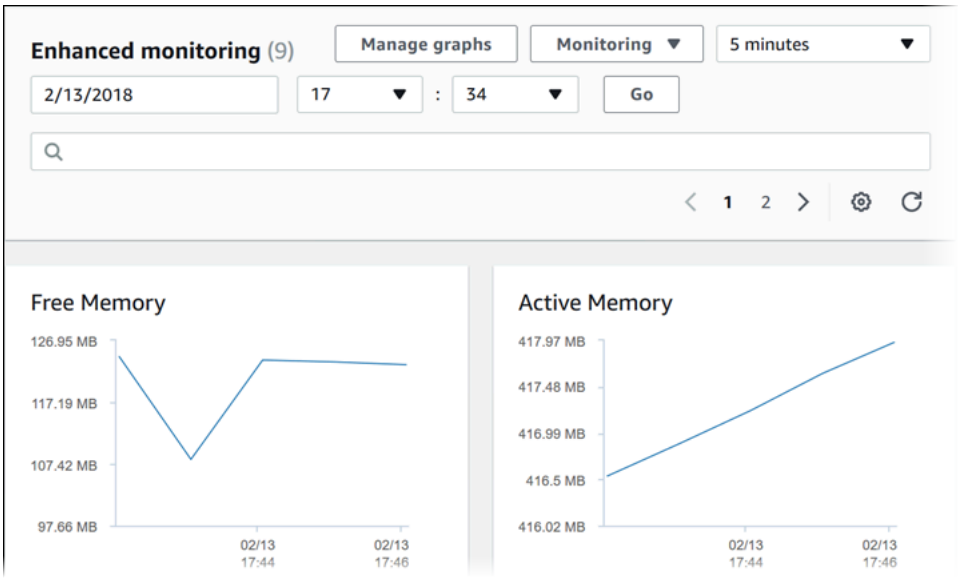
JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-account-id"
        }
      }
    }
  ]
}
```

## Melihat metrik OS di konsol RDS

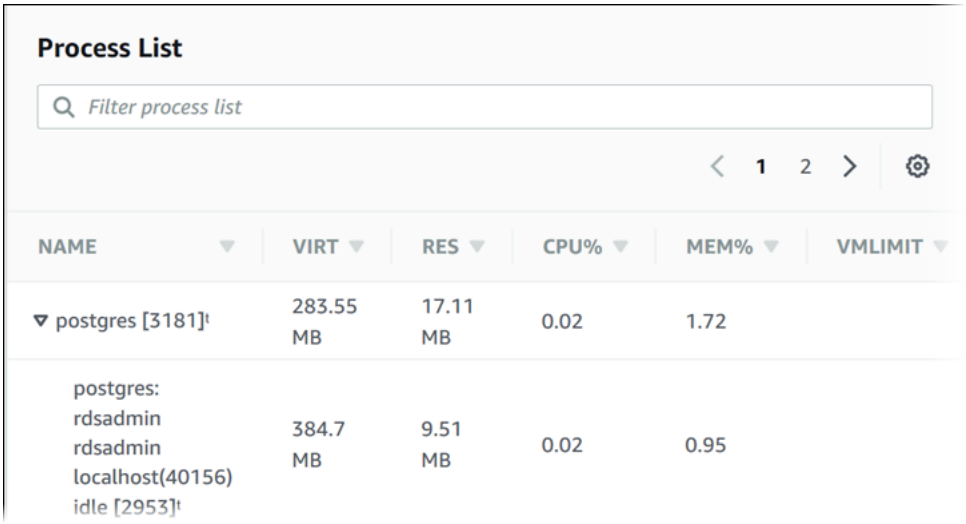
Anda dapat melihat metrik OS yang dilaporkan oleh Enhanced Monitoring di RDS konsol dengan memilih Pemantauan yang ditingkatkan untuk Pemantauan.

Contoh berikut menunjukkan halaman Pemantauan yang Ditingkatkan. Untuk deskripsi metrik Pemantauan yang Ditingkatkan, lihat Metrik OS dalam Pemantauan yang Ditingkatkan.




Jika Anda ingin melihat detail proses yang berjalan pada instans DB Anda, pilih Daftar proses OS untuk Pemantauan.

Tampilan Daftar Proses ditunjukkan sebagai berikut.




Metrik Pemantauan yang Ditingkatkan yang ditunjukkan dalam tampilan Daftar proses disusun sebagai berikut:

- RDSproses anak — Menampilkan ringkasan RDS proses yang mendukung instans DB, misalnya `aurora`untuk cluster Amazon Aurora DB `mysqld` . Rangkaian proses muncul bersarang di bawah proses induk. Thread proses menunjukkan CPU pemanfaatan hanya karena metrik lainnya

sama untuk semua utas untuk proses tersebut. Konsol menampilkan maksimal 100 proses dan rangkaian. Hasilnya adalah kombinasi dari proses dan utas CPU konsumsi dan memori teratas. Jika ada lebih dari 50 proses dan lebih dari 50 rangkaian, konsol akan menampilkan 50 pengonsumsi teratas di setiap kategori. Tampilan ini membantu Anda mengidentifikasi proses mana yang memiliki dampak paling besar pada performa.

- RDSproses — Menunjukkan ringkasan sumber daya yang digunakan oleh agen RDS manajemen, proses pemantauan diagnostik, dan AWS proses lain yang diperlukan untuk mendukung instans RDS DB.
- Proses OS – Menampilkan ringkasan proses sistem dan kernel, yang umumnya berdampak minimal pada performa.

Item yang tercantum untuk setiap proses meliputi:

- VIRT— Menampilkan ukuran virtual proses.
- RES— Menampilkan memori fisik aktual yang digunakan oleh proses.
- CPU% — Menampilkan persentase dari total CPU bandwidth yang digunakan oleh proses.
- MEM% — Menampilkan persentase dari total memori yang digunakan oleh proses.

Data pemantauan yang ditampilkan di RDS konsol diambil dari Amazon CloudWatch Logs. Anda juga dapat mengambil metrik untuk instans DB sebagai aliran log dari CloudWatch Log. Untuk informasi selengkapnya, lihat Melihat metrik OS menggunakan CloudWatch Log.

Metrik Pemantauan yang Ditingkatkan tidak ditampilkan selama:

- Failover instans DB.
- Mengubah kelas instans dari instans DB (komputasi skala).

Metrik Pemantauan yang Ditingkatkan ditampilkan selama proses boot ulang instans DB karena hanya mesin basis data yang di-boot ulang. Metrik untuk sistem operasi tetap dilaporkan.

## Melihat metrik OS menggunakan CloudWatch Log

Setelah mengaktifkan Enhanced Monitoring untuk , Anda dapat melihat metriknya menggunakan CloudWatch Log, dengan setiap aliran log mewakili instans DB tunggal atau cluster DB yang dipantau. Pengidentifikasi log stream adalah pengidentifikasi sumber daya (`DbiResourceId`) untuk instans DB atau klaster DB.

Untuk melihat data log Pemantauan yang Ditingkatkan

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Jika perlu, pilih Anda. Wilayah AWS Untuk informasi selengkapnya, lihat Wilayah dan titik akhir dalam Referensi Umum Amazon Web Services.
3. Pilih Log di panel navigasi.
4. Pilih RDSOSMetricsdari daftar grup log.
5. Pilih log stream yang ingin dilihat dari daftar log stream.

# Referensi metrik untuk Aurora

Dalam referensi ini, Anda dapat menemukan deskripsi metrik Amazon Aurora Amazon untuk , Performance Insights, dan Enhanced Monitoring.

Topik


- CloudWatch Metrik Amazon untuk Amazon Aurora
- CloudWatch Dimensi Amazon untuk Aurora
- Ketersediaan metrik Aurora di konsol Amazon RDS
- CloudWatch Metrik Amazon untuk Amazon RDS Performance Insights
- Metrik penghitung Wawasan Performa
- Statistik SQL untuk Wawasan Performa
- Metrik OS dalam Pemantauan yang Ditingkatkan


## CloudWatch Metrik Amazon untuk Amazon Aurora

Ruang nama `AWS/RDS` mencakup metrik berikut yang berlaku untuk entitas basis data yang berjalan di Amazon Aurora. Beberapa metrik berlaku untuk Aurora MySQL, Aurora PostgreSQL, atau keduanya. Selain itu, beberapa metrik bersifat spesifik untuk klaster DB, instans DB replika, atau semua instans DB.

Untuk metrik basis data global Aurora, lihat CloudWatch Metrik Amazon untuk penerusan tulis di Aurora MySQL dan CloudWatch Metrik Amazon untuk penerusan tulis di Aurora PostgreSQL. Untuk metrik kueri paralel Aurora, lihat Memantau query paralel untuk Aurora My SQL.

Topik


- Metrik tingkat klaster untuk Amazon Aurora
- Metrik tingkat instans untuk Amazon Aurora
- Metrik CloudWatch penggunaan Amazon untuk


### Metrik tingkat klaster untuk Amazon Aurora

Tabel berikut menjelaskan metrik yang spesifik untuk klaster Aurora.

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraGlobalDBDataTransferBytes` | Dalam Database Global Aurora, jumlah data redo log yang ditransfer dari AWS Wilayah sumber ke Wilayah sekunder. AWS<br><br>**Note**<br>Metrik ini hanya tersedia di Wilayah AWS sekunder. | Aurora MySQL dan Aurora PostgreSQL | Byte |
| `AuroraGlobalDBProgressLag` | Dalam Basis Data Global Aurora, ukuran seberapa jauh klaster sekunder tertinggal dari klaster primer untuk transaksi pengguna dan transaksi sistem.<br><br>**Note**<br>Metrik ini hanya tersedia di Wilayah AWS sekunder. | Aurora MySQL dan Aurora PostgreSQL | Milidetik |
| `AuroraGlobalDBReplicatedWriteIO` | Dalam Database Global Aurora, jumlah I/O operasi tulis direplikasi dari AWS Region primer ke volume cluster di Region sekunder. AWS Perhitungan penagihan untuk AWS Wilayah sekunder dalam | Aurora MySQL dan Aurora PostgreSQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | database global digunakan `VolumeWriteIOPs` untuk memperhitungkan penulisan yang dilakukan dalam cluster. Perhitungan penagihan untuk AWS Wilayah utama dalam database global digunakan `VolumeWriteIOPs` untuk memperhitungkan aktivitas penulisan dalam klaster tersebut, dan `AuroraGlobalDBReplicatedWriteIO` untuk memperhitungkan replikasi Lintas wilayah dalam database global.<br><br>**Note**<br>Metrik ini hanya tersedia di Wilayah AWS sekunder. | | |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraGlobalDBReplicationLag` | Untuk Basis Data Global Aurora, jumlah lag saat mereplikasi pembaruan dari Wilayah AWS primer.<br><br>**Note**<br>Metrik ini hanya tersedia di Wilayah AWS sekunder. | Aurora MySQL dan Aurora PostgreSQL | Milidetik |
| `AuroraGlobalDBRPOLag` | Dalam Basis Data Aurora Global, waktu lag sasaran titik pemulihan (RPO). Metrik ini mengukur seberapa jauh klaster sekunder tertinggal dari klaster primer untuk transaksi pengguna.<br><br>**Note**<br>Metrik ini hanya tersedia di Wilayah AWS sekunder. | Aurora MySQL dan Aurora PostgreSQL | Milidetik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraVolumeBytesLeftTotal` | Sisa ruang yang tersedia untuk volume klaster. Saat volume klaster bertambah , nilai ini menurun. Jika mencapai nol, cluster melaporkan out-of-space kesalahan.<br><br>Jika Anda ingin mendeteksi apakah klaster Aurora MySQL Anda mendekati batas ukuran 128 tebibytes (TiB), nilai ini lebih sederhana dan lebih dapat diandalkan untuk dipantau dibandingkan `VolumeBytesUsed` . `AuroraVolumeBytesLeftTotal` memperhitungkan penyimpanan yang digunakan untuk housekeeping internal dan alokasi lain yang tidak memengaruhi tagihan penyimpanan Anda. | Aurora MySQL | Byte |
| `BacktrackChangeRecordsCreationRate` | Jumlah catatan perubahan pelacakan mundur yang dibuat selama 5 menit untuk klaster DB Anda. | Aurora MySQL | Hitungan per 5 menit |
| `BacktrackChangeRecordsStored` | Jumlah catatan perubahan pelacakan mundur yang digunakan oleh klaster DB Anda. | Aurora MySQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `BackupRetentionPeriodStorageUsed` | Jumlah total penyimpanan cadangan yang digunakan untuk mendukung fitur point-in-time pemulihan dalam jendela retensi cadangan cluster Aurora DB. Jumlah ini termasuk dalam total yang dilaporkan oleh metrik `TotalBackupStorageBilled` . Metrik ini dihitung secara terpisah untuk setiap klaster Aurora. Untuk petunjuk, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. | Aurora MySQL dan Aurora PostgreSQL | Byte |
| `ServerlessDatabaseCapacity` | Kapasitas klaster DB Aurora Serverless saat ini. | Aurora MySQL dan Aurora PostgreSQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `SnapshotStorageUsed` | Jumlah total penyimpanan cadangan yang digunakan oleh semua snapshot Aurora untuk klaster DB Aurora di luar periode penyimpanan cadangan. Jumlah ini termasuk dalam total yang dilaporkan oleh metrik `TotalBackupStorageBilled` . Metrik ini dihitung secara terpisah untuk setiap klaster Aurora. Untuk petunjuk, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. | Aurora MySQL dan Aurora PostgreSQL | Byte |
| `TotalBackupStorageBilled` | Jumlah total penyimpanan cadangan dalam byte yang akan ditagihkan untuk klaster DB Aurora tertentu. Metrik ini mencakup penyimpanan cadangan yang diukur oleh metrik `BackupRetentionPeriodStorageUsed` dan `SnapshotStorageUsed` . Metrik ini dihitung secara terpisah untuk setiap klaster Aurora. Untuk petunjuk, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. | Aurora MySQL dan Aurora PostgreSQL | Byte |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `VolumeBytesUsed` | Jumlah penyimpanan yang digunakan oleh cluster Aurora DB Anda.<br><br>Nilai ini memengaruhi biaya klaster DB Aurora (untuk informasi harga, lihat halaman harga Amazon RDS).<br><br>Nilai ini tidak mencermin kan beberapa alokasi penyimpanan internal yang tidak memengaru hi tagihan penyimpan an. Untuk Aurora MySQL Anda dapat mengantis ipasi out-of-space masalah lebih akurat dengan menguji `AuroraVol umeBytesLeftTotal` apakah mendekati nol daripada membandingkan `VolumeBytesUsed` dengan batas penyimpanan 128 TiB.<br><br>Untuk cluster yang merupakan klon, nilai metrik ini tergantung pada jumlah data yang ditambahkan atau diubah pada klon. Metrik juga dapat meningkat atau berkurang ketika cluster | Aurora MySQL dan Aurora PostgreSQL | Byte |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | asli dihapus, atau sebagai klon baru ditambahkan atau dihapus. Untuk detailnya , lihat Menghapus volume klaster sumber<br><br>Perhatikan bahwa tidak masuk akal untuk memilih `--period` nilai yang kecil, karena Amazon RDS mengumpulkan metrik ini secara berkala, tidak terus menerus. | | |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `VolumeReadIOPs` | Jumlah I/O operasi baca yang ditagih dari volume cluster dalam interval 5 menit.<br><br>Operasi baca yang ditagih dihitung pada tingkat volume klaster, dikumpulkan dari semua instans di klaster DB Aurora, lalu dilaporkan pada interval 5 menit. Nilai ini dihitung dengan mengambil nilai dari metrik Operasi baca selama periode 5 menit. Anda dapat menentukan jumlah operasi baca yang ditagih per detik dengan mengambil nilai metrik Operasi baca yang ditagih dan membaginya dengan 300 detik. Misalnya, jika Operasi baca yang ditagih menghasilkan 13.686, maka operasi baca yang ditagih per detik adalah 45 (13.686 / 300 = 45,62).<br><br>Anda mengakumulasi operasi baca yang ditagih untuk kueri yang meminta halaman basis data yang tidak ada dalam cache buffer dan harus dimuat | Aurora MySQL dan Aurora PostgreSQL | Hitungan per 5 menit |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | dari penyimpanan. Anda mungkin melihat lonjakan dalam operasi baca yang ditagih karena hasil kueri dibaca dari penyimpanan, lalu dimuat ke dalam cache buffer.<br><br>Perhatikan bahwa tidak masuk akal untuk memilih `--period` nilai yang kecil, karena Amazon RDS mengumpulkan metrik ini secara berkala, tidak terus menerus.<br><br>**Tip**<br>Jika klaster Aurora MySQL Anda menggunakan kueri paralel, Anda mungkin melihat peningkatan nilai `VolumeReadIOPS` . Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kuerinya cepat, pemrosesan yang dioptimalkan ini dapat menghasil | | |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | kan peningkatan operasi baca dan biaya terkait. | | |
| `VolumeWriteIOPs` | Jumlah I/O operasi tulis disk ke volume cluster, dilaporkan pada interval 5 menit. Untuk penjelasan mendetail tentang cara menghitung operasi tulis yang ditagih, lihat `VolumeReadIOPs` .<br><br>Perhatikan bahwa tidak masuk akal untuk memilih `--period` nilai yang kecil, karena Amazon RDS mengumpulkan metrik ini secara berkala, tidak terus menerus. | Aurora MySQL dan Aurora PostgreSQL | Hitungan per 5 menit |

## Metrik tingkat instans untuk Amazon Aurora

Metrik CloudWatch Amazon khusus instance berikut berlaku untuk semua instance Aurora MySQL dan Aurora PostgreSQL kecuali dinyatakan lain.

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AbortedClients` | Jumlah koneksi klien yang belum ditutup dengan benar. | Aurora MySQL | Hitungan |
| `ActiveTransactions` | Rata-rata jumlah transaksi saat ini yang dijalankan pada | Aurora MySQL | Hitungan per detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | instans basis data Aurora per detik.<br><br>Secara default, Aurora tidak mengaktifkan metrik ini. Untuk mulai mengukur nilai ini, tetapkan `innodb_monitor_enable='all'` dalam grup parameter DB untuk instans DB spesifik. | | |
| `ACUUtilization` | Nilai metrik `ServerlessDatabaseCapacity` dibagi dengan nilai ACU maksimum klaster DB.<br><br>Metrik ini hanya berlaku untuk Aurora Serverless v2. | Aurora MySQL dan Aurora PostgreSQL | Persentase |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraBinlogReplicaLag` | Jumlah waktu cluster DB replika log biner yang berjalan di Aurora MySQL tertinggal di belakang sumber replikasi log biner. Lag berarti bahwa sumber menghasilkan catatan lebih cepat daripada replika dapat menerapkannya.<br><br>Metrik ini melaporkan nilai yang berbeda tergantung pada versi mesin:<br><br>Aurora MySQL versi 2<br><br>Bidang `Seconds_Behind_Master` dari `SHOW SLAVE STATUS` MySQL<br><br>Aurora MySQL versi 3<br><br>`SHOW REPLICA STATUS`<br><br>Anda dapat menggunakan metrik ini untuk memantau kesalahan dan lag replika dalam klaster yang bertindak sebagai replika log biner. Nilai metrik menunjukkan hal berikut: | Primer untuk Aurora MySQL | Detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | Nilai tinggi<br><br>Replika mengalami lag dari sumber replikasi.<br><br>**0** atau nilai yang mendekati **0**<br><br>Proses replika aktif dan terkini.<br><br>**-1**<br><br>Aurora tidak dapat menentukan lag, yang dapat terjadi selama pengaturan replika atau ketika replika mengalami kesalahan.<br><br>Karena replikasi log biner hanya terjadi pada instans penulis klaster, sebaiknya gunakan versi metrik ini yang terkait dengan peran WRITER.<br><br>Untuk informasi selengkap nya tentang melakukan replikasi, lihat Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS. Untuk informasi selengkapnya tentang pemecahan masalah, lihat Masalah replikasi Amazon Aurora MySQL. | | |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraDMLRejectedMasterFull` | Jumlah kueri yang diteruskan yang ditolak karena sesi penuh pada instance DB penulis. | Primer untuk Aurora MySQL versi 2 | Hitungan |
| `AuroraDMLRejectedWriterFull` | Jumlah kueri yang diteruskan yang ditolak karena sesi penuh pada instance DB penulis. | Primer untuk Aurora MySQL versi 3 | Hitungan |
| `AuroraEstimatedSharedMemoryBytes` | Perkiraan jumlah memori buffer bersama atau pool buffer yang digunakan secara aktif selama interval polling yang terakhir dikonfigurasi. | Aurora PostgreSQL | Byte |
| `AuroraMemoryHealthState` | Menunjukkan kondisi kesehatan memori. Nilai yang `0` sama. `NORMAL` Nilai `10` sama`RESERVED`, yang berarti bahwa server mendekati tingkat kritis penggunaan memori.<br><br>Untuk informasi selengkapnya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.06.1 dan lebih tinggi | Pengukur |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraMemoryNumDeclinedSqlTotal` | Jumlah kueri tambahan ditolak sebagai bagian dari penghindaran out-of-memory (OOM).<br><br>Untuk informasi selengkapnya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.06.1 dan lebih tinggi | Hitungan |
| `AuroraMemoryNumKilledConnTotal` | Jumlah tambahan koneksi ditutup sebagai bagian dari penghindaran OOM.<br><br>Untuk informasi selengkapnya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.06.1 dan lebih tinggi | Hitungan |
| `AuroraMemoryNumKilledQueryTotal` | Jumlah tambahan kueri berakhir sebagai bagian dari penghindaran OOM.<br><br>Untuk informasi selengkapnya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.06.1 dan lebih tinggi | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraMilliseconds SpentInOomRecovery` | Jumlah waktu sejak kesehatan memori turun di bawah keadaan normal.<br><br>Untuk informasi selengkap nya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.08.0 dan lebih tinggi | Milidetik |
| `AuroraNumOomRecove rySuccessful` | Berapa kali kesehatan memori dikembalikan ke keadaan normal.<br><br>Untuk informasi selengkap nya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.08.0 dan lebih tinggi | Hitungan |
| `AuroraNumOomRecove ryTriggered` | Berapa kali kesehatan memori turun di bawah keadaan normal.<br><br>Untuk informasi selengkap nya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. | Aurora MySQL versi 3.08.0 dan lebih tinggi | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraOptimizedReadsCacheHitRatio` | Persentase permintaan yang dilayani oleh cache Bacaan yang Dioptimalkan.<br><br>Nilai dihitung menggunakan rumus berikut:<br><br>`orcache_blks_hit/(orcache_blks_hit + storage_blks_read)`<br><br>Kapan `AuroraOptimizedReadsCacheHitRatio` 100%, itu berarti bahwa semua halaman dibaca dari cache pembacaan yang dioptimalkan. Jika ya`0`, `AuroraOptimizedReadsCacheHitRatio` itu berarti tidak ada halaman yang dibaca dari cache pembacaan yang dioptimalkan. | Primer untuk Aurora PostgreSQL | Persentase |
| `AuroraReplicaLag` | Untuk replika Aurora, jumlah lag saat mereplikasi pembaruan dari instans primer. | Replika untuk Aurora MySQL dan Aurora PostgreSQL | Milidetik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `AuroraReplicaLagMaximum` | Jumlah maksimum lag antara instans primer dan instans DB Aurora dalam klaster DB.<br><br>Ketika replika baca dihapus atau diganti namanya, mungkin ada lonjakan sementara dalam kelambatan replikasi karena sumber daya lama mengalami proses daur ulang. Untuk mendapatkan representasi akurat dari lag replikasi selama periode tersebut, kami sarankan Anda memantau `AuroraReplicaLag` metrik pada setiap instance replika baca. | Primer untuk Aurora MySQL dan Aurora PostgreSQL | Milidetik |
| `AuroraReplicaLagMinimum` | Jumlah minimum lag antara instans primer dan instans DB Aurora dalam klaster DB. | Primer untuk Aurora MySQL dan Aurora PostgreSQL | Milidetik |
| `AuroraSlowConnectionHandleCount` | Jumlah koneksi yang telah menunggu dua detik atau lebih lama untuk memulai handshake.<br><br>Metrik ini hanya berlaku untuk Aurora MySQL versi 3. | Aurora MySQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| AuroraSlowHandshakeCount | Jumlah koneksi yang membutuhkan waktu 50 milidetik atau lebih lama untuk menyelesaikan handshake.<br><br>Metrik ini hanya berlaku untuk Aurora MySQL versi 3. | Aurora MySQL | Hitungan |
| BacktrackWindowActual | Perbedaan antara periode pelacakan mundur target dan periode pelacakan mundur sebenarnya. | Primer untuk Aurora MySQL | Menit |
| BacktrackWindowAlert | Jumlah berapa kali periode pelacakan mundur sebenarnya lebih kecil daripada periode pelacakan mundur target untuk periode waktu tertentu. | Primer untuk Aurora MySQL | Hitungan |
| BlockedTransactions | Jumlah rata-rata transaksi dalam basis data yang diblok detik. | Aurora MySQL | Hitungan per detik |
| BufferCacheHitRatio | Persentase permintaan yang dilayani oleh cache buffer. | Aurora MySQL dan Aurora PostgreSQL | Persentase |
| CommitLatency | Durasi rata-rata yang diperlukan oleh mesin dan penyimpanan untuk menyelesaikan operasi commit. | Aurora MySQL dan Aurora PostgreSQL | Milidetik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `CommitThroughput` | Rata-rata jumlah operasi commit per detik. | Aurora MySQL dan Aurora PostgreSQL | Hitungan per detik |
| `ConnectionAttempts` | Jumlah percobaan untuk terhubung ke sebuah instans, baik berhasil maupun tidak. | Aurora MySQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `CPUCreditBalance` | Jumlah kredit CPU yang diakumulasikan oleh satu instans, yang dilaporkan pada interval 5 menit. Anda dapat menggunakan metrik ini untuk menentukan berapa lama instans DB dapat melakukan burst melampaui tingkat performa dasarnya pada tingkat tertentu.<br><br>Metrik ini hanya berlaku untuk kelas instance ini:<br><br>• Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3`, dan `db.t4g`<br>• Aurora PostgreSQL: `db.t3` dan `db.t4g`<br><br>**Note**<br>Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans | Aurora MySQL dan Aurora PostgreSQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | T, lihat Jenis kelas instans DB.<br><br>Kredit peluncuran bekerja dengan cara yang sama di Amazon RDS seperti yang mereka lakukan di Amazon EC2. Untuk informasi selengkapnya, lihat Kredit peluncuran dalam Panduan Pengguna Amazon Elastic Compute Cloud untuk Instans Linux. | | |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `CPUCreditUsage` | Jumlah CPU yang digunakan selama periode tertentu, yang dilaporkan pada interval 5 menit. Metrik ini mengukur jumlah waktu selama fisik CPUs telah digunakan untuk memproses instruksi oleh virtual yang CPUs dialokasi kan ke instans DB.<br><br>Metrik ini hanya berlaku untuk kelas instance ini:<br><br>• Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3`, dan `db.t4g`<br>• Aurora PostgreSQL: `db.t3` dan `db.t4g`<br><br>**Note**<br>Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans | Aurora MySQL dan Aurora PostgreSQL | Hitungan |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| | T, lihat Jenis kelas instans DB. | | |
| `CPUSurplusCreditBalance` | Jumlah kredit surplus yang telah digunakan oleh sebuah instans tak terbatas ketika nilai `CPUCreditBalance` -nya nol.<br><br>Nilai `CPUSurplusCreditBalance` ditutupi dengan kredit CPU yang diperoleh. Jika jumlah kredit surplus melebihi jumlah kredit maksimum yang dapat diperoleh instans dalam jangka waktu 24 jam, kredit surplus yang digunakan di atas jumlah maksimum akan dikenai biaya tambahan.<br><br>Metrik kredit CPU hanya tersedia dalam frekuensi 5 menit. | Aurora MySQL dan Aurora PostgreSQL | Kredit (vCPU-menit) |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `CPUSurplusCreditsCharged` | Jumlah kredit surplus yang digunakan yang tidak ditutupi oleh kredit CPU yang diperoleh, sehingga menimbulkan biaya tambahan.<br><br>Kredit surplus yang digunakan akan dikenai biaya jika salah satu dari hal berikut terjadi:<br><br>• Kredit surplus yang digunakan melampaui jumlah kredit maksimum yang bisa didapatkan oleh instans dalam periode 24 jam. Kredit surplus yang digunakan di atas jumlah maksimum akan dikenai biaya pada akhir jam.<br>• Instans dihentikan atau diakhiri.<br>• instans dialihkan dari `unlimited` ke `standard`.<br><br>Metrik kredit CPU hanya tersedia dalam frekuensi 5 menit. | Aurora MySQL dan Aurora PostgreSQL | Kredit (vCPU-menit) |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `CPUUtilization` | Persentase CPU yang digunakan oleh instans DB Aurora. | Aurora MySQL dan Aurora PostgreSQL | Persentase |
| `DatabaseConnections` | Jumlah koneksi jaringan klien ke instans basis data.<br><br>Jumlah sesi basis data bisa lebih tinggi dari nilai metrik karena nilai metrik tidak termasuk yang berikut:<br><br>• Sesi yang tidak lagi memiliki koneksi jaringan, tetapi belum dibersihkan oleh basis data<br>• Sesi yang dibuat oleh mesin basis data untuk tujuannya sendiri<br>• Sesi yang dibuat oleh kemampuan eksekusi paralel mesin basis data<br>• Sesi yang dibuat oleh penjadwal pekerjaan mesin basis data<br>• Koneksi Amazon Aurora | Aurora MySQL dan Aurora PostgreSQL | Hitungan |
| `DDLLatency` | Durasi rata-rata permintaan seperti contoh, membuat, mengubah, dan membatalkan permintaan. | Aurora MySQL | Milidetik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| DDLThroughput | Jumlah rata-rata permintaan DDL per detik. | Aurora MySQL | Hitungan per detik |
| Deadlocks | Jumlah rata-rata deadlock di basis data per detik. | Aurora MySQL dan Aurora PostgreSQL | Hitungan per detik |
| DeleteLatency | Durasi rata-rata operasi penghapusan. | Aurora MySQL | Milidetik |
| DeleteThroughput | Jumlah rata-rata kueri penghapusan per detik. | Aurora MySQL | Hitungan per detik |
| DiskQueueDepth | Jumlah read/write permintaan yang belum selesai menunggu untuk mengakses disk. | Aurora MySQL dan Aurora PostgreSQL | Hitungan |
| DMLLatency | Durasi rata-rata penyisipan, pembaruan, dan penghapusan. | Aurora MySQL | Milidetik |
| DMLThroughput | Jumlah rata-rata penyisipan, pembaruan, dan penghapusan per detik. | Aurora MySQL | Hitungan per detik |
| EngineUptime | Jumlah waktu yang digunakan untuk menjalankan instans. | Aurora MySQL dan Aurora PostgreSQL | Detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `FreeableMemory` | Jumlah memori akses acak yang tersedia. Untuk database Aurora MySQL dan Aurora PostgreSQL, metrik ini melaporkan nilai bidang. `MemAvailable` `/proc/meminfo` | Aurora MySQL dan Aurora PostgreSQL | Byte |
| `FreeEphemeralStorage` | Jumlah penyimpanan Ephemeral NVMe yang tersedia. | Aurora PostgreSQL | Byte |
| `FreeLocalStorage` | Jumlah penyimpanan lokal yang tersedia.<br><br>Berbeda dengan mesin DB lainnya, untuk instans DB Aurora, metrik ini melaporkan jumlah penyimpanan yang tersedia untuk setiap instans DB. Nilai ini bergantung pada kelas instans DB (untuk informasi harga, lihat halaman Harga Amazon RDS). Anda dapat meningkatkan jumlah ruang penyimpanan gratis untuk sebuah instans dengan memilih kelas instans DB yang lebih besar untuk instans Anda.<br><br>(Hal ini tidak berlaku untuk Aurora Serverless v2.) | Aurora MySQL dan Aurora PostgreSQL | Byte |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `InsertLatency` | Durasi rata-rata operasi penyisipan. | Aurora MySQL | Milidetik |
| `InsertThroughput` | Jumlah rata-rata operasi penyisipan per detik. | Aurora MySQL | Hitungan per detik |
| `LoginFailures` | Jumlah rata-rata percobaan masuk gagal per detik. | Aurora MySQL | Hitungan per detik |
| `MaximumUsedTransactionIDs` | Umur ID transaksi tertua yang belum divakum, dalam transaksi. Jika nilai ini mencapai 2.146.483 .648 (2^31 - 1.000.000), basis data akan dipaksa ke mode hanya baca untuk menghindari wraparound ID transaksi. Untuk informasi selengkapnya, lihat Preventing transaction ID wraparound failures dalam dokumentasi PostgreSQL. | Aurora PostgreSQL | Hitungan |
| `NetworkReceiveThroughput` | Jumlah hasil jaringan yang diterima dari klien oleh setiap instans di klaster DB Aurora . Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster Aurora. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik (konsol menunjukkan Megabyte per detik) |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| NetworkThroughput | Jumlah hasil jaringan yang diterima dari dan dikirim ke klien oleh setiap instans di klaster DB Aurora . Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster Aurora. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |
| NetworkTransmitThroughput | Jumlah throughput jaringan yang dikirim ke klien oleh setiap instans dalam klaster DB Aurora. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik (konsol menunjukkan Megabyte per detik) |
| NumBinaryLogFiles | Jumlah file binlog yang dihasilkan. | Aurora MySQL | Hitungan |
| OldestReplicationSlotLag | Ukuran lagging replika paling lambat terkait data log write-ahead (WAL) yang diterima. | Aurora PostgreSQL | Byte |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `PurgeBoundary` | Nomor transaksi hingga pembersihan InnoDB diizinkan. Jika metrik ini tidak maju untuk jangka waktu yang lama, ini merupakan indikasi yang baik bahwa pembersihan InnoDB diblokir oleh transaksi yang berjalan lama. Untuk menyelidiki, periksa transaksi aktif pada cluster DB MySQL Aurora Anda. | Aurora MySQL versi 2, versi 2.11 dan lebih tinggi<br><br>Aurora MySQL versi 3, versi 3.08 dan lebih tinggi | Hitungan |
| `PurgeFinishedPoint` | Nomor transaksi hingga pembersihan InnoDB dilakukan. Metrik ini dapat membantu Anda memeriksa seberapa cepat proses pembersihan InnoDB. | Aurora MySQL versi 2, versi 2.11 dan lebih tinggi<br><br>Aurora MySQL versi 3, versi 3.08 dan lebih tinggi | Hitungan |
| `Queries` | Rata-rata jumlah kueri yang dijalankan per detik. | Aurora MySQL | Hitungan per detik |
| `RDSToAuroraPostgreSQLReplicaLag` | Lag saat mereplikasi pembaruan dari instans RDS PostgreSQL primer ke simpul lain dalam klaster. | Replika untuk Aurora PostgreSQL | Detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| ReadIOPS | Rata-rata jumlah I/O operasi disk per detik tetapi laporan membaca dan menulis secara terpisah, dalam interval 1 menit. | Aurora MySQL dan Aurora PostgreSQL | Hitungan per detik |
| ReadIOPSEphemeralStorage | Jumlah rata-rata I/O operasi membaca disk ke penyimpanan Ephemeral NVMe . | Aurora PostgreSQL | Hitungan per detik |
| ReadLatency | Jumlah rata-rata waktu yang dibutuhkan per I/O operasi disk. | Aurora MySQL dan Aurora PostgreSQL | Detik |
| ReadLatencyEphemeralStorage | Jumlah rata-rata waktu yang dibutuhkan per I/O operasi pembacaan disk untuk penyimpanan Ephemeral NVMe . | Aurora PostgreSQL | Milidetik |
| ReadThroughput | Jumlah byte rata-rata yang dibaca dari disk per detik. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |
| ReadThroughputEphemeralStorage | Jumlah rata-rata byte yang dibaca dari disk per detik untuk penyimpanan Ephemeral NVMe . | Aurora PostgreSQL | Byte per detik |
| ReplicationSlotDiskUsage | Jumlah ruang disk yang dikonsumsi oleh file slot replikasi. | Aurora PostgreSQL | Byte |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| ResultSetCacheHitRatio | Persentase permintaan yang dilayani oleh cache Resultset. | Aurora MySQL versi 2 | Persentase |
| RollbackSegmentHistoryListLength | Log undo yang mencatat transaksi yang dilakukan dengan catatan yang ditandai hapus. Catatan ini dijadwalkan untuk diproses oleh operasi purge InnoDB. | Aurora MySQL | Hitungan |
| RowLockTime | Total waktu yang dihabiskan guna mendapatkan kunci baris untuk tabel InnoDB. | Aurora MySQL | Milidetik |
| SelectLatency | Jumlah waktu rata-rata untuk operasi pemilihan. | Aurora MySQL | Milidetik |
| SelectThroughput | Jumlah rata-rata kueri pemilihan per detik. | Aurora MySQL | Hitungan per detik |
| ServerlessDatabaseCapacity | Kapasitas klaster DB Aurora Serverless saat ini. | Aurora MySQL dan Aurora PostgreSQL | Hitungan |
| StorageNetworkReceiveThroughput | Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instans di klaster DB. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `StorageNetworkThroughput` | Jumlah throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |
| `StorageNetworkTransmitThroughput` | Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |
| `SumBinaryLogSize` | Ukuran total file binlog. | Aurora MySQL | Byte |
| `SwapUsage` | Jumlah ruang swap yang digunakan. Metrik ini tidak tersedia untuk kelas instans DB berikut:<br>• db.r3.*, db.r4.*, dan db.r7g.* (Aurora MySQL)<br>• db.r7g.* (Aurora PostgreSQL) | Aurora MySQL dan Aurora PostgreSQL | Byte |
| `TempStorageIOPS` | Jumlah IOPS untuk baca dan tulis pada penyimpanan lokal yang dilampirkan ke instans DB. Metrik ini merepresentasikan hitungan dan diukur sekali per detik.<br>Metrik ini hanya berlaku untuk Aurora Serverless v2. | Aurora MySQL dan Aurora PostgreSQL | Hitungan per detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `TempStorageThroughput` | Jumlah data yang ditransfer ke dan dari penyimpanan lokal yang terkait dengan instans DB. Metrik ini merepresentasikan byte dan diukur sekali per detik.<br><br>Metrik ini hanya berlaku untuk Aurora Serverless v2. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |
| `TransactionAgeMaximum` | Usia transaksi berjalan aktif tertua. | Aurora MySQL versi 3, versi 3.08 dan lebih tinggi | Detik |
| `TransactionLogsDiskUsage` | Jumlah ruang disk yang digunakan oleh log transaksi di instans DB Aurora PostgreSQL.<br><br>Metrik ini dihasilkan hanya ketika Aurora PostgreSQL menggunakan replikasi logis atau. AWS Database Migration Service Secara default, Aurora PostgreSQL menggunakan catatan log, bukan log transaksi. Saat log transaksi tidak digunakan, nilai untuk metrik ini adalah `-1`. | Primer untuk Aurora PostgreSQL | Byte |

| Metrik | Deskripsi | Berlaku untuk | Unit |
| --- | --- | --- | --- |
| `TruncateFinishedPoint` | Pengidentifikasi transaksi hingga pemotongan undo dilakukan. | Aurora MySQL versi 2, versi 2.11 dan lebih tinggi<br><br>Aurora MySQL versi 3, versi 3.08 dan lebih tinggi | Hitungan |
| `UpdateLatency` | Jumlah waktu rata-rata yang digunakan untuk operasi pembaruan. | Aurora MySQL | Milidetik |
| `UpdateThroughput` | Jumlah rata-rata pembaruan per detik. | Aurora MySQL | Hitungan per detik |
| `WriteIOPS` | Jumlah catatan penulisan penyimpanan Aurora yang dihasilkan per detik. Metrik ini kurang lebih adalah jumlah catatan log yang dihasilkan oleh basis data. Metrik ini tidak sesuai dengan penulisan halaman 8K, dan tidak sesuai dengan paket jaringan yang dikirim. | Aurora MySQL dan Aurora PostgreSQL | Hitungan per detik |
| `WriteIOPSEphemeralStorage` | Jumlah rata-rata I/O operasi penulisan disk ke penyimpanan Ephemeral NVMe . | Aurora PostgreSQL | Hitungan per detik |

| Metrik | Deskripsi | Berlaku untuk | Unit |
|---|---|---|---|
| `WriteLatency` | Jumlah rata-rata waktu yang dibutuhkan per I/O operasi disk. | Aurora MySQL dan Aurora PostgreSQL | Detik |
| `WriteLatencyEphemeralStorage` | Jumlah rata-rata waktu yang dibutuhkan per I/O operasi penulisan disk untuk penyimpanan Ephemeral NVMe . | Aurora PostgreSQL | Milidetik |
| `WriteThroughput` | Jumlah rata-rata byte yang ditulis ke penyimpanan persisten setiap detik. | Aurora MySQL dan Aurora PostgreSQL | Byte per detik |
| `WriteThroughputEphemeralStorage` | Jumlah rata-rata byte yang ditulis ke disk per detik untuk penyimpanan Ephemeral NVMe . | Aurora PostgreSQL | Byte per detik |

## Metrik CloudWatch penggunaan Amazon untuk

`AWS/Usage`Namespace di Amazon CloudWatch menyertakan metrik penggunaan tingkat akun untuk kuota layanan Amazon RDS Anda. CloudWatch mengumpulkan metrik penggunaan secara otomatis untuk semua. Wilayah AWS

Untuk informasi selengkapnya, lihat metrik CloudWatch penggunaan di Panduan CloudWatch Pengguna Amazon. Untuk informasi selengkapnya tentang kuota, lihat Kuota dan batasan untuk Amazon Aurora dan Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

| Metrik | Deskripsi | Unit* |
| --- | --- | --- |
| `AuthorizationsPerDBSecurityGroup` | Jumlah aturan masuk per grup keamanan DB di grup Anda Akun AWS. Nilai yang digunakan adalah jumlah aturan masuk tertinggi dalam grup keamanan DB di akun. Grup keamanan DB lainnya di akun mungkin memiliki jumlah aturan masuk yang lebih rendah. | Hitungan |
| `CustomEndpointsPerDBCluster` | Jumlah titik akhir kustom per cluster DB di Anda Akun AWS. Nilai yang digunakan adalah jumlah titik akhir kustom tertinggi dalam cluster DB di akun. Cluster DB lain di akun mungkin memiliki jumlah titik akhir kustom yang lebih rendah. | Hitungan |
| `DBClusterParameterGroups` | Jumlah grup parameter klaster DB di Akun AWS Anda. Hitungan tidak termasuk grup parameter default. | Hitungan |
| `DBClusterRoles` | Jumlah peran terkait AWS Identity and Access Management (IAM) per kluster DB di kluster Anda Akun AWS. Nilai yang digunakan adalah jumlah tertinggi peran IAM terkait untuk cluster DB di akun. Cluster DB lain di akun mungkin memiliki jumlah peran IAM terkait yang lebih rendah. | Hitungan |
| `DBClusters` | Jumlah klaster DB Amazon Aurora di Akun AWS Anda. | Hitungan |
| `DBInstanceRoles` | Jumlah peran terkait AWS Identity and Access Management (IAM) per instans DB di Anda Akun AWS. Nilai yang digunakan adalah jumlah tertinggi peran IAM terkait untuk instans DB di akun. Instans DB lain di akun mungkin memiliki jumlah peran IAM terkait yang lebih rendah. | Hitungan |
| `DBInstances` | Jumlah instans DB di Akun AWS Anda. | Hitungan |
| `DBParameterGroups` | Jumlah grup parameter DB di Akun AWS Anda. Hitungan tidak termasuk grup parameter DB default. | Hitungan |

| Metrik | Deskripsi | Unit* |
| --- | --- | --- |
| DBSubnetGroups | Jumlah grup subnet DB di Akun AWS Anda. Hitungan tidak termasuk grup subnet default. | Hitungan |
| EventSubscriptions | Jumlah langganan pemberitahuan acara di Anda Akun AWS. | Hitungan |
| Integrations | Jumlah integrasi nol-ETL dengan Amazon Redshift di Anda. Akun AWS | Hitungan |
| ManualClusterSnapshots | Jumlah snapshot klaster DB yang dibuat secara manual di Akun AWS Anda. Hitungan tidak termasuk snapshot yang tidak valid. | Hitungan |
| OptionGroups | Jumlah grup opsi di Akun AWS Anda. Hitungan tidak termasuk grup opsi default. | Hitungan |
| Proxies | Jumlah proxy RDS di akun Anda. AWS | Hitungan |
| ReadReplicasPerMaster | Jumlah replika baca per instans DB di akun Anda. Nilai yang digunakan adalah jumlah replika baca tertinggi untuk instance DB di akun. Instans DB lain di akun mungkin memiliki jumlah replika baca yang lebih rendah. | Hitungan |
| ReservedDBInstances | Jumlah instans DB yang dialokasikan di Akun AWS Anda. Hitungan tidak termasuk instans yang dipensiunkan atau ditolak. | Hitungan |
| SubnetsPerDBSubnetGroup | Jumlah subnet per grup subnet DB di Anda. Akun AWS Jumlah subnet tertinggi untuk grup subnet DB di akun. Grup subnet DB lainnya di akun mungkin memiliki jumlah subnet yang lebih rendah. | Hitungan |

> **Note**
>
> Amazon RDS tidak memublikasikan unit untuk metrik penggunaan. CloudWatch Unit hanya muncul dalam dokumentasi.

## CloudWatch Dimensi Amazon untuk Aurora

Anda dapat memfilter data metrik Aurora dengan menggunakan dimensi apa pun dalam tabel berikut.

| Dimensi | Memfilter data yang diminta terhadap . . . |
|---|---|
| `DBInstanceIdentifier` | Instans basis data tertentu. |
| `DBClusterIdentifier` | Klaster basis data Aurora tertentu. |
| `DBClusterIdentifier, Role` | Cluster Aurora DB tertentu, menggabungkan metrik dengan peran instance (/). WRITER READER Misalnya, Anda dapat menggabungkan metrik untuk semua READER instance milik klaster. |
| `DbClusterIdentifier, EngineName` | Kombinasi nama klaster dan mesin basis data Aurora tertentu. Misalnya, Anda dapat menampilkan metrik `VolumeReadIOPs` untuk klaster `ams1` dan mesin `aurora`. |
| `DatabaseClass` | Semua instans dalam kelas basis data. Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans kelas basis data `db.r5.large` . |
| `EngineName` | Hanya nama mesin diidentifikasi. Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans yang memiliki nama mesin `aurora-postgresql` . |
| `SourceRegion` | Hanya Wilayah yang ditentukan. Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans basis data dalam Wilayah `us-east-1` . |

# Ketersediaan metrik Aurora di konsol Amazon RDS

Tidak semua metrik yang disediakan oleh Amazon Aurora tersedia di konsol AmazonRDS. Anda dapat melihat metrik ini menggunakan alat seperti AWS CLI dan CloudWatch API. Selain itu, beberapa metrik di RDS konsol Amazon ditampilkan hanya untuk kelas instance tertentu, atau dengan nama dan unit pengukuran yang berbeda.

Topik


- Metrik Aurora tersedia dalam tampilan Jam Terakhir
- Metrik Aurora yang tersedia dalam kasus tertentu
- Metrik Aurora yang tidak tersedia di konsol


## Metrik Aurora tersedia dalam tampilan Jam Terakhir

Anda dapat melihat subset metrik Aurora yang dikategorikan dalam tampilan Jam Terakhir default di konsol Amazon. RDS Tabel berikut mencantumkan kategori dan metrik terkait yang ditampilkan di RDS konsol Amazon untuk instance Aurora.

| Kategori | Metrik |
| --- | --- |
| SQL | `ActiveTransactions`<br>`BlockedTransactions`<br>`BufferCacheHitRatio`<br>`CommitLatency`<br>`CommitThroughput`<br>`DatabaseConnections`<br>`DDLLatency`<br>`DDLThroughput`<br>`Deadlocks`<br>`DMLLatency` |

| Kategori | Metrik |
| --- | --- |
| | `DMLThroughput`<br>`LoginFailures`<br>`ResultSetCacheHitRatio`<br>`SelectLatency`<br>`SelectThroughput` |
| Sistem | `AuroraReplicaLag`<br>`AuroraReplicaLagMaximum`<br>`AuroraReplicaLagMinimum`<br>`CPUCreditBalance`<br>`CPUCreditUsage`<br>`CPUUtilization`<br>`FreeableMemory`<br>`FreeLocalStorage` (Hal ini tidak berlaku untuk Aurora Serverless v2.)<br>`NetworkReceiveThroughput` |
| Deployment | `AuroraReplicaLag`<br>`BufferCacheHitRatio`<br>`ResultSetCacheHitRatio`<br>`SelectThroughput` |

## Metrik Aurora yang tersedia dalam kasus tertentu

Selain itu, beberapa metrik Aurora ditunjukkan hanya untuk kelas instans tertentu, atau hanya untuk instans DB, atau dengan nama dan unit ukur yang berbeda-beda:

- `CPUCreditUsage`Metrik `CPUCreditBalance` dan ditampilkan hanya untuk kelas Aurora SQL `db.t2` My instance dan untuk kelas instance Aurora SQL `db.t3` Postgre.
- Metrik berikut ini ditampilkan dengan nama yang berbeda-beda, seperti yang tercantum:

| Metrik | Nama tampilan |
| --- | --- |
| `AuroraReplicaLagMaximum` | Lag replika maksimum |
| `AuroraReplicaLagMinimum` | Lag replika minimum |
| `DDLThroughput` | DDL |
| `NetworkReceiveThroughput` | Throughput jaringan |
| `VolumeBytesUsed` | [Ditagih] Byte volume yang digunakan |
| `VolumeReadIOPs` | [Ditagih] Pembacaan volume IOPS |
| `VolumeWriteIOPs` | [Ditagih] Volume IOPS |

- Metrik berikut berlaku untuk seluruh cluster Aurora DB, tetapi hanya ditampilkan saat melihat instans DB untuk cluster Aurora DB di konsol Amazon: RDS
  - `VolumeBytesUsed`
  - `VolumeReadIOPs`
  - `VolumeWriteIOPs`
- Metrik berikut ditampilkan dalam megabyte, bukan byte, di konsol Amazon: RDS
  - `FreeableMemory`
  - `FreeLocalStorage`
  - `NetworkReceiveThroughput`
  - `NetworkTransmitThroughput`
- Metrik berikut berlaku untuk cluster Aurora SQL Postgre DB dengan Aurora Optimized Reads:
  - `AuroraOptimizedReadsCacheHitRatio`

- FreeEphemeralStorage
- ReadIOPSEphemeralStorage
- ReadLatencyEphemeralStorage
- ReadThroughputEphemeralStorage
- WriteIOPSEphemeralStorage
- WriteLatencyEphemeralStorage
- WriteThroughputEphemeralStorage

### Metrik Aurora yang tidak tersedia di konsol

Metrik Aurora berikut tidak tersedia di konsol Amazon: RDS

- AuroraBinlogReplicaLag
- DeleteLatency
- DeleteThroughput
- EngineUptime
- InsertLatency
- InsertThroughput
- NetworkThroughput
- Queries
- UpdateLatency
- UpdateThroughput

## CloudWatch Metrik Amazon untuk Amazon RDS Performance Insights

Performance Insights secara otomatis menerbitkan beberapa metrik ke Amazon. CloudWatch Data yang sama dapat ditanyakan dari Performance Insights, tetapi memiliki metrik CloudWatch memudahkan untuk menambahkan alarm. CloudWatch Ini juga memudahkan untuk menambahkan metrik ke CloudWatch Dasbor yang ada.

| Metrik | Deskripsi |
| --- | --- |
| DBLoad | Jumlah sesi aktif untuk database. Biasanya, Anda menginginkan data untuk jumlah rata-rata sesi aktif. Dalam Wawasan Performa, data ini dikueri sebagai `db.load.avg` . |
| DBLoadCPU | Jumlah sesi aktif dengan jenis peristiwa tunggu berupa CPU. Dalam Wawasan Performa, data ini dikueri sebagai `db.load.avg` , difilter berdasarkan jenis peristiwa tunggu `CPU`. |
| DBLoadNonCPU | Jumlah sesi aktif dengan jenis peristiwa tunggu bukan CPU. |
| DBLoadRelativeToNumVCPUs | Rasio beban DB dengan jumlah virtual CPUs untuk database. |

> **Note**
>
> Metrik ini dipublikasikan CloudWatch hanya jika ada beban pada instans DB.

Anda dapat memeriksa metrik ini menggunakan CloudWatch konsol, the AWS CLI, atau CloudWatch API. Anda juga dapat memeriksa metrik penghitung Performance Insights lainnya menggunakan fungsi matematika metrik khusus. Untuk informasi selengkapnya, lihat Menanyakan metrik penghitung Performance Insights lainnya di CloudWatch.

Misalnya, Anda bisa mendapatkan statistik untuk `DBLoad` metrik dengan menjalankan get-metric-statisticsperintah.

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
```

```
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

Contoh ini menghasilkan output yang terlihat seperti berikut.

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
  "Unit": "None",
  "Average": 1.5
  },
  {
  "Timestamp": "2021-07-19T21:32:00Z",
  "Unit": "None",
  "Average": 1.8
  },
  {
  "Timestamp": "2021-07-19T21:29:00Z",
  "Unit": "None",
  "Average": 3.0
  },
  {
  "Timestamp": "2021-07-19T21:33:00Z",
  "Unit": "None",
  "Average": 2.4
  }
  ],
```

```
  "Label": "DBLoad"
  }
```

Untuk informasi selengkapnya CloudWatch, lihat Apa itu Amazon CloudWatch? di Panduan CloudWatch Pengguna Amazon.

### Menanyakan metrik penghitung Performance Insights lainnya di CloudWatch

> **Note**
>
> Jika Anda mengaktifkan mode Advanced Database Insights, Amazon RDS menerbitkan metrik penghitung Performance Insights ke Amazon. CloudWatch Dengan Database Insights, Anda tidak perlu menggunakan fungsi matematika `DB_PERF_INSIGHTS` metrik. Anda dapat menggunakan dasbor CloudWatch Database Insights untuk mencari, menanyakan, dan menyetel alarm untuk metrik penghitung Performance Insights.

Anda dapat melakukan kueri, alarm, dan grafik pada metrik Performance Insights RDS. CloudWatch Anda dapat mengakses informasi tentang Anda dengan menggunakan fungsi matematika `DB_PERF_INSIGHTS` metrik untuk CloudWatch. Fungsi ini memungkinkan Anda menggunakan metrik Performance Insights yang tidak dilaporkan secara langsung CloudWatch untuk membuat deret waktu baru.

Anda dapat menggunakan fungsi Matematika Metrik baru dengan mengklik menu tarik-turun Tambahkan Matematika di layar Select metric di CloudWatch konsol. Anda dapat menggunakannya untuk membuat alarm dan grafik pada metrik Performance Insights atau pada kombinasi dan metrik CloudWatch Performance Insights, termasuk alarm resolusi tinggi untuk metrik sub-menit. Anda juga dapat menggunakan fungsi secara terprogram dengan menyertakan ekspresi Matematika Metrik dalam permintaan. `get-metric-data` Untuk informasi selengkapnya, lihat Sintaks dan fungsi matematika metrik dan Membuat alarm pada metrik penghitung Performance Insights dari database. AWS

## Metrik penghitung Wawasan Performa

Metrik penghitung adalah metrik performa sistem operasi dan basis data di dasbor Wawasan Performa. Untuk membantu mengidentifikasi dan menganalisis masalah performa, Anda dapat mengaitkan metrik penghitung dengan muatan DB. Anda harus menambahkan fungsi

statistik ke metrik untuk mendapatkan nilai metrik. Misalnya, fungsi yang didukung untuk metrik `os.memory.active` adalah `.avg`, `.min`, `.max`, `.sum`, dan `.sample_count`.

Metrik penghitung dikumpulkan sekali setiap menit. Kumpulan metrik OS bergantung pada apakah Pemantauan yang Ditingkatkan diaktifkan atau dinonaktifkan. Jika Pemantauan yang Ditingkatkan dinonaktifkan, metrik OS dikumpulkan sekali setiap menit. Jika Pemantauan yang Ditingkatkan diaktifkan, metrik OS dikumpulkan untuk periode waktu yang dipilih. Untuk informasi selengkapnya tentang cara mengaktifkan atau menonaktifkan Pemantauan yang Ditingkatkan, lihat Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan.

Topik

- Penghitung sistem operasi Wawasan Performa
- Penghitung Wawasan Performa untuk Aurora MySQL
- Penghitung Wawasan Performa untuk Aurora PostgreSQL

## Penghitung sistem operasi Wawasan Performa

Penghitung sistem operasi berikut, yang diawali dengan `os`, tersedia di Wawasan Performa untuk Aurora PostgreSQL dan Aurora MySQL.

Anda dapat menggunakan API `ListAvailableResourceMetrics` untuk mengetahui daftar metrik penghitung yang tersedia untuk instans DB Anda. Untuk informasi selengkapnya, lihat ListAvailableResourceMetricsdi panduan Referensi API Amazon RDS Performance Insights.

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Aktif | Memori | Kilobyte | os.memory .active | Jumlah memori yang ditetapkan, dalam kilobyte. |
| Buffer | Memori | Kilobyte | os.memory .buffers | Jumlah memori yang digunakan untuk I/O permintaa n buffering sebelum menulis ke perangkat |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| | | | | penyimpanan, dalam kilobyte. |
| Di-cache | Memori | Kilobyte | os.memory .cached | Jumlah memori yang digunakan untuk meng-cach e I/O berbasis sistem file, dalam kilobyte. |
| Cache DB | Memori | Byte | os.memory .db.cache | Jumlah memori yang digunakan untuk cache halaman oleh proses basis data termasuk tmpfs (shmem), dalam byte. |
| Ukuran Set Residen DB | Memori | Byte | os.memory.db. residentSetSize | Jumlah memori yang digunakan untuk cache anonim dan swap oleh proses basis data tidak termasuk tmpfs (shmem), dalam byte. |
| Swap DB | Memori | Byte | os.memory .db.swap | Jumlah memori yang digunakan untuk swap berdasarkan proses basis data, dalam byte. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Kotor | Memori | Kilobyte | os.memory.dirty | Jumlah halaman memori dalam RAM yang telah diubah, tetapi tidak ditulis ke blok data terkaitnya dalam penyimpanan, dalam kilobyte. |
| Kosong | Memori | Kilobyte | os.memory.free | Jumlah memori yang tidak ditetapkan, dalam kilobyte. |
| Halaman Besar Kosong | Memori | Halaman | os.memori. hugePagesFree | Jumlah halaman besar yang kosong. Halaman besar adalah fitur dari kernel Linux. |
| Rsvd Halaman Besar | Memori | Halaman | os.memori. hugePagesRsvd | Jumlah halaman besar yang khusus. |
| Ukuran Halaman Besar | Memori | Kilobyte | os.memori. hugePagesSize | Ukuran untuk setiap unit halaman besar, dalam kilobyte. |
| Surplus Halaman Besar | Memori | Halaman | os.memori. hugePagesSurp | Jumlah halaman besar surplus yang tersedia terhadap total. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Total Halaman Besar | Memori | Halaman | os.memori. hugePagesTotal | Jumlah total halaman besar. |
| Nonaktif | Memori | Kilobyte | os.memory .inactive | Jumlah halaman memori yang jarang digunakan , dalam kilobyte. |
| Dipetakan | Memori | Kilobyte | os.memory .mapped | Jumlah total konten sistem file yang memorinya dipetakan di dalam ruang alamat proses, dalam kilobyte. |
| Jumlah Kill Kehabisan Memori | Memori | Membunuh | os.memori. outOfMemo ryKillCount | Jumlah kill OOM yang terjadi selama interval pengumpulan terakhir. |
| Tabel Halaman | Memori | Kilobyte | os.memory .pageTables | Jumlah memori yang digunakan berdasarkan tabel halaman, dalam kilobyte. |
| Lempengan | Memori | Kilobyte | os.memory.slab | Jumlah struktur data kernel yang dapat digunakan kembali, dalam kilobyte. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Total | Memori | Kilobyte | os.memory.total | Jumlah total memori, dalam kilobyte. |
| Writeback | Memori | Kilobyte | os.memory .writeback | Jumlah halaman kotor dalam RAM yang masih ditulis ke penyimpanan cadangan, dalam kilobyte. |
| Tamu | Penggunaan CPU | Persentase | os.cpuUtilization. guest | Persentase CPU yang digunakan oleh program tamu. |
| Idle | Penggunaan CPU | Persentase | os.cpuUtilization. idle | Persentase CPU saat idle. |
| Irq | Penggunaan CPU | Persentase | os.cpuUtilization. irq | Persentase CPU yang digunakan oleh gangguan perangkat lunak. |
| Nice | Penggunaan CPU | Persentase | os.cpuUtilization. nice | Persentase CPU yang digunakan oleh program yang berjalan pada prioritas terendah. |
| Steal | Penggunaan CPU | Persentase | os.cpuUtilization. steal | Persentase CPU yang digunakan oleh mesin virtual lainnya. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Sistem | Penggunaan CPU | Persentase | os.cpuUtilization.system | Persentase CPU yang digunakan oleh kernel. |
| Total | Penggunaan CPU | Persentase | os.cpuUtilization.total | Total persentase CPU yang digunakan. Nilai ini mencakup nilai nice. |
| Pengguna | Penggunaan CPU | Persentase | os.cpuUtilization.user | Persentase CPU yang digunakan oleh program pengguna. |
| Tunggu | Penggunaan CPU | Persentase | os.cpuUtilization.wait | Persentase CPU yang tidak digunakan saat menunggu I/O akses. |
| Penyimpanan Aurora Penyimpanan Aurora Byte Rx | IO Disk | Byte per detik | os.Diskio.AuroraStorage.auroraStorageBytesRx | Jumlah byte yang diterima dari penyimpanan Aurora per detik. |
| Penyimpanan Aurora Penyimpanan Aurora Byte Tx | IO Disk | Byte per detik | os.Diskio.AuroraStorage.auroraStorageBytesTx | Jumlah byte yang diunggah ke penyimpanan aurora per detik. |
| Kedalaman Antrean Disk Penyimpanan Aurora | IO Disk | Permintaan | os.Diskio.AuroraStorage.diskQueueDepth | Panjang antrian disk penyimpanan Aurora. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Penyimpanan Aurora Baca PS IOs | IO Disk | Permintaan per detik | os.diskio .aurorastorage.Baca PS IOs | Jumlah operasi baca per detik. |
| Latensi Baca Penyimpanan Aurora | IO Disk | Milidetik | os.diskIO .auroraStorage.readLatency | Latensi rata-rata I/O permintaan baca ke penyimpanan Aurora, dalam milidetik. |
| Throughput Baca Penyimpanan Aurora | IO Disk | Byte per detik | os.diskIO .auroraStorage.readThroughput | Jumlah throughput jaringan yang digunakan oleh permintaan ke klaster DB, dalam byte per detik. |
| Penyimpanan Aurora Tulis PS IOs | IO Disk | Permintaan per detik | os.diskio .aurorastorage.tulis PS IOs | Jumlah operasi tulis per detik. |
| Latensi Tulis Penyimpanan Aurora | IO Disk | Milidetik | os.diskIO .auroraStorage.writeLatency | Latensi rata-rata I/O permintaan tulis ke penyimpanan Aurora, dalam milidetik. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Throughput Tulis Penyimpanan Aurora | IO Disk | Byte per detik | os.diskIO .auroraSt orage.wri teThroughput | Jumlah throughput jaringan yang digunakan oleh respons dari klaster DB, dalam byte per detik. |
| Len Antrean Rata-rata Rdstemp | IO Disk | Permintaan | os.Diskio .RDStemp. avgQueueLen | Jumlah permintaan yang menunggu dalam antrian I/O perangkat. |
| Sz Req Rata-rata Rdstemp | IO Disk | Permintaan | os.Diskio .RDStemp. avgReqSz | Jumlah permintaan yang menunggu dalam antrian I/O perangkat. |
| Menunggu Rdstemp | IO Disk | Milidetik | os.diskIO .rdstemp.await | Jumlah milidetik yang diperlukan untuk merespons permintaan, termasuk waktu antrean dan waktu layanan. |
| Rdstemp Baca PS IOs | IO Disk | Permintaan | os.diskio .rdstemp.Baca PS IOs | Jumlah operasi baca per detik. |
| KB Baca Rdstemp | IO Disk | Kilobyte | os.diskIO .rdstemp.readKb | Jumlah total kilobyte yang dibaca. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| PS KB Baca Rdstemp | IO Disk | Kilobyte per detik | os.diskIO .rdstemp. readKbPS | Jumlah kilobyte yang dibaca per detik. |
| PS Rrqm Rdstemp | IO Disk | Permintaan per detik | os.diskIO .rdstemp.rrqmPS | Jumlah permintaan baca gabungan yang diantrekan per detik. |
| TPS Rdstemp | IO Disk | Transaksi per detik | os.diskIO .rdstemp.tps | Jumlah I/O transaksi per detik. |
| Utilitas Rdstemp | IO Disk | Persentase | os.diskIO .rdstemp.util | Persentase waktu CPU saat permintaan dikeluarkan. |
| Rdstemp Tulis PS IOs | IO Disk | Permintaan per detik | os.diskio .rdstemp.tulis PS IOs | Jumlah operasi tulis per detik. |
| KB Tulis Rdstemp | IO Disk | Kilobyte | os.diskIO .rdstemp.writeKb | Jumlah total kilobyte yang ditulis. |
| PS KB Tulis Rdstemp | IO Disk | Kilobyte per detik | os.diskIO .rdstemp. writeKbPS | Jumlah kilobyte yang ditulis per detik. |
| PS Wrqm Rdstemp | IO Disk | Permintaan per detik | os.diskIO .rdstemp. wrqmPS | Jumlah permintaan tulis gabungan yang diantrekan per detik. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Diblokir | Tugas | Tugas | os.tasks.blocked | Jumlah tugas yang diblokir. |
| Berjalan | Tugas | Tugas | os.tasks.running | Jumlah tugas yang berjalan. |
| Tidur | Tugas | Tugas | os.tasks.sleeping | Jumlah tugas yang tidur. |
| Dihentikan | Tugas | Tugas | os.tasks.stopped | Jumlah tugas yang dihentikan. |
| Total | Tugas | Tugas | os.tasks.total | Jumlah total tugas. |
| Zombie | Tugas | Tugas | os.tasks.zombie | Jumlah tugas turunan yang tidak aktif dengan tugas induk aktif. |
| Satu | Menit Rata-Rata Muatan | Proses | os. loadAvera geMinute.satu | Jumlah proses yang meminta waktu CPU selama satu menit terakhir. |
| Lima belas | Menit Rata-Rata Muatan | Proses | os. loadAvera geMinute.lima belas | Jumlah proses yang meminta waktu CPU selama 15 menit terakhir. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Lima | Menit Rata-Rata Muatan | Proses | os. loadAvera geMinute.lima | Jumlah proses yang meminta waktu CPU selama 5 menit terakhir. |
| Di-cache | Swap | Kilobyte | os.swap.cached | Jumlah memori swap, dalam kilobyte, yang digunakan sebagai memori cache. |
| Kosong | Swap | Kilobyte | os.swap.free | Jumlah memori swap yang kosong, dalam kilobyte. |
| Masuk | Swap | Kilobyte | os.swap.in | Jumlah memori, dalam kilobyte, yang ditukar ke dalam dari disk. |
| Keluar | Swap | Kilobyte | os.swap.out | Jumlah memori, dalam kilobyte, yang ditukar ke luar dari disk. |
| Total | Swap | Kilobyte | os.swap.total | Jumlah total memori swap yang tersedia dalam kilobyte. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| File Maks | Sistem File | Berkas | os.fileSys.maxFiles | Jumlah maksimum file yang dapat dibuat untuk sistem file. |
| File yang Digunakan | Sistem File | Berkas | os.fileSys.usedFiles | Jumlah file dalam sistem file. |
| Persen File Digunakan | Sistem File | Berkas | os.Filesys.usedFilePercent | Persentase file tersedia yang digunakan. |
| Persen Digunakan | Sistem File | Persentase | os.fileSys.usedPercent | Persentase ruang disk sistem file yang digunakan. |
| Digunakan | Sistem File | Kilobyte | os.fileSys.used | Jumlah ruang disk yang digunakan oleh file dalam sistem file, dalam kilobyte. |
| Total | Sistem File | Kilobyte | os.fileSys.total | Jumlah total ruang disk yang tersedia untuk sistem file, dalam kilobyte. |
| Rx | Jaringan | Byte per detik | os.network.rx | Jumlah byte yang diterima per detik. |

| Penghitung | Jenis | Unit | Metrik | Deskripsi |
|---|---|---|---|---|
| Tx | Jaringan | Byte per detik | os.network.tx | Jumlah byte yang diunggah per detik. |
| Penggunaan Acu | Umum | Persentase | os.genera l.acuUtilization | Persentase kapasitas saat ini dari kapasitas maksimum yang dikonfigurasi. |
| Acu Maks yang Dikonfigurasi | Umum | ACUs | os.umum. maxConfig uredAcu | Kapasitas maksimum yang dikonfigurasi oleh pengguna, dalam unit kapasitas Aurora ()ACUs. |
| Acu Min yang Dikonfigurasi | Umum | ACUs | os.umum. minConfig uredAcu | Kapasitas minimum yang dikonfigurasi oleh pengguna, di ACUs. |
| Bil VCPUs | Umum | v CPUs | os.general.num VCPUs | Jumlah virtual CPUs (vCPUs) untuk instance DB. |
| Kapasitas Basis Data Nirserver | Umum | ACUs | os.umum. serverles sDatabase Capacity | Kapasitas saat ini dari instance, di ACUs. |

# Penghitung Wawasan Performa untuk Aurora MySQL

Penghitung basis data berikut ini tersedia di Wawasan Performa untuk Aurora MySQL.

Topik


- Penghitung native untuk Aurora MySQL
- Penghitung non-native untuk Aurora MySQL


## Penghitung native untuk Aurora MySQL

Metrik native ditentukan oleh mesin basis data, bukan Amazon Aurora. Anda dapat menemukan definisi metrik native ini di Server status variables dalam dokumentasi MySQL.

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| Com_analyze | SQL | Kueri per detik | db.SQL.Com_analyze |
| Com_optimize | SQL | Kueri per detik | db.SQL.Com_optimize |
| Com_select | SQL | Kueri per detik | db.SQL.Com_select |
| Innodb_rows_deleted | SQL | Baris per detik | db.SQL.Innodb_rows_deleted |
| Innodb_rows_inserted | SQL | Baris per detik | db.SQL.Innodb_rows_inserted |
| Innodb_rows_read | SQL | Baris per detik | db.SQL.Innodb_rows_read |
| Innodb_rows_updated | SQL | Baris per detik | db.SQL.Innodb_rows_updated |
| Kueri | SQL | Kueri per detik | db.SQL.Queries |

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| Pertanyaan | SQL | Kueri per detik | db.SQL.Questions |
| Select_full_join | SQL | Kueri per detik | db.SQL.Select_full_join |
| Select_full_range_join | SQL | Kueri per detik | db.SQL.Select_full_range_join |
| Select_range | SQL | Kueri per detik | db.SQL.Select_range |
| Select_range_check | SQL | Kueri per detik | db.SQL.Select_range_check |
| Select_scan | SQL | Kueri per detik | db.SQL.Select_scan |
| Slow_queries | SQL | Kueri per detik | db.SQL.Slow_queries |
| Sort_merge_passes | SQL | Kueri per detik | db.SQL.Sort_merge_passes |
| Sort_range | SQL | Kueri per detik | db.SQL.Sort_range |
| Sort_rows | SQL | Kueri per detik | db.SQL.Sort_rows |
| Sort_scan | SQL | Kueri per detik | db.SQL.Sort_scan |
| Total_query_time | SQL | Milidetik | db.SQL.Total_query_time |
| Table_locks_immediate | Kunci | Permintaan per detik | db.Locks.Table_locks_immediate |

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| Table_locks_waited | Kunci | Permintaan per detik | db.Locks.Table_locks_waited |
| Innodb_row_lock_time | Kunci | Milidetik (rata-rata) | db.Locks.Innodb_row_lock_time |
| Aborted_clients | Pengguna | Koneksi | db.Users.Aborted_clients |
| Aborted_connects | Pengguna | Koneksi | db.Users.Aborted_connects |
| Koneksi | Pengguna | Koneksi | db.Users.Connections |
| External_threads_connected | Pengguna | Koneksi | db.Users.External_threads_connected |
| max_connections | Pengguna | Koneksi | DB.Users.Max_Connections |
| Threads_connected | Pengguna | Koneksi | db.Users.Threads_connected |
| Threads_created | Pengguna | Koneksi | db.Users.Threads_created |
| Threads_running | Pengguna | Koneksi | db.Users.Threads_running |
| Created_tmp_disk_tables | Temp | Tabel per detik | db.Temp.Created_tmp_disk_tables |
| Created_tmp_tables | Temp | Tabel per detik | db.Temp.Created_tmp_tables |
| Innodb_buffer_pool_pages_data | Cache | Halaman | db.Cache.Innodb_buffer_pool_pages_data |
| Innodb_buffer_pool_pages_total | Cache | Halaman | db.Cache.Innodb_buffer_pool_pages_total |
| Innodb_buffer_pool_read_requests | Cache | Halaman per detik | db.Cache.Innodb_buffer_pool_read_requests |
| Innodb_buffer_pool_reads | Cache | Halaman per detik | db.Cache.Innodb_buffer_pool_reads |

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| Opened_tables | Cache | Tabel | db.Cache.Opened_tables |
| Opened_table_definitions | Cache | Tabel | db.Cache.Opened_table_definitions |
| Qcache_hits | Cache | Kueri | db.Cache.Qcache_hits |

Penghitung non-native untuk Aurora MySQL

Metrik penghitung non-native adalah penghitung yang ditentukan oleh Amazon RDS. Metrik non-asli bisa berupa metrik yang Anda dapatkan dengan kueri tertentu. Metrik non-native juga bisa berupa metrik turunan, dengan dua penghitung native atau lebih digunakan dalam penghitungan untuk mengetahui rasio, tingkat hit, atau latensi.

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| active_transactions | Transa | db.Trar ctions.a tive_tra saction | Total transaksi aktif. | `SELECT COUNT(1) AS active_tr ansactions FROM INFOR MATION_SC HEMA.INNO DB_TRX` | |
| innodb_buffer_pool _hit_rate | Cache | db.Cac innoDB ffer_po _hit_rat | Jumlah baca yang dapat dipenuhi oleh InnoDB dari kumpulan buffer. | `100 * innodb_bu ffer_pool _read_req uests / (innodb_ buffer_po ol_read_r equests + innodb_bu ffer_pool _reads)` | |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| innodb_buffer_pool_hits | Cache | Halama per detik | db.Cache.innoDB_buffer_pool_hits | Jumlah bacaan yang dapat dipenuhi oleh InnoDB dari kumpulan buffer. | `innodb_buffer_pool_read_requests - innodb_buffer_pool_reads` |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| innodb_buffer_pool_usage | Cache | Persen e | db.Cache. innoDB_buffer_pool_usage | Persentase kumpulan buffer InnoDB yang berisi data (halaman).<br><br>**Note**<br>Saat menggunakan tabel terkompresi, nilai ini bisa bervariasi. Untuk informasi selengkapnya, lihat informasi tentang `Innodb_buffer_pool_pages_data` dan `Innodb_buffer_pool_pages_total` di Server status variables dalam dokumentasi MySQL. | `Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0` |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| innodb_deadlocks | Kunci | db.Lock innodb_ adlocks | Jumlah total penguncian. | `SELECT COUNT AS innodb_de adlocks FROM INFORMATION_SC HEMA.INNO DB_METRICS WHERE NAME= 'lock_dea dlocks'` | |
| innodb_lo ck_timeouts | Kunci | db.Lock innodb_ ck_time ts | Jumlah total penguncian yang habis waktunya. | `SELECT COUNT AS innodb_lo ck_timeouts FROM INFOR MATION_SC HEMA.INNO DB_METRICS WHERE NAME= 'lock_tim eouts'` | |
| innodb_ro w_lock_waits | Kunci | db.Lock innodb_ w_lock_ its | Jumlah total kunci baris yang menghasil kan peristiwa tunggu. | `SELECT COUNT AS innodb_ro w_lock_waits FROM INFOR MATION_SC HEMA.INNO DB_METRICS WHERE NAME ='lock_ro w_lock_waits'` | |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| innodb_rows_changed | SQL | db.SQL.Innodb_rows_changed | Total operasi baris InnoDB. | db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated | |
| query_cache_hit_rate | Cache | Persen | db.Cache.query_cache_hit_rate | Rasio hit untuk cache set hasil MySQL (cache kueri). | Qcache_hits / (QCache_hits + Com_select) * 100 |
| temp_disk_tables_percent | Temp | db.temp.temp_disk_tables_percent | Persentase tabel sementara yang dibuat pada disk oleh server saat menjalankan pernyataan. | (db.Temp.Created_tmp_disk_tables / db.Temp.Created_tmp_tables) * 100 | |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| trx_rseg_history_len | Transa | Tidak ada | db.Transactions.trx_rseg_history_len | Daftar halaman log undo untuk transaksi yang telah dijalankan yang dipertahankan oleh sistem transaksi InnoDB untuk menerapkan kontrol konkurensi multi-versi. Untuk informasi selengkapnya tentang detail data log undo, lihat https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html dalam dokumentasi MySQL. | `SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'` |

## Penghitung Wawasan Performa untuk Aurora PostgreSQL

Penghitung basis data berikut tersedia di Wawasan Performa untuk Aurora PostgreSQL.

Topik

- Penghitung asli untuk Aurora PostgreSQL
- Penghitung non-native untuk Aurora PostgreSQL

### Penghitung asli untuk Aurora PostgreSQL

Metrik native ditentukan oleh mesin basis data, bukan Amazon Aurora. Anda dapat menemukan definisi untuk metrik native ini dalam Viewing Statistics dalam dokumentasi PostgreSQL.

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| tup_deleted | SQL | Tuple per detik | db.SQL.tup_deleted |

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| tup_fetched | SQL | Tuple per detik | db.SQL.tup_fetched |
| tup_inserted | SQL | Tuple per detik | db.SQL.tup_inserted |
| tup_returned | SQL | Tuple per detik | db.SQL.tup_returned |
| tup_updated | SQL | Tuple per detik | db.SQL.tup_updated |
| blks_hit | Cache | Blok per detik | db.Cache.blks_hit |
| buffers_alloc | Cache | Blok per detik | db.Cache.buffers_alloc |
| buffers_checkpoint | Titik pemeriksaan | Blok per detik | db.Checkpoint.buffers_checkpoint |
| checkpoints_req | Titik pemeriksaan | Titik pemeriksaan per menit | db.Checkpoint.checkpoints_req |
| checkpoint_sync_time | Titik pemeriksaan | Milidetik per titik pemeriksaan | db.Checkpoint.checkpoint_sync_time |
| checkpoints_timed | Titik pemeriksaan | Titik pemeriksaan per menit | db.Checkpoint.checkpoints_timed |
| checkpoint_write_time | Titik pemeriksaan | Milidetik per titik pemeriksaan | db.Checkpoint.checkpoint_write_time |
| maxwritten_clean | Titik pemeriksaan | Penghentian pembersihan Bgwriter per menit | db.Checkpoint.maxwritten_clean |
| penguncian | Konkurensi | Penguncian per menit | db.Concurrency.deadlocks |
| blk_read_time | I/O | Milidetik | db.IO.blk_read_time |
| blks_read | I/O | Blok per detik | db.IO.blks_read |

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| buffers_backend | I/O | Blok per detik | db.IO.buffers_backend |
| buffers_backend_fsync | I/O | Blok per detik | db.IO.buffers_backend_fsync |
| buffers_clean | I/O | Blok per detik | db.IO.buffers_clean |
| temp_bytes | Temp | Byte per detik | db.Temp.temp_bytes |
| temp_files | Temp | File per menit | db.Temp.temp_files |
| xact_commit | Transaksi | Commit per detik | db.Transactions.xact_commit |
| xact_rollback | Transaksi | Pemulihan per detik | db.Transactions.xact_rollback |
| numbackends | Pengguna | Koneksi | db.User.numbackends |
| archived_count | WAL | File per menit | db.WAL.archived_count |

Penghitung non-native untuk Aurora PostgreSQL

Metrik penghitung non-native adalah penghitung yang ditentukan oleh Amazon Aurora. Metrik non-asli bisa berupa metrik yang Anda dapatkan dengan kueri tertentu. Metrik non-native juga bisa berupa metrik turunan, dengan dua penghitung native atau lebih digunakan dalam penghitungan untuk mengetahui rasio, tingkat hit, atau latensi.

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| checkpoint_sync_latency | Titik pemeriksaan | Milidetik | db.Checkpoint.checkpoint_sy | Jumlah total waktu yang telah dihabiskan di bagian pemrosesa | `checkpoint_sync_time / (checkpoi` |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| | | | nc_latency | n titik pemeriksaan tempat file disinkronkan ke disk. | `nts_timed + checkpoints_req)` |
| checkpoint_write_latency | Titik pemeriksaan | Milidetik | db.Checkpoint.checkpoint_write_latency | Jumlah total waktu yang telah dihabiskan di bagian pemrosesan titik pemeriksaan tempat file ditulis ke disk. | `checkpoint_write_time / (checkpoints_timed + checkpoints_req)` |
| local_blks_read | I/O | Blok | db.IO.local_blks_read | Jumlah total blok lokal yang dibaca. | Tidak berlaku |
| local_blk_read_time | I/O | Milidetik | db.IO.local_blk_read_time | Jika `track_io_timing` diaktifkan, ini akan melacak total waktu yang dihabiskan untuk membaca blok file data lokal, dalam milidetik, jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing. | Tidak berlaku |
| num_blocked_sessions | Kunci | db.locks.num_blocked_sessions | Jumlah sesi yang diblokir. | – | |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| orcache_blks_hit | I/O | Kueri | db.IO.orcache_blks_hit | Jumlah total hit blok bersama dari cache pembacaan yang dioptimalkan. | Tidak berlaku |
| orcache_blk_read_time | I/O | Milidetik | db.IO.orcache_blk_read_time | Jika `track_io_timing` diaktifkan, ini akan melacak total waktu yang dihabiskan untuk membaca blok file data dari cache Optimized Reads, dalam milidetik, jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing. | Tidak berlaku |
| read_latency | I/O | Milidetik | db.IO.read_latency | Waktu yang dihabiskan untuk membaca blok file data oleh backend dalam instans ini. | `blk_read_time / blks_read` |
| storage_blks_read | I/O | Blok | db.IO.storage_blks_read | Jumlah total blok bersama yang dibaca dari penyimpanan Aurora. | Tidak berlaku |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| storage_blk_read_time | I/O | Milidetik | db.IO.storage_blk_read_time | Jika `track_io_timing` diaktifkan, ini akan melacak total waktu yang dihabiskan untuk membaca blok file data dari penyimpanan Aurora, dalam milidetik, jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing. | Tidak berlaku |
| num_blocked_sessions | Kunci | db.locks.num_blocked_sessions | Jumlah sesi yang diblokir. | – | |
| active_count | Status | Sesi | db.state.active_count | Jumlah sesi di `active` negara bagian. | Tidak berlaku |
| idle_count | Status | Sesi | db.state.idle_count | Jumlah sesi di `idle` negara bagian. | Tidak berlaku |
| idle_in_transaction_aborted_count | Status | Sesi | db.state.idle_in_transaction_aborted_count | Jumlah sesi di `idle in transaction (aborted)` negara bagian. | Tidak berlaku |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| idle_in_transaction_count | Status | Sesi | db.state.idle_in_transaction_count | Jumlah sesi di `idle in transaction` negara bagian. | Tidak berlaku |
| idle_in_transaction_max_time | Status | Detik | db.state.idle_in_transaction_max_time | Durasi transaksi berjalan terpanjang di `idle in transaction` negara bagian, dalam hitungan detik. | Tidak berlaku |
| logical_reads | SQL | Blok | db.SQL.logical_reads | Jumlah total blok yang dicapai dan dibaca. | `blks_hit + blks_read` |
| queries_started | SQL | Kueri | db.SQL.queries | Jumlah kueri dimulai. | Tidak berlaku |
| queries_finished | SQL | Kueri | db.SQL.queries | Jumlah kueri selesai. | Tidak berlaku |
| total_query_time | SQL | Milidetik | db.SQL.total_query_time | Total waktu yang dihabiskan untuk mengeksekusi pernyataan, dalam milidetik. | Tidak berlaku |
| active_transactions | Transaksi | Transaksi | db.Transactions.active_transactions | Jumlah transaksi aktif. | Tidak berlaku |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| blocked_transactions | Transaksi | Transaksi | db.Transactions.blocked_transactions | Jumlah transaksi yang diblokir. | Tidak berlaku |
| commit_latency | Transaksi | Mikrodetik | db.Transactions.commit_latency | Durasi rata-rata operasi commit. | `db.Transactions.duration_commits / db.Transactions.xact_commit` |
| duration_commits | Transaksi | Milidetik | db.Transactions.duration_commits | Total waktu transaksi yang dihabiskan di menit terakhir, dalam milidetik. | Tidak berlaku |
| max_used_xact_ids | Transaksi | Transaksi | db.Transactions.max_used_xact_ids | Jumlah transaksi yang belum disedot. | Tidak berlaku |
| oldest_inactive_logical_replication_slot_xid_age | Transaksi | Panjang | db.transactions.oldest_inactive_logical_replication_slot_xid_age | Usia transaksi tertua di slot replikasi logis yang tidak aktif. | Tidak berlaku |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| oldest_active_logical_replication_slot_xid_age | Transaksi | Panjang | db.transactions.oldest_active_logical_replication_slot_xid_age | Usia transaksi tertua di slot replikasi logis aktif. | Tidak berlaku |
| oldest_reader_feedback_xid_age | Transaksi | Panjang | db.transactions.oldest_reader_feedback_xid_age | Usia transaksi tertua dari transaksi yang berjalan lama pada instance pembaca Aurora atau instance pembaca DB global Aurora. | Tidak berlaku |
| oldest_prepared_transaction_xid_age | Transaksi | Panjang | db.transactions.oldest_prepared_transaction_xid_age | Usia transaksi tertua yang disiapkan. | Tidak berlaku |
| oldest_running_transaction_xid_age | Transaksi | Panjang | db.transactions.oldest_running_transaction_xid_age | Usia transaksi berjalan tertua. | Tidak berlaku |

| Penghitung | Jenis | Unit | Metrik | Deskripsi | Definisi |
|---|---|---|---|---|---|
| max_connections | Pengguna | Pengguna | db.User.max_connections | Jumlah maksimum koneksi yang diizinkan untuk database sebagaimana dikonfigurasi dalam `max_connections` parameter. | Tidak berlaku |
| total_auth_attempts | Pengguna | Pengguna | db.User.total_auth_attempts | Jumlah upaya koneksi untuk contoh ini. | Tidak berlaku |
| archive_failed_count | WAL | File per menit | db.WAL.archive_failed_count | Jumlah upaya gagal untuk mengarsipkan file WAL, dalam file per menit. | Tidak berlaku |

## Statistik SQL untuk Wawasan Performa

Statistik SQL adalah metrik terkait performa tentang kueri SQL yang dikumpulkan oleh Wawasan Performa. Wawasan Performa mengumpulkan statistik untuk setiap detik yang digunakan untuk menjalankan kueri dan untuk setiap panggilan SQL. Statistik SQL adalah rata-rata untuk rentang waktu yang dipilih.

Digest SQL adalah gabungan semua kueri yang memiliki pola tertentu, tetapi tidak harus memiliki nilai literal yang sama. Digest menggantikan nilai literal dengan tanda tanya. Misalnya, `SELECT * FROM emp WHERE lname= ?`. Digest ini mungkin terdiri dari kueri turunan berikut:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Semua mesin mendukung statistik SQL untuk kueri digest.

Untuk informasi dukungan wilayah, mesin DB, dan kelas instans untuk fitur ini, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Topik

- Statistik SQL untuk Aurora MySQL
- Statistik SQL untuk Aurora PostgreSQL

## Statistik SQL untuk Aurora MySQL

Aurora MySQL mengumpulkan statistik SQL hanya pada tingkat digest. Tidak ada statistik yang ditampilkan di tingkat pernyataan.

Topik

- Statistik digest untuk Aurora MySQL
- Statistik per detik untuk Aurora MySQL
- Statistik per panggilan untuk Aurora MySQL
- Statistik utama untuk

### Statistik digest untuk Aurora MySQL

Wawasan Performa mengumpulkan statistik digest SQL dari tabel `events_statements_summary_by_digest`. Tabel `events_statements_summary_by_digest` dikelola oleh basis data Anda.

Tabel digest tidak memiliki kebijakan pengosongan. Ketika tabel penuh, AWS Management Console menunjukkan pesan berikut:

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

Dalam situasi ini, Aurora MySQL tidak melacak kueri SQL. Untuk mengatasi masalah ini, Wawasan Performa secara otomatis memotong tabel digest jika kedua kondisi berikut terpenuhi:

- Tabel penuh.
- Wawasan Performa mengelola Skema Performa secara otomatis.

Untuk manajemen otomatis, parameter `performance_schema` harus diatur ke `0` dan Sumber tidak boleh diatur ke `user`. Jika Wawasan Performa tidak mengelola Skema Performa secara otomatis, lihat Ikhtisar Skema Kinerja untuk Performance Insights di Aurora.

Dalam AWS CLI, periksa sumber nilai parameter dengan menjalankan describe-db-parametersperintah.

Statistik per detik untuk Aurora MySQL

Statistik SQL berikut ini tersedia untuk klaster Aurora MySQL.

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.count_star_per_sec | Panggilan per detik |
| db.sql_tokenized.stats.sum_timer_wait_per_sec | Latensi rata-rata per detik (dalam ms) |
| db.sql_tokenized.stats.sum_select_full_join_per_sec | Memilih penggabungan penuh per detik |
| db.sql_tokenized.stats.sum_select_range_check_per_sec | Memilih pemeriksaan rentang per detik |
| db.sql_tokenized.stats.sum_select_scan_per_sec | Memilih pemindaian per detik |
| db.sql_tokenized.stats.sum_sort_merge_passes_per_sec | Mengurutkan pass penggabungan per detik |
| db.sql_tokenized.stats.sum_sort_scan_per_sec | Mengurutkan pemindaian per detik |
| db.sql_tokenized.stats.sum_sort_range_per_sec | Mengurutkan rentang per detik |
| db.sql_tokenized.stats.sum_sort_rows_per_sec | Mengurutkan baris per detik |
| db.sql_tokenized.stats.sum_rows_affected_per_sec | Baris yang terpengaruh per detik |

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.sum_rows_examined_per_sec | Baris yang diperiksa per detik |
| db.sql_tokenized.stats.sum_rows_sent_per_sec | Baris yang dikirim per detik |
| db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec | Tabel disk sementara yang dibuat per detik |
| db.sql_tokenized.stats.sum_created_tmp_tables_per_sec | Tabel disk sementara yang dibuat per detik |
| db.sql_tokenized.stats.sum_lock_time_per_sec | Waktu penguncian per detik (dalam md) |

Statistik per panggilan untuk Aurora MySQL

Metrik berikut menyediakan statistik per panggilan untuk pernyataan SQL.

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.sum_timer_wait_per_call | Latensi rata-rata per panggilan (dalam md) |
| db.sql_tokenized.stats.sum_select_full_join_per_call | Memilih penggabungan penuh per panggilan |
| db.sql_tokenized.stats.sum_select_range_check_per_call | Memilih pemeriksaan rentang per panggilan |
| db.sql_tokenized.stats.sum_select_scan_per_call | Memilih pemindaian per panggilan |
| db.sql_tokenized.stats.sum_sort_merge_passes_per_call | Mengurutkan pass penggabungan per panggilan |
| db.sql_tokenized.stats.sum_sort_scan_per_call | Mengurutkan pemindaian per panggilan |
| db.sql_tokenized.stats.sum_sort_range_per_call | Mengurutkan rentang per panggilan |

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.sum_sort_rows_per_call | Mengurutkan baris per panggilan |
| db.sql_tokenized.stats.sum_rows_affected_per_call | Baris yang terpengaruh per panggilan |
| db.sql_tokenized.stats.sum_rows_examined_per_call | Baris yang diperiksa per panggilan |
| db.sql_tokenized.stats.sum_rows_sent_per_call | Baris yang terkirim per panggilan |
| db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call | Tabel disk sementara yang dibuat per panggilan |
| db.sql_tokenized.stats.sum_created_tmp_tables_per_call | Tabel sementara yang dibuat per panggilan |
| db.sql_tokenized.stats.sum_lock_time_per_call | Waktu penguncian per panggilan (dalam md) |

Statistik utama untuk

Statistik SQL berikut ini tersedia untuk klaster Aurora MySQL.

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.count_star | Panggilan |
| db.sql_tokenized.stats.sum_timer_wait | Waktu tunggu (dalam ms) |
| db.sql_tokenized.stats.sum_select_full_join | Pilih bergabung penuh |
| db.sql_tokenized.stats.sum_select_range_check | Pilih pemeriksaan rentang |
| db.sql_tokenized.stats.sum_select_scan | Pilih pemindaian |
| db.sql_tokenized.stats.sum_sort_merge_passes | Urutkan pass gabungan |
| db.sql_tokenized.stats.sum_sort_scan | Urutkan pemindaian |

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.sum_sort_range | Urutkan rentang |
| db.sql_tokenized.stats.sum_sort_rows | Urutkan baris |
| db.sql_tokenized.stats.sum_rows_affected | Baris terpengaruh |
| db.sql_tokenized.stats.sum_rows_examined | Baris diperiksa |
| db.sql_tokenized.stats.sum_rows_sent | Baris dikirim |
| db.sql_tokenized.stats.sum_created_tmp_disk_tables | Membuat tabel disk sementara |
| db.sql_tokenized.stats.sum_created_tmp_tables | Membuat tabel sementara |
| db.sql_tokenized.stats.sum_lock_time | Waktu kunci (dalam ms) |

## Statistik SQL untuk Aurora PostgreSQL

Untuk setiap panggilan SQL dan untuk setiap detik eksekusi kueri, Wawasan Performa mengumpulkan statistik SQL. Semua mesin Aurora mengumpulkan statistik hanya pada tingkat digest.

Berikut ini, Anda dapat menemukan informasi tentang statistik tingkat digest untuk Aurora PostgreSQL.

Topik

- Statistik digest untuk Aurora PostgreSQL
- Statistik digest per detik untuk Aurora PostgreSQL
- Statistik digest per panggilan untuk AuroraPostgreSQL
-

### Statistik digest untuk Aurora PostgreSQL

Untuk menampilkan statistik digest SQL, pustaka `pg_stat_statements` harus dimuat. Untuk klaster DB Aurora PostgreSQL yang kompatibel dengan PostgreSQL 10, pustaka ini dimuat

secara default. Untuk klaster DB Aurora PostgreSQL yang kompatibel dengan PostgreSQL 9.6, aktifkan pustaka ini secara manual. Untuk mengaktifkannya secara manual, tambahkan `pg_stat_statements` ke `shared_preload_libraries` di grup parameter DB yang terkait dengan instans DB. Lalu boot ulang instans DB Anda. Untuk informasi selengkapnya, lihat .

> **Note**
>
> Wawasan Performa hanya dapat mengumpulkan statistik untuk kueri dalam `pg_stat_activity` yang tidak terpotong. Secara default, basis data PostgreSQL memotong kueri yang lebih panjang dari 1.024 byte. Untuk menambah ukuran kueri, ubah parameter `track_activity_query_size` dalam grup parameter DB yang terkait dengan instans DB Anda. Jika Anda mengubah parameter ini, instans DB harus di-boot ulang.

Statistik digest per detik untuk Aurora PostgreSQL

Statistik digest SQL berikut tersedia untuk instans DB Aurora PostgreSQL.

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.calls_per_sec | Panggilan per detik |
| db.sql_tokenized.stats.rows_per_sec | Baris per detik |
| db.sql_tokenized.stats.total_time_per_sec | Eksekusi aktif rata-rata per detik (AAE) |
| db.sql_tokenized.stats.shared_blks_hit_per_sec | Hit blok per detik |
| db.sql_tokenized.stats.shared_blks_read_per_sec | Pembacaan blok per detik |
| db.sql_tokenized.stats.shared_blks_dirtied_per_sec | Blok kotor per detik |
| db.sql_tokenized.stats.shared_blks_written_per_sec | Penulisan blok per detik |
| db.sql_tokenized.stats.local_blks_hit_per_sec | Hit blok lokal per detik |
| db.sql_tokenized.stats.local_blks_read_per_sec | Pembacaan blok lokal per detik |

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.local_blks_dirtied_per_sec | Blok lokal kotor per detik |
| db.sql_tokenized.stats.local_blks_written_per_sec | Penulisan blok lokal per detik |
| db.sql_tokenized.stats.temp_blks_written_per_sec | Penulisan sementara per detik |
| db.sql_tokenized.stats.temp_blks_read_per_sec | Pembacaan sementara per detik |
| db.sql_tokenized.stats.blk_read_time_per_sec | Pembacaan serentak rata-rata per detik |
| db.sql_tokenized.stats.blk_write_time_per_sec | Penulisan serentak rata-rata per detik |

Statistik digest per panggilan untuk AuroraPostgreSQL

Metrik berikut menyediakan statistik per panggilan untuk pernyataan SQL.

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.rows_per_call | Baris per panggilan |
| db.sql_tokenized.stats.avg_latency_per_call | Latensi rata-rata per panggilan (dalam md) |
| db.sql_tokenized.stats.shared_blks_hit_per_call | Hit blok per panggilan |
| db.sql_tokenized.stats.shared_blks_read_per_call | Pembacaan blok per panggilan |
| db.sql_tokenized.stats.shared_blks_written_per_call | Penulisan blok per panggilan |
| db.sql_tokenized.stats.shared_blks_dirtied_per_call | Blok kotor per panggilan |
| db.sql_tokenized.stats.local_blks_hit_per_call | Hit blok lokal per panggilan |

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.local_blks_read_per_call | Pembacaan blok lokal per panggilan |
| db.sql_tokenized.stats.local_blks_dirtied_per_call | Blok lokal dikotori per panggilan |
| db.sql_tokenized.stats.local_blks_written_per_call | Penulisan blok lokal per panggilan |
| db.sql_tokenized.stats.temp_blks_written_per_call | Penulisan blok sementara per panggilan |
| db.sql_tokenized.stats.temp_blks_read_per_call | Pembacaan blok sementara per panggilan |
| db.sql_tokenized.stats.blk_read_time_per_call | Waktu baca per panggilan (dalam md) |
| db.sql_tokenized.stats.blk_write_time_per_call | Waktu tulis per panggilan (dalam md) |

Statistik SQL berikut tersedia untuk instans Aurora PostgreSQL DB.

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.calls | Panggilan |
| db.sql_tokenized.stats.rows | Baris |
| db.sql_tokenized.stats.total_time | Total waktu (dalam ms) |
| db.sql_tokenized.stats.shared_blks_hit | Blok hits |
| db.sql_tokenized.stats.shared_blks_read | Memblokir membaca |
| db.sql_tokenized.stats.shared_blks_dirtied | Blok kotor |
| db.sql_tokenized.stats.shared_blks_written | Blok menulis |
| db.sql_tokenized.stats.local_blks_hit | Hit blok lokal |

| Metrik | Unit |
| --- | --- |
| db.sql_tokenized.stats.local_blks_read | Pembacaan blok lokal |
| db.sql_tokenized.stats.local_blks_dirtied | Blok lokal kotor |
| db.sql_tokenized.stats.local_blks_written | Blok lokal menulis |
| db.sql_tokenized.stats.temp_blks_written | Menulis sementara |
| db.sql_tokenized.stats.temp_blks_read | Bacaan sementara |
| db.sql_tokenized.stats.blk_read_time | Rata-rata pembacaan bersamaan (dalam ms) |
| db.sql_tokenized.stats.blk_write_time | Rata-rata penulisan bersamaan (dalam ms) |

Untuk informasi selengkapnya tentang metrik ini, lihat pg_stat_statements dalam dokumentasi PostgreSQL.

## Metrik OS dalam Pemantauan yang Ditingkatkan

Amazon Aurora menyediakan metrik secara real time untuk sistem operasi (OS) tempat cluster DB Anda berjalan. Aurora memberikan metrik dari Enhanced Monitoring ke akun Amazon Logs Anda. CloudWatch Tabel berikut mencantumkan metrik OS yang tersedia menggunakan Amazon CloudWatch Logs.

Topik

- Metrik OS untuk Aurora

### Metrik OS untuk Aurora

| Grup | Metrik | Nama konsol | Deskripsi |
| --- | --- | --- | --- |
| `General` | `engine` | Tidak berlaku | Mesin basis data untuk instans DB. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | instanceID | Tidak berlaku | Pengidentifikasi instans DB. |
| | instanceResourceID | Tidak berlaku | Pengidentifikasi yang tidak dapat diubah untuk instans DB yang unik untuk AWS Wilayah, juga digunakan sebagai pengidentifikasi aliran log. |
| | numVCPUs | Tidak berlaku | Jumlah CPUs virtual untuk instans DB. |
| | timestamp | Tidak berlaku | Waktu pengambilan metrik. |
| | uptime | Tidak berlaku | Jumlah waktu saat instans DB telah aktif. |
| | version | Tidak berlaku | Versi format aliran JSON metrik OS. |
| cpuUtilization | guest | CPUTamu | Persentase CPU yang digunakan oleh program tamu. |
| | idle | CPUMenganggur | Persentase CPU itu menganggur. |
| | irq | CPU IRQ | Persentase CPU yang digunakan oleh perangkat lunak mengganggu. |
| | nice | CPUBagus | Persentase CPU yang digunakan oleh program yang berjalan pada prioritas terendah. |
| | steal | CPUMencuri | Persentase CPU yang digunakan oleh mesin virtual lainnya. |
| | system | CPUSistem | Persentase CPU yang digunakan oleh kernel. |
| | total | CPUJumlah | Persentase total CPU yang digunakan. Nilai ini mencakup nilai nice. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | `user` | CPUPengguna | Persentase CPU yang digunakan oleh program pengguna. |
| | `wait` | CPUTunggu | Persentase yang CPU tidak digunakan saat menunggu akses I/O. |
| `diskIO` | `avgQueueLen` | Ukuran Antrean Rata-rata | Jumlah permintaan yang menunggu dalam antrean perangkat I/O. |
| | `avgReqSz` | Ukuran Permintaan Rata-rata | Ukuran permintaan rata-rata, dalam kilobyte. |
| | `await` | I/O Disk Tunggu | Jumlah milidetik yang diperlukan untuk merespons permintaan, termasuk waktu antrean dan waktu layanan. |
| | `device` | Tidak berlaku | Pengidentifikasi perangkat disk yang digunakan. |
| | `readIOsPS` | IO/d Baca | Jumlah operasi baca per detik. |
| | `readKb` | Total Baca | Jumlah total kilobyte yang dibaca. |
| | `readKbPS` | Kb/d Baca | Jumlah kilobyte yang dibaca per detik. |
| | `readLatency` | Latensi Baca | Waktu yang berlalu antara pengiriman permintaan I/O baca dan penyelesaiannya, dalam milidetik.<br><br>Metrik ini hanya tersedia untuk Amazon Aurora. |
| | `readThroughput` | Throughput Baca | Jumlah throughput jaringan yang digunakan oleh permintaan ke klaster DB, dalam byte per detik.<br><br>Metrik ini hanya tersedia untuk Amazon Aurora. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | `rrqmPS` | Rrqms | Jumlah permintaan baca gabungan yang diantrekan per detik. |
| | `tps` | TPS | Jumlah transaksi I/O per detik. |
| | `util` | Penggunaan I/O Disk | Persentase CPU waktu di mana permintaan dikeluarkan. |
| | `writeIOsPS` | IO/d Tulis | Jumlah operasi tulis per detik. |
| | `writeKb` | Total Tulis | Jumlah total kilobyte yang ditulis. |
| | `writeKbPS` | Kb/d Tulis | Jumlah kilobyte yang ditulis per detik. |
| | `writeLatency` | Latensi Tulis | Waktu yang berlalu antara pengiriman permintaan I/O tulis dan penyelesaiannya, dalam milidetik.<br>Metrik ini hanya tersedia untuk Amazon Aurora. |
| | `writeThroughput` | Throughput Tulis | Jumlah throughput jaringan yang digunakan oleh respons dari klaster DB, dalam byte per detik.<br>Metrik ini hanya tersedia untuk Amazon Aurora. |
| | `wrqmPS` | Wrqms | Jumlah permintaan tulis gabungan yang diantrekan per detik. |
| `fileSys` | `maxFiles` | Inode Maks | Jumlah maksimum file yang dapat dibuat untuk sistem file. |
| | `mountPoint` | Tidak berlaku | Jalur ke sistem file. |
| | `name` | Tidak berlaku | Nama sistem file. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | `total` | Total Sistem File | Jumlah total ruang disk yang tersedia untuk sistem file, dalam kilobyte. |
| | `used` | Sistem file yang digunakan | Jumlah ruang disk yang digunakan oleh file dalam sistem file, dalam kilobyte. |
| | `usedFilePercent` | Inode yang Digunakan | Persentase file tersedia yang digunakan. |
| | `usedFiles` | % yang Digunakan | Jumlah file dalam sistem file. |
| | `usedPercent` | Sistem File yang Digunakan | Persentase ruang disk sistem file yang digunakan. |
| `loadAverageMinute` | `fifteen` | Rata-rata Muatan 15 menit | Jumlah proses yang meminta CPU waktu selama 15 menit terakhir. |
| | `five` | Rata-rata Muatan 5 menit | Jumlah proses yang meminta CPU waktu selama 5 menit terakhir. |
| | `one` | Rata-rata Muatan 1 menit | Jumlah proses yang meminta CPU waktu selama menit terakhir. |
| `memory` | `active` | Memori Aktif | Jumlah memori yang ditetapkan, dalam kilobyte. |
| | `buffers` | Memori yang Di-buffer | Jumlah memori yang digunakan untuk buffering permintaan I/O sebelum menulis ke perangkat penyimpanan, dalam kilobyte. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | `cached` | Memori yang Di-cache | Jumlah memori yang digunakan untuk meng-cache file I/O berbasis sistem file. |
| | `dirty` | Memori Kotor | Jumlah halaman memori RAM yang telah dimodifikasi tetapi tidak ditulis ke blok data terkait dalam penyimpanan, dalam kilobyte. |
| | `free` | Memori Kosong | Jumlah memori yang tidak ditetapkan, dalam kilobyte. |
| | `hugePagesFree` | Halaman Besar Kosong | Jumlah halaman besar yang kosong. Halaman besar adalah fitur dari kernel Linux. |
| | `hugePagesRsvd` | Rsvd Halaman Besar | Jumlah halaman besar yang khusus. |
| | `hugePagesSize` | Ukuran Halaman Besar | Ukuran untuk setiap unit halaman besar, dalam kilobyte. |
| | `hugePagesSurp` | Surplus Halaman Besar | Jumlah halaman besar surplus yang tersedia terhadap total. |
| | `hugePagesTotal` | Total Halaman Besar | Jumlah total halaman besar. |
| | `inactive` | Memori Tidak Aktif | Jumlah halaman memori yang jarang digunakan, dalam kilobyte. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | `mapped` | Memori yang Dipetakan | Jumlah total konten sistem file yang memorinya dipetakan di dalam ruang alamat proses, dalam kilobyte. |
| | `pageTables` | Tabel Halaman | Jumlah memori yang digunakan berdasarkan tabel halaman, dalam kilobyte. |
| | `slab` | Memori Slab | Jumlah struktur data kernel yang dapat digunakan kembali, dalam kilobyte. |
| | `total` | Memori Total | Jumlah total memori, dalam kilobyte. |
| | `writeback` | Memori Tulis Ulang | Jumlah halaman kotor RAM yang masih ditulis ke penyimpanan pendukung, dalam kilobyte. |
| `network` | `interface` | Tidak berlaku | Pengidentifikasi untuk antarmuka jaringan yang digunakan untuk instans DB. |
| | `rx` | RX | Jumlah byte yang diterima per detik. |
| | `tx` | TX | Jumlah byte yang diunggah per detik. |
| `processList` | `cpuUsedPc` | CPU % | Persentase yang CPU digunakan oleh proses. |
| | `id` | Tidak berlaku | Pengidentifikasi proses. |
| | `memoryUsedPc` | MEM% | Persentase memori yang digunakan oleh proses. |
| | `name` | Tidak berlaku | Nama proses. |
| | `parentID` | Tidak berlaku | Pengidentifikasi proses untuk proses induk dari proses. |

| Grup | Metrik | Nama konsol | Deskripsi |
|---|---|---|---|
| | `rss` | RES | Jumlah yang RAM dialokasikan untuk proses, dalam kilobyte. |
| | `tgid` | Tidak berlaku | Pengidentifikasi grup utas, yaitu angka yang mewakili ID proses di tempat utas berada. Pengidentifikasi ini digunakan untuk mengelompokkan utas dari proses yang sama. |
| | `vss` | VIRT | Jumlah memori virtual yang dialokasikan untuk proses, dalam kilobyte. |
| `swap` | `total` | Swap | Jumlah memori swap yang tersedia, dalam kilobyte. |
| | `in` | Swap dalam | Jumlah memori, dalam kilobyte, yang ditukar ke dalam dari disk. |
| | `out` | Swap luar | Jumlah memori, dalam kilobyte, yang ditukar ke luar dari disk. |
| | `free` | Swap Kosong | Jumlah memori swap yang kosong, dalam kilobyte. |
| | `cached` | Swap Komit | Jumlah memori swap, dalam kilobyte, yang digunakan sebagai memori cache. |
| `tasks` | `blocked` | Tugas Diblokir | Jumlah tugas yang diblokir. |
| | `running` | Tugas Berjalan | Jumlah tugas yang berjalan. |
| | `sleeping` | Tugas Tidur | Jumlah tugas yang tidur. |
| | `stopped` | Tugas Dihentikan | Jumlah tugas yang dihentikan. |

| Grup | Metrik | Nama konsol | Deskripsi |
| --- | --- | --- | --- |
| | `total` | Total Tugas | Jumlah total tugas. |
| | `zombie` | Tugas Zombie | Jumlah tugas turunan yang tidak aktif dengan tugas induk aktif. |

Saat Anda memantau basis data Amazon Aurora dan AWS solusi Anda yang lain, tujuan Anda adalah mempertahankan hal-hal berikut:

- Keandalan
- Ketersediaan
- Performa
- Keamanan

Memantau metrik di klaster Amazon Aurora menjelaskan cara memantau klaster menggunakan metrik. Solusi lengkap juga harus memantau peristiwa database, file log, dan aliran aktivitas. AWS menyediakan Anda dengan alat pemantauan berikut:

- Amazon EventBridge adalah layanan bus acara tanpa server yang memudahkan untuk menghubungkan aplikasi Anda dengan data dari berbagai sumber. EventBridge memberikan aliran data real-time dari aplikasi Anda sendiri, aplikasi S oftware-as-a -Service (SaaS), dan layanan. AWS EventBridge merutekan data tersebut ke target seperti AWS Lambda. Dengan demikian, Anda dapat memantau peristiwa yang terjadi di layanan dan membangun arsitektur berbasis peristiwa. Untuk informasi selengkapnya, lihat Panduan EventBridge Pengguna Amazon.
- Amazon CloudWatch Logs menyediakan cara untuk memantau, menyimpan, dan mengakses file log Anda dari instans Aurora Amazon, AWS CloudTrail, dan sumber lainnya. Amazon CloudWatch Logs dapat memantau informasi dalam file log dan memberi tahu Anda ketika ambang batas tertentu terpenuhi. Anda juga dapat mengarsipkan data log dalam penyimpanan yang sangat durabel. Untuk informasi selengkapnya, lihat Panduan Pengguna Amazon CloudWatch Logs.
- AWS CloudTrailmenangkap API panggilan dan acara terkait yang dibuat oleh atau atas nama Anda Akun AWS. CloudTrail mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Anda dapat mengidentifikasi pengguna dan akun mana yang dipanggil AWS, alamat IP sumber dari mana panggilan dilakukan, dan kapan panggilan terjadi. Untuk informasi selengkapnya, lihat Panduan Pengguna AWS CloudTrail.
- Aliran Aktivitas Database Amazon Aurora  mendorong aktivitas ke aliran data Amazon Kinesis. Aliran Kinesis dibuat secara otomatis. Dari Kinesis, Anda dapat mengonfigurasi AWS layanan seperti Amazon Data Firehose dan AWS Lambda menggunakan aliran dan menyimpan data.

Topik

- Melihat log, peristiwa, dan streaming di konsol Amazon RDS

- 
- Memantau log Amazon Aurora Amazon
- Memantau panggilan Amazon Aurora API AWS CloudTrail
- Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data
- Memantau ancaman dengan Perlindungan Amazon GuardDuty RDS untuk Amazon Aurora

## Melihat log, peristiwa, dan streaming di konsol Amazon RDS

Amazon RDS terintegrasi dengan Layanan AWS untuk menampilkan informasi tentang log, peristiwa, dan aliran aktivitas database di RDS konsol.

Tab Log & peristiwa untuk cluster Aurora DB Anda menampilkan informasi berikut:

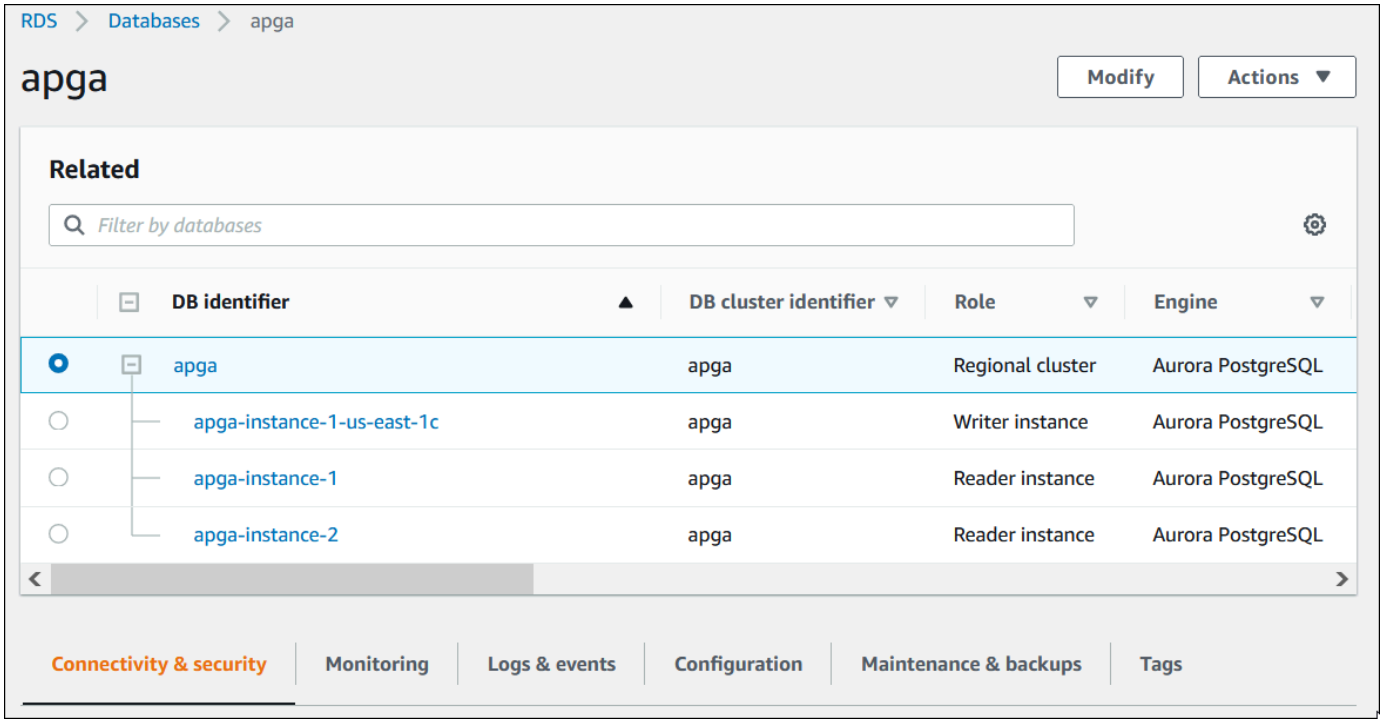
- Kebijakan dan aktivitas penskalaan otomatis – Menampilkan kebijakan dan aktivitas yang berkaitan dengan fitur Auto Scaling Aurora. Informasi ini hanya muncul di tab Log & peristiwa pada tingkat klaster.
- Alarm Amazon — Menampilkan CloudWatch alarm metrik apa pun yang telah Anda konfigurasikan untuk instans DB di klaster Aurora Anda. Jika Anda belum mengonfigurasi alarm, Anda dapat membuatnya di RDS konsol.
- Peristiwa terbaru — Menampilkan ringkasan peristiwa (perubahan lingkungan) untuk instans atau cluster Aurora DB Anda. Untuk informasi selengkapnya, lihat Melihat RDS acara Amazon.
- Log – Menampilkan file log basis data yang dihasilkan oleh instans DB di klaster Aurora Anda. Untuk informasi selengkapnya, lihat Memantau log Amazon Aurora Amazon.

Tab Konfigurasi akan menayangkan informasi tentang stream aktivitas basis data.

Untuk melihat log, peristiwa, dan aliran untuk cluster instansAurora Anda di konsol RDS

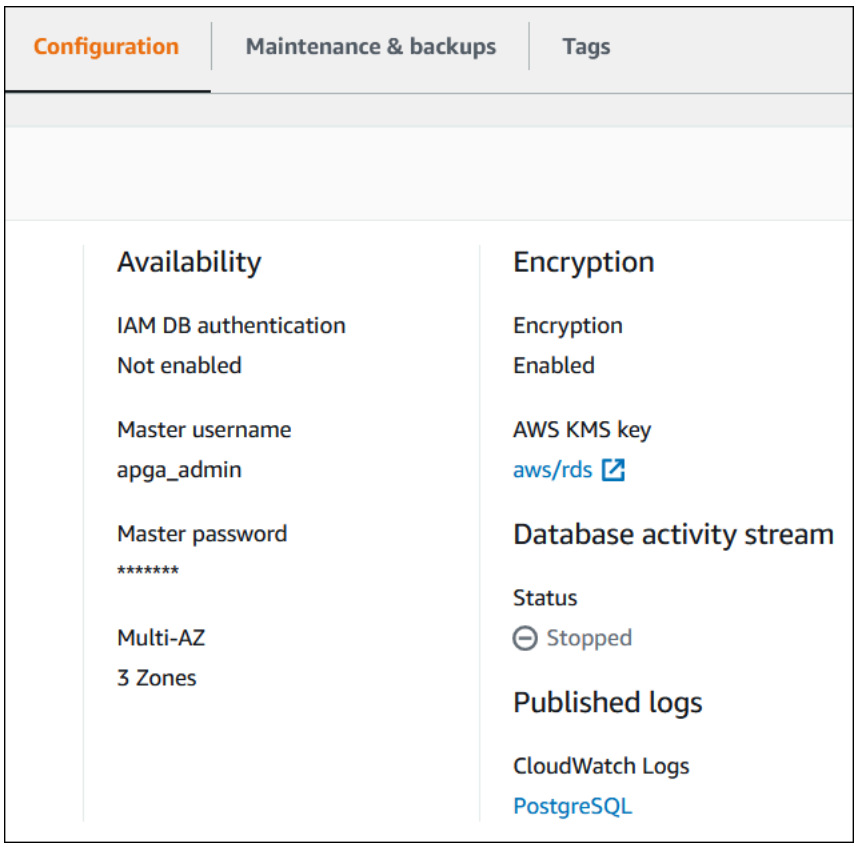
1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama klaster DB Aurora yang ingin dipantau.

   Halaman basis data akan muncul. Contoh berikut menunjukkan Amazon Aurora SQL Postgre DB cluster bernama. `apga`



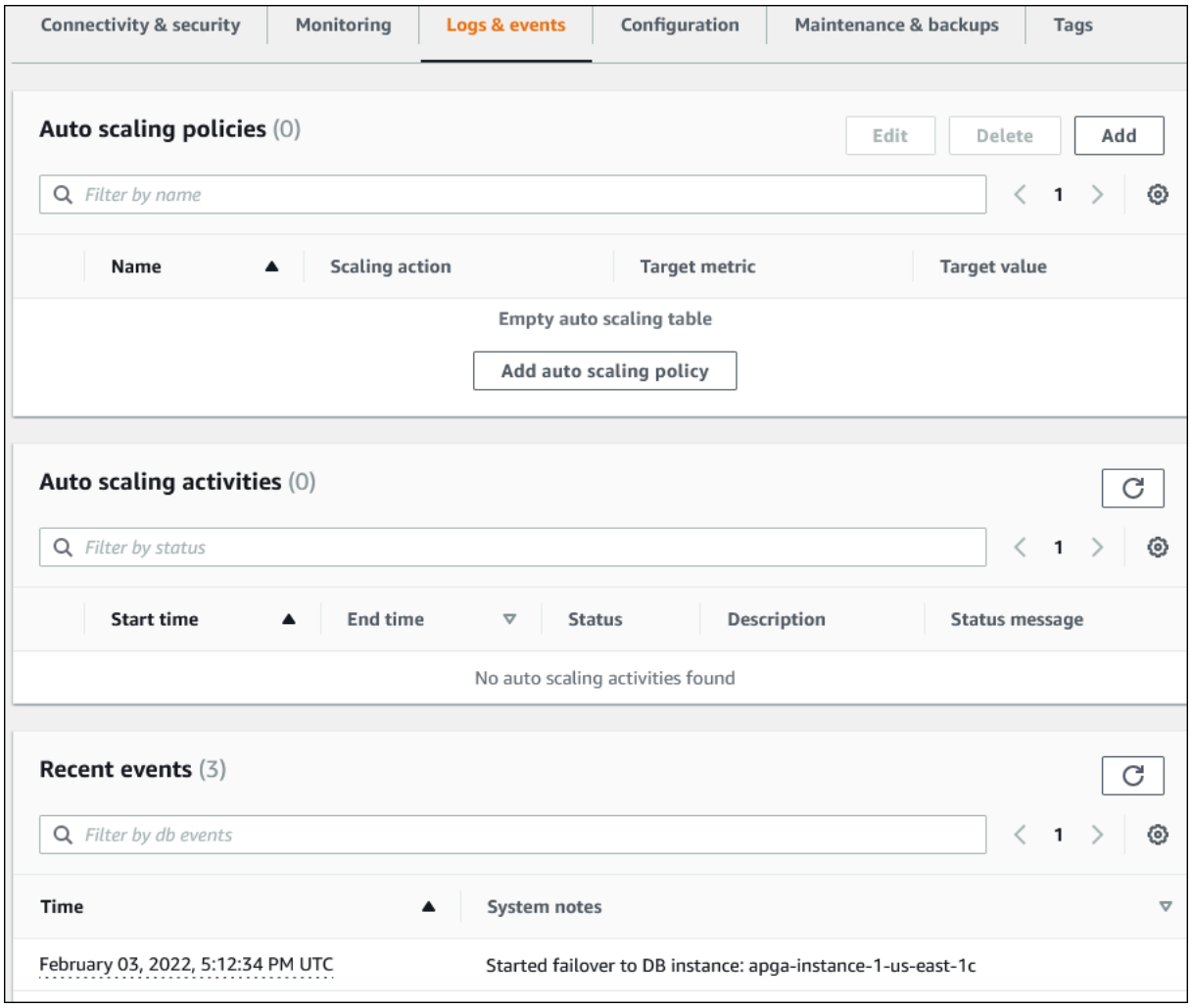

4. Gulir turun dan pilih Konfigurasi.

   Contoh berikut menunjukkan status stream aktivitas basis data untuk klaster Anda.



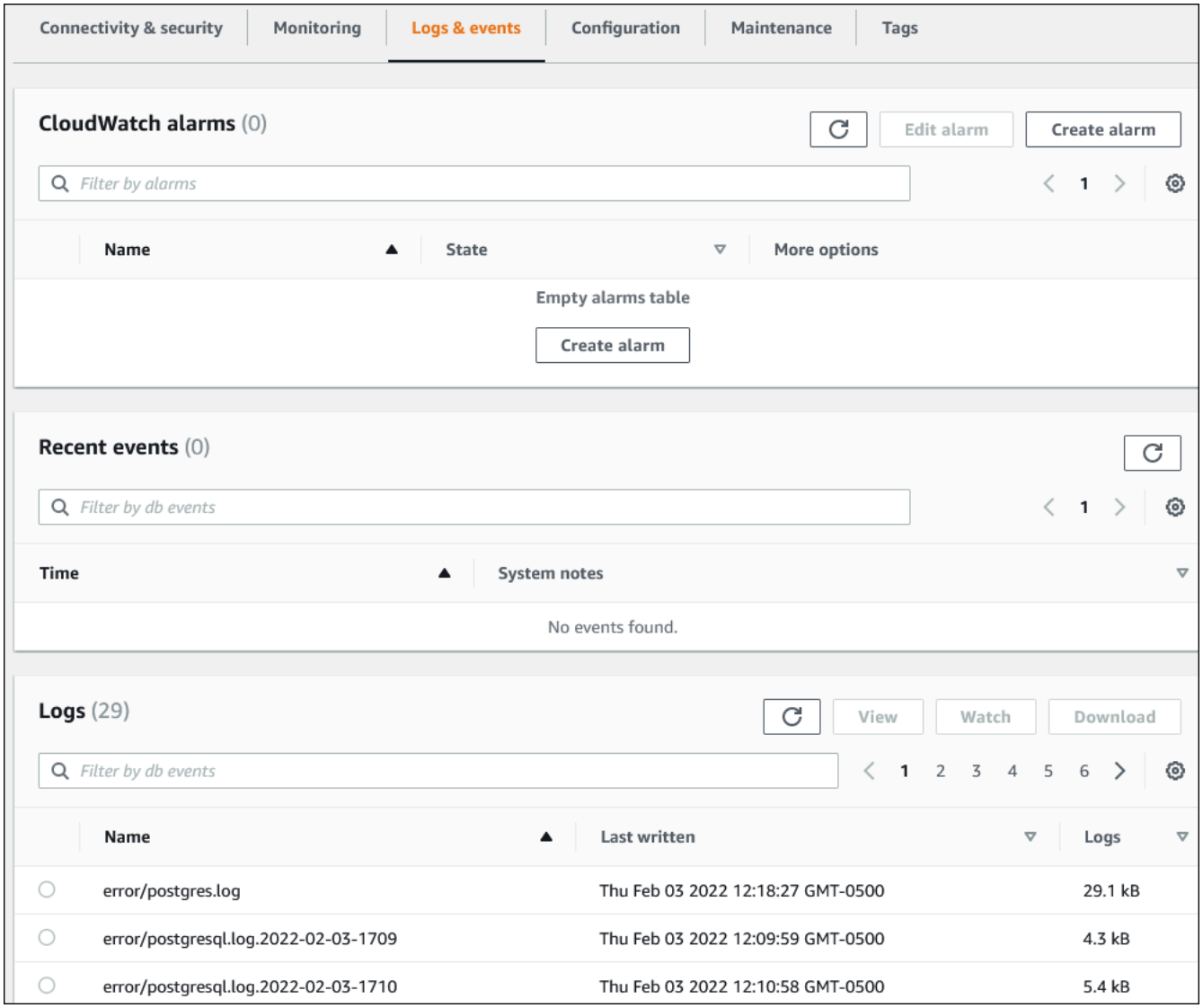
5. Pilih Log & peristiwa.

   Bagian Log & peristiwa akan muncul.




6. Pilih instans DB di klaster Aurora Anda, lalu pilih Log & peristiwa untuk instans itu.

   Contoh berikut menunjukkan perbedaan konten antara halaman instans DB dan halaman klaster DB. Halaman instans DB menampilkan log dan alarm.


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

Peristiwa menunjukkan perubahan dalam lingkungan. Ini bisa berupa AWS lingkungan, layanan atau aplikasi mitra SaaS, atau aplikasi atau layanan khusus. Untuk deskripsi peristiwa Aurora, lihat. Kategori acara Amazon RDS dan pesan acara untuk Aurora

Topik


- 
- Melihat RDS acara Amazon
- Bekerja dengan pemberitahuan RDS acara Amazon
- Membuat aturan yang dipicu pada peristiwa Amazon Aurora
- Kategori acara Amazon RDS dan pesan acara untuk Aurora


Sebuah RDSperistiwa menunjukkan perubahan di lingkungan Aurora . Misalnya, Aurora menghasilkan peristiwa saat klaster DB ditambal, berjalan. Aurora mengirimkan acara ke EventBridge dalam waktu yang hampir nyata.

> **Note**
>
> Amazon RDS memancarkan acara dengan upaya terbaik. Sebaiknya Anda menghindari menulis program yang mengandalkan urutan atau keberadaan peristiwa pemberitahuan, karena program tersebut mungkin tidak berurutan atau hilang.

Amazon RDS mencatat peristiwa yang berhubungan dengan sumber daya berikut:

- Klaster DB

  Untuk daftar peristiwa klaster, lihat Peristiwa klaster DB.
- Instans DB

  Untuk daftar peristiwa instans DB, lihat Peristiwa instans DB.
- Grup parameter DB

  Untuk daftar peristiwa grup parameter DB, lihat Peristiwa grup parameter DB.
- Grup keamanan DB

  Untuk daftar peristiwa grup keamanan DB, lihat Peristiwa grup keamanan DB.

- Snapshot klaster DB

  Untuk daftar peristiwa snapshot klaster DB, lihat Peristiwa snapshot klaster DB.
- RDSAcara proxy

  Untuk daftar peristiwa RDS Proxy, lihatPeristiwa Proksi RDS.
- Peristiwa deployment blue/green

  Untuk daftar peristiwa deployment blue/green, lihat Peristiwa deployment blue/green.

Informasi ini mencakup hal berikut:

- Tanggal dan waktu peristiwa
- Nama sumber dan jenis sumber peristiwa
- Pesan yang terkait dengan peristiwa
- Pemberitahuan peristiwa mencakup tag sejak pesan dikirim dan mungkin tidak mencerminkan tag pada saat peristiwa terjadi

# Melihat RDS acara Amazon

Anda dapat mengambil informasi acara berikut untuk sumber daya Aurora Anda:

- Nama sumber daya
- Jenis sumber daya
- Waktu peristiwa
- Ringkasan pesan peristiwa

Anda dapat mengakses acara di bagian berikut AWS Management Console:

- Tab Acara, yang menampilkan peristiwa dari 24 jam terakhir.
- Tabel peristiwa terbaru di bagian Log & peristiwa di tab Database, yang dapat menampilkan peristiwa hingga 2 minggu terakhir.

Anda juga dapat mengambil peristiwa dengan menggunakan AWS CLI perintah describe-events, atau operasi. DescribeEventsRDSAPI Jika Anda menggunakan AWS CLI atau RDS API untuk melihat acara, Anda dapat mengambil acara hingga 14 hari terakhir.

> **Note**
>
> Jika Anda perlu menyimpan acara untuk jangka waktu yang lebih lama, Anda dapat mengirim RDS acara Amazon ke EventBridge. Untuk informasi selengkapnya, silakan lihat Membuat aturan yang dipicu pada peristiwa Amazon Aurora

Untuk deskripsi acara Aurora RDS Amazon, lihat. Kategori acara Amazon RDS dan pesan acara untuk Aurora

Untuk mengakses informasi terperinci tentang peristiwa yang menggunakan AWS CloudTrail, termasuk parameter permintaan, lihatCloudTrail acara.

Konsol

Untuk melihat semua RDS acara Amazon selama 24 jam terakhir

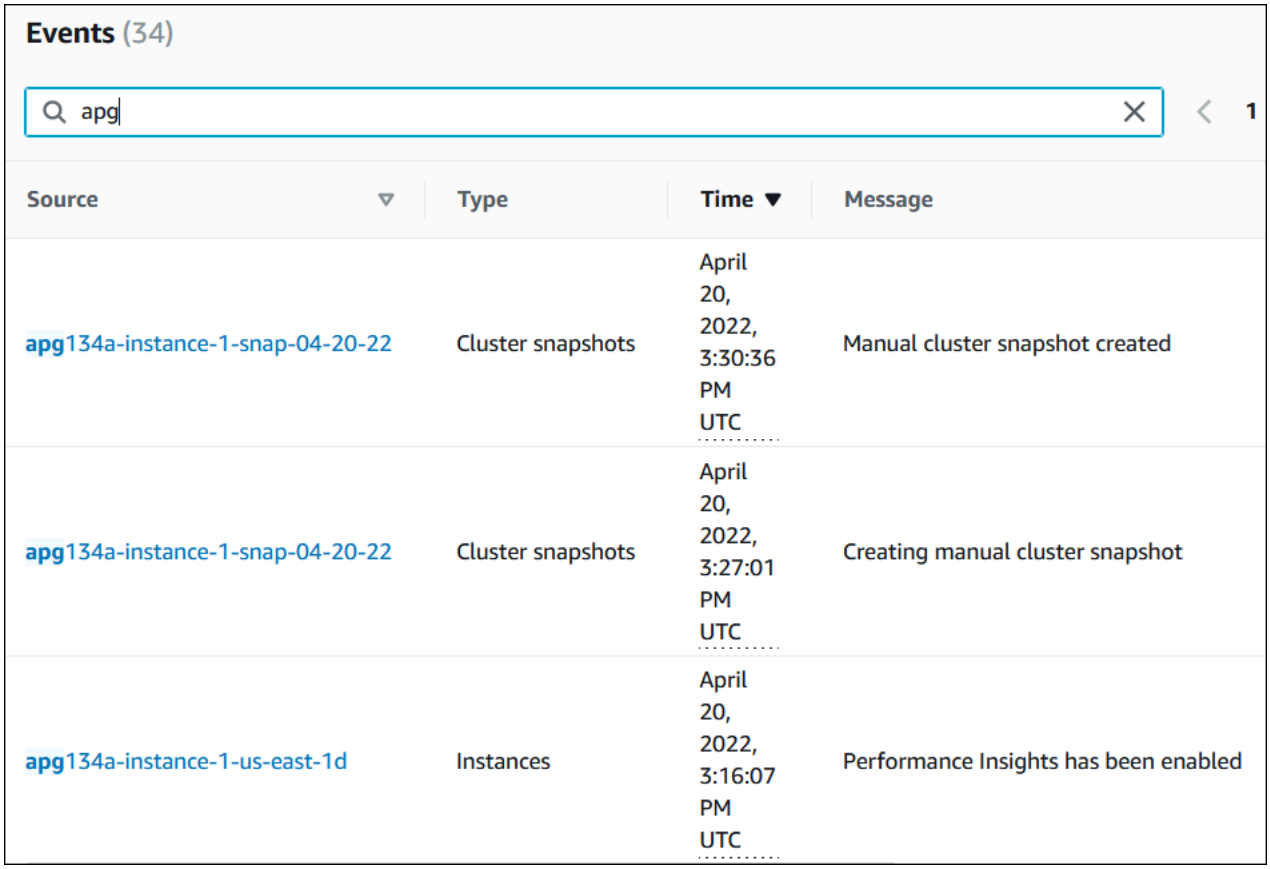
1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Peristiwa.

   Peristiwa yang tersedia akan muncul dalam daftar.

3. (Opsional) Masukkan istilah pencarian untuk memfilter hasil.

   Contoh berikut menunjukkan daftar peristiwa yang difilter oleh karakter **apg**.




AWS CLI

Untuk melihat semua peristiwa yang dihasilkan dalam satu jam terakhir, panggil describe-events tanpa parameter.

```
aws rds describe-events
```

Output sampel berikut menunjukkan bahwa instans klaster DB telah memulai pemulihan.

```
{
    "Events": [
        {
            "EventCategories": [
```

```
                "recovery"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
            "Date": "2022-04-20T15:02:38.416Z",
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

Untuk melihat semua RDS peristiwa Amazon selama 10080 menit terakhir (7 hari), panggil AWS CLI perintah deskripsikan peristiwa dan atur parameternya ke. `--duration 10080`

```
aws rds describe-events --duration 10080
```

Contoh berikut menunjukkan peristiwa dalam rentang waktu yang ditentukan untuk contoh DB *`test-instance`*.

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

Output sampel berikut menampilkan status cadangan.

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
```

```
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
    ]
}
```

API

Anda dapat melihat semua peristiwa RDS instans Amazon selama 14 hari terakhir dengan memanggil DescribeEventsRDSAPIoperasi dan menyetel `Duration` parameternya`20160`.

# Bekerja dengan pemberitahuan RDS acara Amazon

Amazon RDS menggunakan Amazon Simple Notification Service (AmazonSNS) untuk memberikan pemberitahuan ketika RDS peristiwa Amazon terjadi. Pemberitahuan ini dapat dalam bentuk pemberitahuan apa pun yang didukung oleh Amazon SNS untuk suatu AWS Wilayah, seperti email, pesan teks, atau panggilan ke HTTP titik akhir.

Topik


- Ikhtisar pemberitahuan RDS acara Amazon
- Memberikan izin untuk menerbitkan pemberitahuan ke topik Amazon SNS
- Berlangganan notifikasi RDS acara Amazon
- Tag dan atribut pemberitahuan RDS acara Amazon
- Daftar langganan pemberitahuan RDS acara Amazon
- Memodifikasi langganan pemberitahuan RDS acara Amazon
- Menambahkan pengenal sumber ke langganan pemberitahuan RDS acara Amazon
- Menghapus pengenal sumber dari langganan pemberitahuan RDS acara Amazon
- Daftar kategori pemberitahuan RDS acara Amazon
- Menghapus langganan pemberitahuan RDS acara Amazon


## Ikhtisar pemberitahuan RDS acara Amazon

Amazon RDS mengelompokkan acara ke dalam kategori yang dapat Anda berlangganan sehingga Anda dapat diberi tahu saat peristiwa dalam kategori tersebut terjadi.

Topik


- RDSsumber daya yang memenuhi syarat untuk berlangganan acara
- Proses dasar untuk berlangganan notifikasi RDS acara Amazon
- Pengiriman pemberitahuan RDS acara
- Penagihan untuk pemberitahuan RDS acara Amazon
- Contoh RDS acara Aurora Amazon EventBridge

RDSsumber daya yang memenuhi syarat untuk berlangganan acara

Untuk Amazon Aurora, peristiwa terjadi baik di klaster DB maupun tingkat instans DB. Anda dapat berlangganan kategori peristiwa untuk sumber daya berikut:

- Instans DB
- Klaster DB
- Snapshot klaster DB
- Grup parameter DB
- Grup keamanan DB
- RDSProksi
- Versi mesin kustom

Misalnya, jika berlangganan kategori pencadangan untuk instans DB tertentu, Anda akan diberi tahu setiap kali ada peristiwa terkait pencadangan yang memengaruhi instans DB. Jika berlangganan kategori perubahan konfigurasi untuk instans DB, Anda akan diberi tahu saat instans DB diubah. Anda juga menerima pemberitahuan saat langganan pemberitahuan peristiwa berubah.

Sebaiknya Anda membuat beberapa langganan yang berbeda. Misalnya, Anda dapat membuat satu langganan yang menerima semua pemberitahuan peristiwa untuk semua instans DB dan langganan lain yang hanya mencakup peristiwa penting untuk sebagian instans DB. Untuk langganan kedua, tentukan satu atau beberapa instans DB dalam filter.

Proses dasar untuk berlangganan notifikasi RDS acara Amazon

Proses berlangganan notifikasi RDS acara Amazon adalah sebagai berikut:

1. Anda membuat langganan notifikasi RDS acara Amazon dengan menggunakan RDS konsol Amazon, AWS CLI, atauAPI.

   Amazon RDS menggunakan SNS topik Amazon untuk mengidentifikasi setiap langganan. ARN RDSKonsol Amazon membuat ARN untuk Anda saat Anda membuat langganan. Buat ARN dengan menggunakan SNS konsol Amazon, Amazon AWS CLI, atau Amazon SNSAPI.
2. Amazon RDS mengirimkan email persetujuan atau SMS pesan ke alamat yang Anda kirimkan dengan langganan Anda.
3. Anda mengonfirmasi langganan Anda dengan memilih tautan di pemberitahuan yang Anda terima.

4. RDSKonsol Amazon memperbarui bagian Langganan Acara Saya dengan status langganan Anda.
5. Amazon RDS mulai mengirimkan notifikasi ke alamat yang Anda berikan saat Anda membuat langganan.

Untuk mempelajari manajemen identitas dan akses saat menggunakan AmazonSNS, lihat Manajemen identitas dan akses di Amazon SNS di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Anda dapat menggunakan AWS Lambda untuk memproses pemberitahuan acara dari instans DB. Untuk informasi selengkapnya, lihat Menggunakan AWS Lambda dengan Amazon RDS di Panduan AWS Lambda Pengembang.

Pengiriman pemberitahuan RDS acara

Amazon RDS mengirimkan pemberitahuan ke alamat yang Anda berikan saat Anda membuat langganan. Pemberitahuan dapat mencakup atribut pesan yang berisi metadata terstruktur tentang pesan tersebut. Untuk mengetahui informasi selengkapnya tentang atribut pesan, lihat Kategori acara Amazon RDS dan pesan acara untuk Aurora.

Pemberitahuan peristiwa mungkin memerlukan waktu hingga lima menit untuk dikirimkan.

> **Important**
>
> Amazon RDS tidak menjamin urutan acara yang dikirim dalam aliran acara. Urutan peristiwa dapat berubah sewaktu-waktu.

Saat Amazon SNS mengirimkan notifikasi ke langganan HTTP atau HTTPS titik akhir, POST pesan yang dikirim ke titik akhir memiliki badan pesan yang berisi dokumen. JSON Untuk informasi selengkapnya, lihat SNSpesan dan JSON format Amazon di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Anda dapat mengonfigurasi SNS untuk memberi tahu Anda dengan pesan teks. Untuk informasi selengkapnya, lihat Pesan teks seluler (SMS) di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Untuk mematikan notifikasi tanpa menghapus langganan, pilih Tidak untuk Diaktifkan di RDS konsol Amazon. Atau Anda dapat mengatur `Enabled` parameter untuk `false` menggunakan AWS CLI atau Amazon RDSAPI.

### Penagihan untuk pemberitahuan RDS acara Amazon

Penagihan untuk pemberitahuan RDS acara Amazon adalah melalui AmazonSNS. SNSBiaya Amazon berlaku saat menggunakan pemberitahuan acara. Untuk informasi selengkapnya tentang SNS penagihan Amazon, lihat harga Layanan Pemberitahuan Sederhana Amazon.

### Contoh RDS acara Aurora Amazon EventBridge

Contoh berikut menggambarkan berbagai jenis JSON peristiwa  Amazon dalam format. Untuk tutorial yang menunjukkan cara menangkap dan melihat peristiwa dalam JSON format, lihatTutorial: Log perubahan status instans DB menggunakan Amazon EventBridge.

Topik

- Contoh peristiwa klaster DB
- Contoh peristiwa grup parameter DB
- Contoh peristiwa snapshot klaster DB

#### Contoh peristiwa klaster DB

Berikut ini adalah contoh peristiwa cluster DB dalam JSON format. Peristiwa ini menunjukkan bahwa klaster bernama `my-db-cluster` di-patch. ID peristiwanya adalah `RDS-EVENT-0173`.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
  "detail": {
    "EventCategories": [
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
```

```
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

Contoh peristiwa grup parameter DB

Berikut ini adalah contoh peristiwa grup parameter DB dalam JSON format. Peristiwa ini menunjukkan bahwa parameter `time_zone` telah diperbarui dalam grup parameter `my-db-param-group`. ID acara adalah RDS - EVENT -0037.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

Contoh peristiwa snapshot klaster DB

Berikut ini adalah contoh peristiwa snapshot cluster DB dalam JSON format. Peristiwa ini menunjukkan pembuatan snapshot bernama `my-db-cluster-snapshot`. ID acara adalah RDS - EVENT -0074.

```
{
```

```
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
}
```

## Memberikan izin untuk menerbitkan pemberitahuan ke topik Amazon SNS

Untuk memberikan izin Amazon RDS untuk memublikasikan notifikasi ke topik Simple Notification Service Amazon (Amazon SNS), lampirkan AWS Identity and Access Management kebijakan (IAM) ke topik tujuan. Untuk mengetahui informasi selengkapnya tentang izin, lihat Contoh kasus untuk kontrol akses Amazon Simple Notification Service di Panduan Developer Amazon Simple Notification Service.

Secara default, topik Amazon SNS memiliki kebijakan yang mengizinkan semua sumber daya Amazon RDS dalam akun yang sama untuk menerbitkan pemberitahuan ke akun tersebut. Anda dapat melampirkan kebijakan kustom untuk mengizinkan pemberitahuan lintas akun, atau untuk membatasi akses ke sumber daya tertentu.

Berikut ini adalah contoh kebijakan IAM yang Anda lampirkan ke topik Amazon SNS tujuan. Ini membatasi topik ke instans DB dengan nama yang cocok dengan awalan yang ditentukan. Untuk menggunakan kebijakan ini, tentukan nilai berikut:

- `Resource` – Amazon Resource Name (ARN) untuk topik Amazon SNS Anda
- `SourceARN` – ARN sumber data RDS Anda
- `SourceAccount`— Akun AWS ID Anda

Untuk melihat daftar jenis sumber daya dan jenis sumber daya ARNs, lihat Sumber Daya yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
```

```
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

## Berlangganan notifikasi RDS acara Amazon

Cara paling sederhana untuk membuat langganan adalah dengan RDS konsol. Jika Anda memilih untuk membuat langganan pemberitahuan acara menggunakan CLI atauAPI, Anda harus membuat topik Layanan Pemberitahuan Sederhana Amazon dan berlangganan topik tersebut dengan SNS konsol Amazon atau Amazon SNSAPI. Anda juga perlu mempertahankan Amazon Resource Name (ARN) dari topik karena digunakan saat mengirimkan CLI perintah atau API operasi. Untuk informasi tentang membuat SNS topik dan berlangganannya, lihat Memulai Amazon SNS di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Anda dapat menentukan jenis sumber yang ingin Anda beri tahu dan RDS sumber Amazon yang memicu peristiwa:

Jenis sumber

Jenis sumber. Misalnya, Jenis sumber mungkin berupa Instans. Anda harus memilih jenis sumber.

***Resources*** untuk memasukkan

RDSSumber daya Amazon yang menghasilkan acara. Misalnya, Anda dapat memilih Pilih instans tertentu dan kemudian myDBInstance1.

Tabel berikut menjelaskan hasil ketika Anda menentukan atau tidak menentukan ***Resources*** untuk memasukkan.

| Sumber daya yang akan disertakan | Deskripsi | Contoh |
| --- | --- | --- |
| Ditentukan | RDSmemberi tahu Anda tentang semua peristiwa untuk sumber daya yang ditentukan saja. | Jika tipe Sumber Anda adalah Instans dan sumber daya Anda adalah myDBInstance1, akan RDS memberi tahu Anda tentang semua peristiwa hanya untuk myDBInstance 1. |
| Tidak ditentukan | RDSmemberi tahu Anda tentang peristiwa untuk jenis sumber yang ditentukan untuk semua sumber RDS daya Amazon Anda. | Jika tipe Sumber Anda adalah Instans, akan RDS memberi tahu |

| Sumber daya yang akan disertakan | Deskripsi | Contoh |
| --- | --- | --- |
| | | Anda tentang semua peristiwa terkait instans di akun Anda. |

Pelanggan SNS topik Amazon menerima setiap pesan yang dipublikasikan ke topik secara default. Untuk menerima subset pesan saja, pelanggan harus menetapkan kebijakan filter untuk langganan topik. Untuk informasi selengkapnya tentang pemfilteran SNS pesan, lihat Pemfilteran SNS pesan Amazon di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon

Konsol

Untuk berlangganan pemberitahuan RDS acara

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan peristiwa.
3. Di panel Langganan peristiwa, pilih Buat langganan peristiwa.
4. Masukkan detail langganan Anda sebagai berikut:

    a. Untuk Nama, masukkan nama langganan pemberitahuan peristiwa.

    b. Untuk Kirim pemberitahuan ke, lakukan salah satu langkah berikut:

    - Pilih Topik email baru. Masukkan nama topik email Anda dan daftar penerima. Sebaiknya Anda mengonfigurasi langganan peristiwa ke alamat email yang sama dengan kontak akun utama Anda. Rekomendasi, peristiwa layanan, dan pesan kondisi pribadi dikirim menggunakan saluran yang berbeda. Langganan ke alamat email yang sama memastikan bahwa semua pesan digabungkan di satu lokasi.
    - Pilih Nama Sumber Daya Amazon (ARN). Kemudian pilih Amazon yang ada SNS ARN untuk SNS topik Amazon.

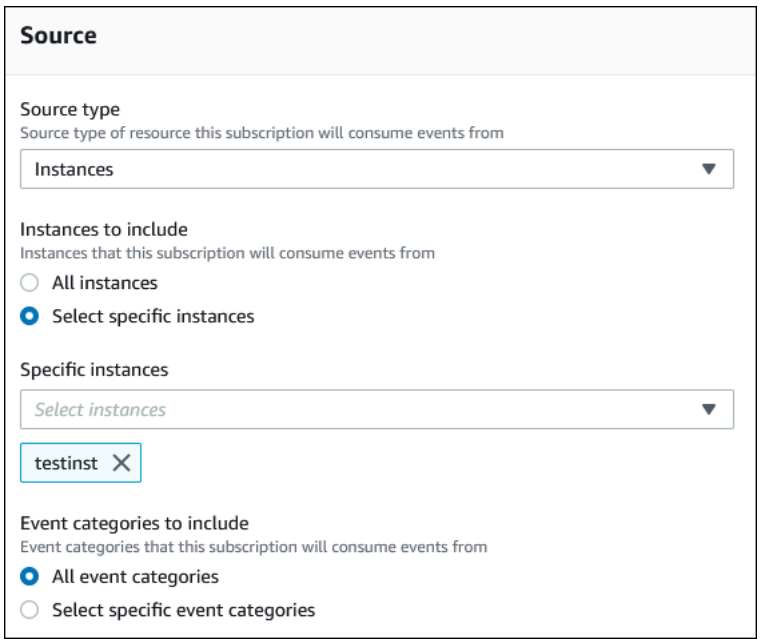
      Jika Anda ingin menggunakan topik yang telah diaktifkan untuk enkripsi sisi server (SSE), berikan Amazon izin RDS yang diperlukan untuk mengakses. AWS KMS key Untuk informasi selengkapnya, lihat Mengaktifkan kompatibilitas antara sumber peristiwa dari

AWS layanan dan topik terenkripsi di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

c. Untuk Jenis sumber, pilih jenis sumber. Misalnya, pilih Klaster atau Snapshot klaster.

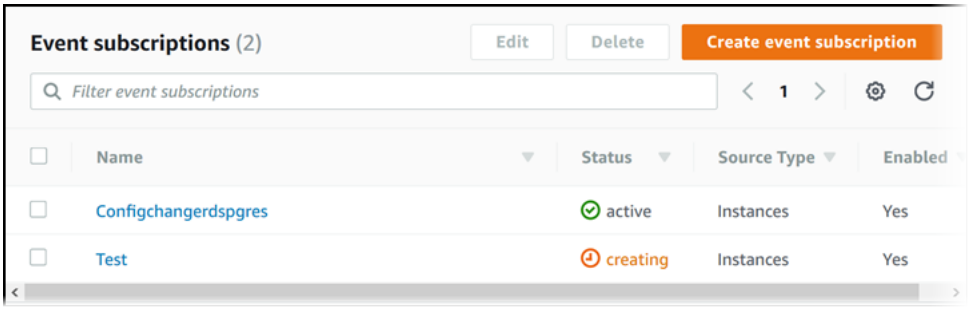
d. Pilih kategori dan sumber daya peristiwa yang pemberitahuan peristiwanya ingin diterima.

Contoh berikut mengonfigurasi pemberitahuan peristiwa untuk instans DB bernama `testinst`.




e. Pilih Buat.

RDSKonsol Amazon menunjukkan bahwa langganan sedang dibuat.




AWS CLI

Untuk berlangganan pemberitahuan RDS acara, gunakan AWS CLI `create-event-subscription`perintah. Sertakan parameter-parameter yang diperlukan berikut:

- `--subscription-name`

- `--sns-topic-arn`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-event-subscription \
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

Untuk Windows:

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

Untuk berlangganan pemberitahuan RDS acara Amazon, hubungi RDS API fungsi Amazon `CreateEventSubscription`. Sertakan parameter-parameter yang diperlukan berikut:

- `SubscriptionName`
- `SnsTopicArn`

## Tag dan atribut pemberitahuan RDS acara Amazon

Saat Amazon RDS mengirimkan pemberitahuan peristiwa ke Amazon Simple Notification Service (SNS) atau Amazon EventBridge, notifikasi tersebut berisi atribut pesan dan tag peristiwa. RDSmengirim atribut pesan secara terpisah bersama dengan pesan, sementara tag peristiwa berada di badan pesan. Gunakan atribut pesan dan RDS tag Amazon untuk menambahkan metadata ke sumber daya Anda. Anda dapat mengubah tag ini dengan notasi Anda sendiri tentang instans DB, klaster Aurora, dan sebagainya. Untuk informasi selengkapnya tentang menandai RDS sumber daya Amazon, lihatMenandai sumber daya Amazon Aurora dan Amazon RDS.

Secara default, Amazon SNS dan Amazon EventBridge menerima setiap pesan yang dikirim kepada mereka. SNSdan EventBridge dapat memfilter pesan dan mengirim notifikasi ke mode komunikasi pilihan, seperti email, pesan teks, atau panggilan ke HTTP titik akhir.

> **Note**
>
> Pemberitahuan yang dikirim dalam email atau pesan teks tidak akan memiliki tag peristiwa.

Tabel berikut menunjukkan atribut pesan untuk RDS peristiwa yang dikirim ke pelanggan topik.

| Atribut RDS acara Amazon | Deskripsi |
| --- | --- |
| EventID | Pengidentifikasi untuk pesan RDS acara, misalnya, RDS - EVENT -0006. |
| Sumber Daya | ARNPengidentifikasi untuk sumber daya yang memancarkan acara, misalnya,. `arn:aws:rds:ap-southeast-2:123456789012:db:database-1` |

RDSTag menyediakan data tentang sumber daya yang dipengaruhi oleh peristiwa layanan. RDSmenambahkan status tag saat ini di badan pesan saat pemberitahuan dikirim ke SNS atau EventBridge.

Untuk informasi selengkapnya tentang memfilter atribut pesanSNS, lihat Pemfilteran SNS pesan Amazon di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Untuk informasi selengkapnya tentang memfilter tag peristiwa EventBridge, lihat Operator perbandingan untuk digunakan dalam pola peristiwa di Amazon EventBridge di Panduan EventBridge Pengguna Amazon.

Untuk informasi selengkapnya tentang memfilter tag berbasis payloadSNS, lihat Memperkenalkan pemfilteran pesan berbasis payload untuk Amazon SNS

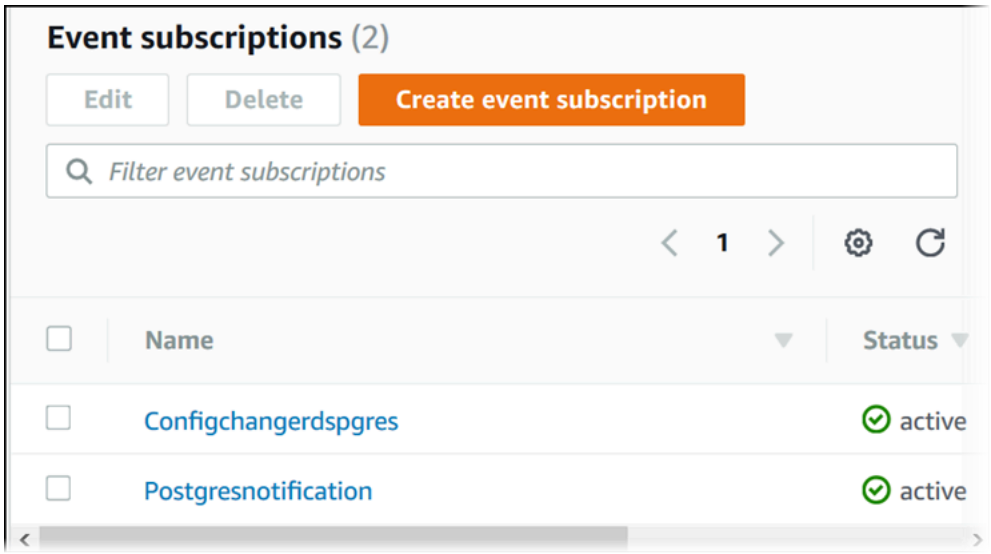
## Daftar langganan pemberitahuan RDS acara Amazon

Anda dapat mencantumkan langganan pemberitahuan RDS acara Amazon Anda saat ini.

### Konsol

Untuk mencantumkan langganan pemberitahuan RDS acara Amazon Anda saat ini

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Berlangganan peristiwa. Panel Langganan peristiwa menampilkan semua langganan pemberitahuan peristiwa.




### AWS CLI

Untuk mencantumkan langganan notifikasi RDS acara Amazon Anda saat ini, gunakan AWS CLI `describe-event-subscriptions`perintah.

### Example

Contoh berikut menjelaskan semua langganan peristiwa.

```
aws rds describe-event-subscriptions
```

Contoh berikut menjelaskan `myfirsteventsubscription`.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

Untuk mencantumkan langganan pemberitahuan RDS acara Amazon Anda saat ini, hubungi RDS API `DescribeEventSubscriptions`tindakan Amazon.

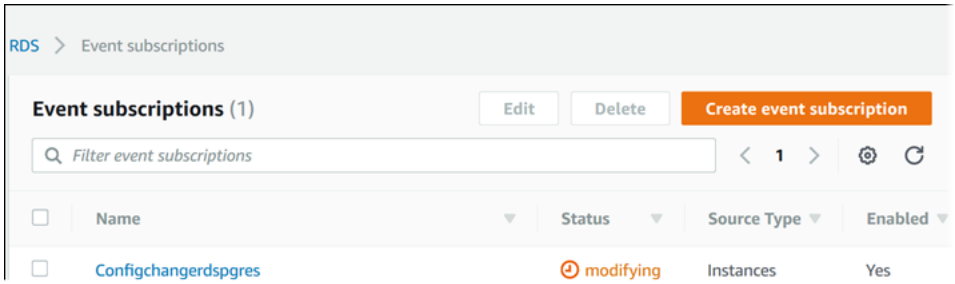
## Memodifikasi langganan pemberitahuan RDS acara Amazon

Setelah membuat langganan, Anda dapat mengubah nama langganan, pengenal sumber, kategori, atau topikARN.

### Konsol

Untuk mengubah langganan pemberitahuan RDS acara Amazon

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan peristiwa.
3. Di panel Langganan peristiwa, pilih langganan yang ingin diubah, lalu pilih Edit.
4. Buat perubahan pada langganan di bagian Target atau Sumber.
5. Pilih Edit. RDSKonsol Amazon menunjukkan bahwa langganan sedang dimodifikasi.




### AWS CLI

Untuk mengubah langganan pemberitahuan RDS acara Amazon, gunakan AWS CLI `modify-event-subscription`perintah. Sertakan parameter wajib berikut:

- `--subscription-name`

### Example

Kode berikut mengaktifkan `myeventsubscription`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
```

```
    --enabled
```

Untuk Windows:

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

Untuk memodifikasi RDS acara Amazon, hubungi RDS API operasi Amazon `ModifyEventSubscription`. Sertakan parameter wajib berikut:

- `SubscriptionName`

## Menambahkan pengenal sumber ke langganan pemberitahuan RDS acara Amazon

Anda dapat menambahkan pengenal sumber (RDSsumber Amazon yang menghasilkan acara) ke langganan yang ada.

### Konsol

Anda dapat dengan mudah menambah atau menghapus pengenal sumber menggunakan RDS konsol Amazon dengan memilih atau membatalkan pilihnya saat memodifikasi langganan. Untuk informasi selengkapnya, lihat Memodifikasi langganan pemberitahuan RDS acara Amazon.

### AWS CLI

Untuk menambahkan pengenal sumber ke langganan pemberitahuan RDS acara Amazon, gunakan AWS CLI `add-source-identifier-to-subscription`perintah. Sertakan parameter-parameter yang diperlukan berikut:

- `--subscription-name`
- `--source-identifier`

#### Example

Contoh berikut menambahkan pengidentifikasi sumber `mysqldb` ke langganan `myrdseventsubscription`.

Untuk Linux, macOS, atau Unix:

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Untuk Windows:

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Untuk menambahkan pengenal sumber ke langganan pemberitahuan RDS acara Amazon, hubungi Amazon RDS API `AddSourceIdentifierToSubscription`. Sertakan parameter-parameter yang diperlukan berikut:

- `SubscriptionName`
- `SourceIdentifier`

## Menghapus pengenal sumber dari langganan pemberitahuan RDS acara Amazon

Anda dapat menghapus pengenal sumber (RDSsumber Amazon yang menghasilkan acara) dari langganan jika Anda tidak lagi ingin diberi tahu tentang peristiwa untuk sumber tersebut.

Konsol

Anda dapat dengan mudah menambah atau menghapus pengenal sumber menggunakan RDS konsol Amazon dengan memilih atau membatalkan pilihnya saat memodifikasi langganan. Untuk informasi selengkapnya, lihat Memodifikasi langganan pemberitahuan RDS acara Amazon.

AWS CLI

Untuk menghapus pengenal sumber dari langganan pemberitahuan RDS peristiwa Amazon, gunakan AWS CLI `remove-source-identifier-from-subscription`perintah. Sertakan parameter-parameter yang diperlukan berikut:

- `--subscription-name`
- `--source-identifier`

Example

Contoh berikut menghapus pengidentifikasi sumber `mysqldb` dari langganan `myrdseventsubscription`.

Untuk Linux, macOS, atau Unix:

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Untuk Windows:

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Untuk menghapus pengenal sumber dari langganan pemberitahuan RDS acara Amazon, gunakan RDS API `RemoveSourceIdentifierFromSubscription`perintah Amazon. Sertakan parameter-parameter yang diperlukan berikut:

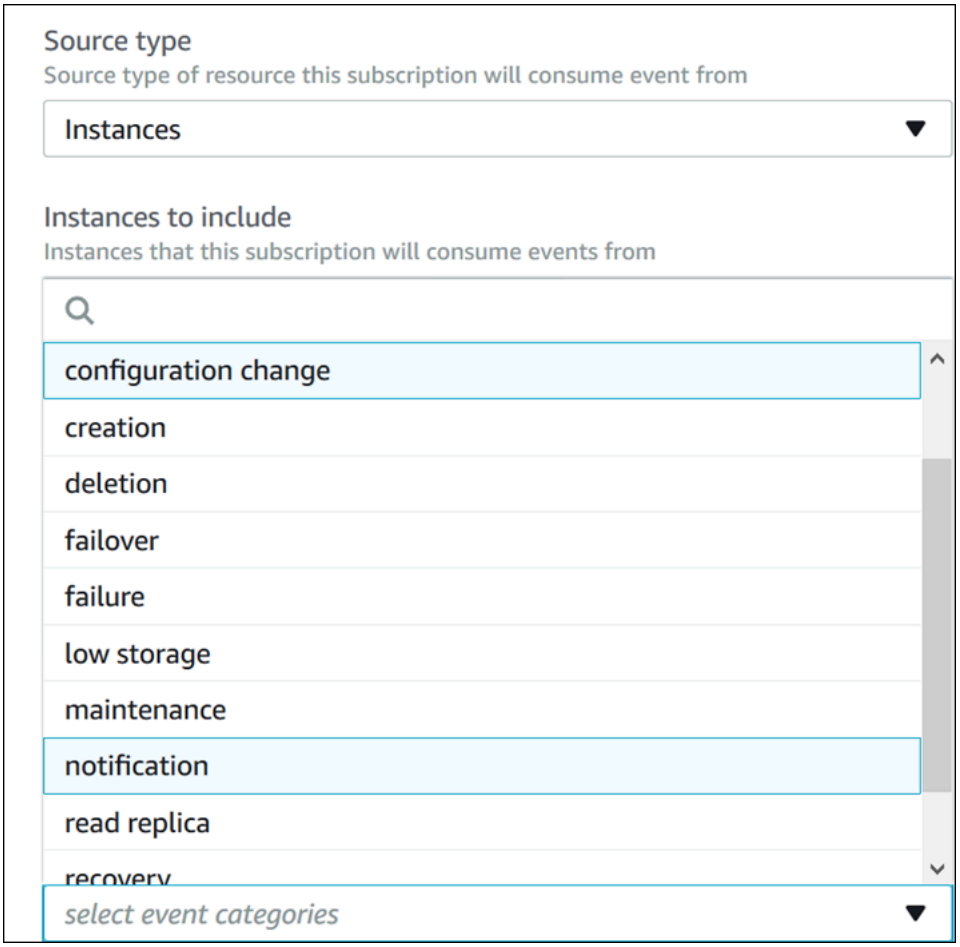
- `SubscriptionName`
- `SourceIdentifier`

## Daftar kategori pemberitahuan RDS acara Amazon

Semua peristiwa untuk jenis sumber daya dikelompokkan dalam beberapa kategori. Untuk melihat daftar kategori yang tersedia, gunakan prosedur berikut.

### Konsol

Saat Anda membuat atau memodifikasi langganan pemberitahuan acara, kategori acara akan ditampilkan di RDS konsol Amazon. Untuk informasi selengkapnya, lihat Memodifikasi langganan pemberitahuan RDS acara Amazon.




### AWS CLI

Untuk mencantumkan kategori notifikasi RDS acara Amazon, gunakan AWS CLI `describe-event-categories`perintah. Perintah ini tidak memiliki parameter yang diperlukan.

### Example

```
aws rds describe-event-categories
```

### API

Untuk mencantumkan kategori notifikasi RDS acara Amazon, gunakan RDS API `DescribeEventCategories`perintah Amazon. Perintah ini tidak memiliki parameter yang diperlukan.

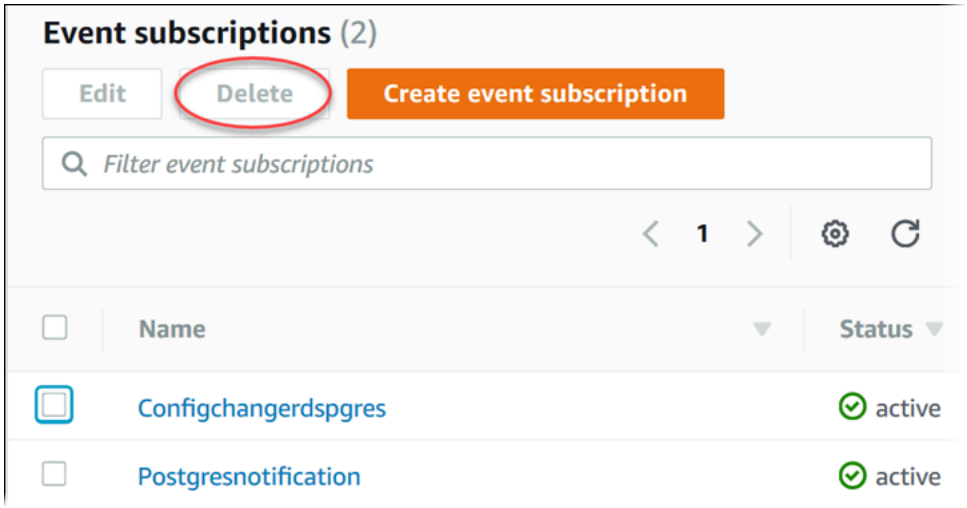
## Menghapus langganan pemberitahuan RDS acara Amazon

Anda dapat menghapus langganan jika sudah tidak membutuhkannya lagi. Semua pelanggan topik tidak akan lagi menerima pemberitahuan peristiwa yang ditentukan oleh langganan.

### Konsol

Untuk menghapus langganan pemberitahuan RDS acara Amazon

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan Peristiwa DB.
3. Di panel Langganan Peristiwa DB Saya, pilih langganan yang ingin dihapus.
4. Pilih Hapus.
5. RDSKonsol Amazon menunjukkan bahwa langganan sedang dihapus.




### AWS CLI

Untuk menghapus langganan pemberitahuan RDS acara Amazon, gunakan AWS CLI `delete-event-subscription`perintah. Sertakan parameter wajib berikut:

- `--subscription-name`

### Example

Contoh berikut menghapus langganan `myrdssubscription`.

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

Untuk menghapus langganan pemberitahuan RDS acara Amazon, gunakan RDS API `DeleteEventSubscription`perintah. Sertakan parameter wajib berikut:

- `SubscriptionName`

# Membuat aturan yang dipicu pada peristiwa Amazon Aurora

Menggunakan Amazon EventBridge, Anda dapat mengotomatiskan AWS layanan dan menanggapi peristiwa sistem seperti masalah ketersediaan aplikasi atau perubahan sumber daya.

Topik


- Tutorial: Log perubahan status instans DB menggunakan Amazon EventBridge


## Tutorial: Log perubahan status instans DB menggunakan Amazon EventBridge

Dalam tutorial ini, Anda membuat AWS Lambda fungsi yang mencatat perubahan status untuk instance . Kemudian, Anda dapat membuat aturan yang menjalankan fungsi setiap kali terjadi perubahan status instans DB RDS yang ada. Tutorial ini mengasumsikan bahwa Anda memiliki instans pengujian kecil yang sedang berjalan yang dapat ditutup sementara.

> **Important**
>
> Jangan lakukan tutorial ini pada instans DB produksi yang sedang berjalan.

Topik


- Langkah 1: Buat AWS Lambda fungsi
- Langkah 2: Buat Aturan
- Langkah 3: Uji aturan


### Langkah 1: Buat AWS Lambda fungsi

Buat fungsi Lambda untuk mencatat peristiwa perubahan status. Tetapkan fungsi ini saat membuat aturan.

Untuk membuat fungsi Lambda

1. Buka AWS Lambda konsol di https://console.aws.amazon.com/lambda/.
2. Jika baru menggunakan Lambda, Anda akan melihat halaman selamat datang. Pilih Mulai Sekarang. Atau, pilih Buat fungsi.
3. Pilih Tulis dari awal.
4. Di halaman Buat fungsi, lakukan langkah berikut:

a. Masukkan nama dan deskripsi untuk fungsi Lambda. Misalnya, beri nama fungsi **`RDSInstanceStateChange`**.

b. Dalam Runtime, pilih Node.js 16x.

c. Untuk Arsitektur, pilih x86_64.

d. Untuk Peran eksekusi, lakukan salah satu langkah berikut:

- Pilih Buat peran baru dengan izin Lambda dasar.
- Untuk Peran yang sudah ada, pilih Gunakan peran yang sudah ada. Pilih peran yang ingin digunakan.

e. Pilih Buat fungsi.

5. Pada InstanceStateChange halaman RDS, lakukan hal berikut:

a. Di Sumber kode, pilih index.js.

b. Di panel index.js, hapus kode yang ada.

c. Masukkan kode berikut:

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

d. Pilih Deploy.

Langkah 2: Buat Aturan

Buat aturan untuk menjalankan fungsi Lambda setiap kali Anda meluncurkan instans Amazon RDS.

Untuk membuat EventBridge aturan

1. Buka EventBridge konsol Amazon di https://console.aws.amazon.com/events/.
2. Di panel navigasi, pilih Aturan.
3. Pilih Buat aturan.
4. Masukkan nama dan deskripsi aturan. Misalnya, masukkan **`RDSInstanceStateChangeRule`**.
5. Pilih Aturan dengan pola peristiwa, lalu pilih Berikutnya.
6. Untuk sumber Acara, pilih AWS acara atau acara EventBridge mitra.

7. Gulir ke bawah ke bagian Pola peristiwa.
8. Untuk Sumber peristiwa, pilih Layanan AWS.
9. Untuk layanan AWS , pilih Layanan Basis Data Relasional (RDS).
10. Untuk Jenis peristiwa, pilih Peristiwa Instans DB RDS.
11. Biarkan pola peristiwa default. Lalu pilih Selanjutnya.
12. Untuk Jenis target, pilih Layanan AWS .
13. Untuk Pilih target, pilih Fungsi Lambda.
14. Untuk Fungsi, pilih fungsi Lambda yang Anda buat. Lalu pilih Selanjutnya.
15. Di Konfigurasikan tag, pilih Berikutnya.
16. Tinjau langkah-langkah dalam aturan Anda. Kemudian, pilih Buat aturan.

### Langkah 3: Uji aturan

Untuk menguji aturan, tutup instans DB RDS. Setelah menunggu beberapa menit untuk instans yang akan dinonaktifkan, verifikasi bahwa fungsi Lambda Anda sudah diinvokasi.

Untuk menguji aturan dengan menghentikan instans DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Hentikan suatu instans DB RDS.
3. Buka EventBridge konsol Amazon di https://console.aws.amazon.com/events/.
4. Di panel navigasi, pilih Aturan, pilih nama aturan yang Anda buat.
5. Dalam Detail aturan, pilih Pemantauan.

   Anda dialihkan ke CloudWatch konsol Amazon. Jika Anda tidak dialihkan, klik Lihat metrik di. CloudWatch
6. Dalam Semua metrik, pilih nama aturan yang Anda buat.

   Grafik harus menunjukkan bahwa aturan diinvokasi.
7. Di panel navigasi, pilih Grup log.
8. Pilih nama grup log untuk fungsi Lambda Anda (/aws/lambda/*function-name*).
9. Pilih nama log stream untuk melihat data yang disediakan oleh fungsi untuk instans yang Anda luncurkan. Anda akan melihat peristiwa yang mirip dengan yang berikut ini:

```
{
```

```
    "version": "0",
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```


Untuk contoh peristiwa RDS lain dalam format JSON, lihat .

10. (Opsional) Setelah selesai, Anda dapat membuka konsol Amazon RDS dan memulai instans yang Anda hentikan.

# Kategori acara Amazon RDS dan pesan acara untuk Aurora

Amazon RDS menghasilkan sejumlah besar peristiwa dalam kategori yang dapat Anda berlangganan menggunakan Konsol Amazon RDS, AWS CLI, atau API.

Topik


- Peristiwa klaster DB
- Peristiwa snapshot klaster DB
- Peristiwa instans DB
- Peristiwa grup parameter DB
- Peristiwa grup keamanan DB
- Acara grup shard DB
- Peristiwa Proksi RDS
- Peristiwa deployment blue/green


## Peristiwa klaster DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat klaster DB berupa jenis sumber.

> **Note**
>
> Tidak ada kategori peristiwa yang ada untuk Aurora Serverless dalam jenis peristiwa klaster DB. Peristiwa Aurora Nirserver berkisar antara RDS-EVENT-0141 hingga RDS-EVENT-0149.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| perubahan konfigurasi | RDS-EVENT-0016 | Atur ulang kredensial utama. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0179 | Aliran Aktivitas Basis Data dimulai pada klaster basis data Anda. | Untuk mengetahui informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| perubahan konfigurasi | RDS-EVENT-0180 | Aliran Aktivitas Basis Data dihentikan pada klaster basis data Anda. | Untuk mengetahui informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data. |
| pembuatan | RDS-EVENT-0170 | Klaster DB dibuat. | Tidak ada |
| penghapusan | RDS-EVENT-0171 | Klaster DB dihapus. | Tidak ada |
| failover | RDS-EVENT-0069 | Failover klaster gagal, periksa kondisi instans klaster dan coba lagi. | Tidak ada |
| failover | RDS-EVENT-0070 | Mempromosikan primer sebelumnya lagi:*name*. | Tidak ada |
| failover | RDS-EVENT-0071 | Failover selesai ke instans DB:*name*. | Tidak ada |
| failover | RDS-EVENT-0072 | Memulai failover AZ yang sama ke instans DB:*name*. | Tidak ada |
| failover | RDS-EVENT-0073 | Mulai melintasi failover AZ ke instans DB:*name*. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| kegagalan | RDS-EVENT-0083 | Amazon RDS tidak dapat membuat kredensil untuk mengakses Bucket Amazon S3 Anda untuk cluster DB Anda. *name* Hal ini disebabkan peran IAM konsumsi snapshot S3 tidak dikonfigurasi dengan benar di akun Anda atau bucket Amazon S3 yang ditentukan tidak dapat ditemukan. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS. | Untuk mengetahui informasi selengkap nya, lihat Migrasi fisik dari MySQL dengan menggunakan XtraBackup Percona dan Amazon S3 . |
| kegagalan | RDS-EVENT-0143 | Cluster DB gagal *units* menskalakan dari ke karena *units* alasan ini:*reason*. | Penskalaan gagal untuk klaster DB Aurora Serverless. |
| kegagalan | RDS-EVENT-0354 | Anda tidak dapat membuat cluster DB karena sumber daya yang tidak kompatibe l. *message*. | *message*Termasuk rincian tentang kegagalan. |
| kegagalan | RDS-EVENT-0355 | Cluster DB tidak dapat dibuat karena batas sumber daya yang tidak mencukupi. *message*. | *message*Termasuk rincian tentang kegagalan. |
| kegagalan | RDS-ACARA-0387 | Gagal mempartisi instance DB di cluster DB *name* untuk ditambal. | Upgrade sistem operasi untuk instans DB di cluster DB gagal. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| failover global | RDS-EVENT-0181 | Peralihan global ke cluster DB *name* di Wilayah *name* dimulai. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").<br><br>Prosesnya dapat ditunda karena operasi lain berjalan di klaster DB. |
| failover global | RDS-EVENT-0182 | Cluster DB primer lama *name* di Wilayah *name* berhasil ditutup. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").<br><br>Instans primer lama dalam basis data global tidak menerima penulisan. Semua volume disinkron kan. |
| failover global | RDS-EVENT-0183 | Menunggu sinkronisasi data di seluruh anggota klaster global. Arus tertinggal di belakang cluster DB primer:*reason*. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").<br><br>Kelambatan replikasi terjadi selama fase sinkronisasi failover basis data global. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| failover global | RDS-EVENT-0184 | Cluster DB primer baru *name* di Wilayah berhasil *name* dipromosikan. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").<br><br>Topologi volume basis data global dibangun kembali dengan volume primer baru. |
| failover global | RDS-EVENT-0185 | Peralihan global ke cluster DB *name* di Wilayah *name* selesai. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").<br><br>Switchover basis data global selesai di klaster DB primer. Replika mungkin membutuhkan waktu lama untuk online setelah failover selesai. |
| failover global | RDS-EVENT-0186 | Peralihan global ke cluster DB *name* di Wilayah dibatalkan*name*. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). |
| failover global | RDS-EVENT-0187 | Peralihan global ke cluster DB *name* di Wilayah *name* gagal. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). |
| failover global | RDS-EVENT-0238 | Failover global ke cluster DB *name* di Wilayah *name* selesai. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| failover global | RDS-EVENT-0239 | Failover global ke cluster DB *name* di Wilayah *name* gagal. | Tidak ada |
| failover global | RDS-EVENT-0240 | Mulai menyinkronkan ulang anggota cluster DB *name* di Wilayah *name* setelah failover global. | Tidak ada |
| failover global | RDS-EVENT-0241 | Selesai menyinkronkan ulang anggota cluster DB *name* di Wilayah *name* setelah failover global. | Tidak ada |
| failover global | RDS-ACARA-0397 | Aurora selesai mengubah nama DNS yang diselesai kan oleh titik akhir penulis global. | Tidak ada |
| failover global | RDS-ACAR-AVENT-0423 | Menunggu sinkronisasi data dengan cluster DB target. Cluster DB target saat ini tertinggal dari cluster DB primer:*%s*. | Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").<br><br>Kelambatan replikasi terjadi selama fase sinkronisasi failover basis data global. |
| pemeliharaan | RDS-EVENT-0156 | Klaster DB memiliki peningkatan versi minor mesin DB yang tersedia. | Tidak ada |
| pemeliharaan | RDS-EVENT-0173 | Versi mesin klaster basis data telah ditingkatkan. | Patching klaster DB telah selesai. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemeliharaan | RDS-ACARA-0174 | Database cluster berada dalam keadaan yang tidak dapat ditingkatkan. | Tidak ada |
| pemeliharaan | RDS-EVENT-0176 | Versi mayor mesin klaster basis data telah ditingkat kan. | Tidak ada |
| pemeliharaan | RDS-ACARA-0177 | Upgrade cluster database sedang berlangsung. | Tidak ada |
| pemeliharaan | RDS-EVENT-0286 | Upgrade *version_number* versi mesin cluster database dimulai. Cluster tetap online. | Tidak ada |
| pemeliharaan | RDS-EVENT-0287 | Persyaratan peningkatan sistem operasi terdeteksi. | Tidak ada |
| pemeliharaan | RDS-EVENT-0288 | Peningkatan sistem operasi klaster dimulai. | Tidak ada |
| pemeliharaan | RDS-EVENT-0289 | Peningkatan sistem operasi klaster selesai. | Tidak ada |
| pemeliharaan | RDS-EVENT-0290 | Kluster basis data telah ditambal: versi sumber *version_number* =>*new_version_number* . | Tidak ada |
| pemeliharaan | RDS-ACARA-0363 | Persiapan upgrade sedang berlangsung: *cluster_name* | Upgrade prechecks telah dimulai untuk cluster DB. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemeliharaan | RDS-ACARA-0388 | Memulai tambalan offline ke instance DB di *name* partisi/ *name* untuk cluster *name* DB:. *partition_n* | Memulai upgrade sistem operasi untuk instans DB di cluster DB. |
| pemeliharaan | RDS-ACARA-0389 | Kami tidak dapat memutakhirkan sistem operasi cluster DB Anda. Anda dapat menunggu jendela pemeliharaan berikutnya, atau Anda dapat memutakhirkan sistem operasi cluster DB Anda secara manual. | Tidak ada |
| pemeliharaan | RDS-ACAR-AVENT-0424 | Cluster *name* DB menjalankan versi*version*, yang lebih tinggi dari versi upgrade target *version* untuk cluster global. Kami tidak menyarankan memiliki cluster sekunder pada versi yang lebih tinggi daripada cluster global, karena dapat menyebabkan masalah selama failover atau switchover. Pertimbangkan untuk meningkatkan klaster global Anda agar sesuai. | Tidak ada |
| pemberitahuan | RDS-EVENT-0076 | Gagal bermigrasi dari *name* ke*name*. Alasan:*reason*. | Migrasi ke klaster DB Aurora gagal. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemberitahuan | RDS-EVENT-0077 | Gagal mengonversi*name*. *name*ke InnoDB. Alasan:*reason*. | Upaya untuk mengonversi tabel dari basis data sumber ke InnoDB gagal selama migrasi ke klaster DB Aurora. |
| pemberitahuan | RDS-EVENT-0085 | Tidak dapat memutakhirkan cluster DB *name* karena instance *name* memiliki status*name*. Selesaikan masalah atau hapus instans dan coba lagi. | Terjadi kesalahan saat mencoba untuk melakukan patch pada klaster DB Aurora. Periksa status instans Anda, selesaikan masalah, dan coba lagi. Untuk mengetahui informasi selengkapnya, lihat Memelihara klaster DB Amazon Aurora. |
| pemberitahuan | RDS-EVENT-0141 | Menskalakan cluster DB dari *units* ke karena *units* alasan ini:*reason*. | Penskalaan dimulai untuk klaster DB Aurora Serverless. |
| pemberitahuan | RDS-EVENT-0142 | Cluster DB telah diskalakan dari *units* ke*units*. | Penskalaan selesai untuk klaster DB Aurora Serverless. |
| pemberitahuan | RDS-EVENT-0144 | Klaster DB sedang dijeda. | Jeda otomatis dimulai untuk klaster DB Aurora Nirserver. |
| pemberitahuan | RDS-EVENT-0145 | Klaster DB dijeda. | Klaster DB Aurora Serverless telah dijeda. |
| pemberitahuan | RDS-EVENT-0146 | Jeda dibatalkan untuk klaster DB. | Jeda dibatalkan untuk klaster DB Aurora Serverless. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| pemberitahuan | RDS-EVENT-0147 | Klaster DB sedang dilanjutkan kembali. | Operasi resume dimulai untuk klaster DB Aurora Serverless. |
| pemberitahuan | RDS-EVENT-0148 | Klaster DB dilanjutkan kembali. | Operasi resume selesai untuk klaster DB Aurora Serverless. |
| pemberitahuan | RDS-EVENT-0149 | Cluster DB telah diskalakan dari *units* ke*units*, tetapi penskalaan tidak mulus karena alasan ini:. *reason* | Penskalaan lancar selesai dengan opsi paksa untuk klaster DB Aurora Serverless. Koneksi mungkin terganggu sesuai kebutuhan. |
| pemberitahuan | RDS-EVENT-0150 | Klaster DB dihentikan. | Tidak ada |
| pemberitahuan | RDS-EVENT-0151 | Klaster DB dimulai. | Tidak ada |
| pemberitahuan | RDS-EVENT-0152 | Perhentian klaster DB gagal. | Tidak ada |
| pemberitahuan | RDS-EVENT-0153 | Klaster DB sedang dimulai karena melebihi waktu maksimal yang diizinkan untuk dihentikan. | Tidak ada |
| pemberitahuan | RDS-EVENT-0172 | Berganti nama menjadi cluster dari *name* ke*name*. | Tidak ada |
| pemberitahuan | RDS-EVENT-0234 | Tugas ekspor gagal. | Tugas ekspor klaster DB gagal. |
| pemberitahuan | RDS-EVENT-0235 | Tugas ekspor dibatalkan. | Tugas ekspor klaster DB dibatalkan. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemberitahuan | RDS-EVENT-0236 | Tugas ekspor selesai. | Tugas ekspor klaster DB selesai. |
| pemberitahuan | RDS-ACARA-0386 | Instans DB di cluster DB *name* telah dipartisi:. *list_of_partitions* Cluster DB sedang online. | Upgrade sistem operasi untuk instans DB di cluster DB berhasil. |
| pemberitahuan | RDS-ACAR-AVENT-0426 | RDS tidak dapat mengonfigurasi cluster DB *name* sebagai sumber replikasi karena transaksi idle atau berjalan lama. Tunggu hingga transaksi selesai atau batalkan, dan coba lagi. | Tidak ada |

## Peristiwa snapshot klaster DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat snapshot klaster DB merupakan jenis sumber.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pencadangan | RDS-EVENT-0074 | Membuat snapshot klaster manual. | Tidak ada |
| pencadangan | RDS-EVENT-0075 | Snapshot klaster manual dibuat. | Tidak ada |
| pemberitahuan | RDS-EVENT-0162 | Tugas ekspor snapshot klaster gagal. | Tidak ada |
| pemberitahuan | RDS-EVENT-0163 | Tugas ekspor snapshot klaster dibatalkan. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemberitahuan | RDS-EVENT-0164 | Tugas ekspor snapshot klaster selesai. | Tidak ada |
| pencadangan | RDS-EVENT-0168 | Membuat snapshot klaster otomatis. | Tidak ada |
| pencadangan | RDS-EVENT-0169 | Snapshot klaster otomatis dibuat. | Tidak ada |

## Peristiwa instans DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat instans DB merupakan jenis sumber.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| ketersediaan | RDS-EVENT-0004 | Instans DB dimatikan. | Tidak ada |
| ketersediaan | RDS-EVENT-0006 | Instans DB dimulai ulang. | Tidak ada |
| ketersediaan | RDS-EVENT-0022 | Kesalahan saat memulai ulang mysql:. *`message`* | Terjadi kesalahan saat memulai ulang Aurora MySQL atau RDS for MariaDB. |
| ketersediaan | RDS-ACAR-AVENT-0419 | Amazon RDS tidak dapat mengakses kunci KMS enkripsi untuk instance *`name`* database. Database ini akan ditempatkan dalam keadaan tidak dapat diakses. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| backtrack | RDS-EVENT-0131 | Periode Backtrack aktual lebih kecil dari periode Backtrack target yang Anda tentukan. Pertimbangkan untuk mengurangi jumlah jam dalam periode Backtrack target Anda. | Untuk mengetahui informasi selengkapnya tentang backtracking, lihat Melakukan backtracking klaster DB Aurora. |
| backtrack | RDS-EVENT-0132 | Periode Backtrack aktual sama dengan periode Backtrack target. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0011 | Diperbarui untuk menggunakan DBParameter Grup*name*. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0012 | Menerapkan modifikasi pada kelas instans basis data. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0014 | Menyelesaikan penerapan modifikasi pada kelas instans DB. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0017 | Menyelesaikan penerapan modifikasi pada penyimpanan yang dialokasikan. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0025 | Menyelesaikan penerapan modifikasi untuk dikonversi menjadi instans DB Multi-AZ. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| perubahan konfigurasi | RDS-EVENT-0029 | Menyelesaikan penerapan modifikasi untuk dikonversi menjadi instans DB standar (AZ Tunggal). | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0033 | Ada *number* pengguna yang cocok dengan nama pengguna utama; hanya mengatur ulang yang tidak terikat ke host tertentu. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0067 | Tidak dapat mengatur ulang kata sandi Anda. Informasi kesalahan :*message*. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0078 | Interval Monitoring diubah menjadi*number*. | Konfigurasi Pemantaua n yang Ditingkatkan telah diubah. |
| perubahan konfigurasi | RDS-EVENT-0092 | Menyelesaikan pembaruan grup parameter DB. | Tidak ada |
| pembuatan | RDS-EVENT-0005 | Instans DB dibuat. | Tidak ada |
| penghapusan | RDS-EVENT-0003 | Instans DB dihapus. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| kegagalan | RDS-EVENT-0035 | Database instance dimasukkan ke dalam*state*. *message*. | Instans DB memiliki parameter yang tidak valid. Misalnya, jika instans DB tidak dapat dimulai karena parameter terkait memori diatur terlalu tinggi untuk kelas instans ini, tindakan Anda adalah memodifik asi parameter memori dan mem-boot ulang instans DB. |
| kegagalan | RDS-EVENT-0036 | Contoh database di*state*. *message*. | Instans DB ada di jaringan yang tidak kompatibel. Beberapa subnet yang ditentukan IDs tidak valid atau tidak ada. |
| kegagalan | RDS-EVENT-0079 | Amazon RDS tidak dapat membuat kredensial untuk pemantauan yang ditingkatkan dan fitur ini telah dinonaktifkan. Ini kemungkinan karena rds-monitoring-role tidak ada dan dikonfigurasi dengan benar di akun Anda. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumenta si Amazon RDS. | Pemantauan yang Ditingkat kan tidak dapat diaktifka n tanpa peran IAM Pemantauan yang Ditingkat kan. Untuk mengetahu i informasi tentang cara membuat peran IAM, lihat Untuk membuat peran IAM untuk pemantauan yang ditingkatkan Amazon RDS. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| kegagalan | RDS-EVENT-0080 | Amazon RDS tidak dapat mengonfigurasi pemantauan yang disempurnakan pada instans Anda: *name* dan fitur ini telah dinonaktifkan. Ini kemungkinan karena rds-monitoring-role tidak ada dan dikonfigurasi dengan benar di akun Anda. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS. | Pemantauan yang Ditingkatkan dinonaktifkan karena kesalahan terjadi saat perubahan konfigurasi. Sepertinya peran IAM Pemantauan yang Ditingkatkan tidak dikonfigurasi dengan benar. Untuk mengetahui informasi tentang cara membuat peran IAM pemantauan yang ditingkatkan, lihat Untuk membuat peran IAM untuk pemantauan yang ditingkatkan Amazon RDS. |
| kegagalan | RDS-EVENT-0082 | Amazon RDS tidak dapat membuat kredensil untuk mengakses Bucket Amazon S3 Anda untuk instans DB Anda. *name* Hal ini disebabkan peran IAM konsumsi snapshot S3 tidak dikonfigurasi dengan benar di akun Anda atau bucket Amazon S3 yang ditentukan tidak dapat ditemukan. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS. | Aurora tidak dapat menyalin data cadangan dari bucket Amazon S3. Sepertinya izin Aurora untuk mengakses bucket Amazon S3 tidak dikonfigurasi dengan benar. Untuk mengetahui informasi selengkapnya, lihat Migrasi fisik dari MySQL dengan menggunakan XtraBackup Percona dan Amazon S3 . |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| kegagalan | RDS-EVENT-0353 | Instans DB tidak dapat dibuat karena batas sumber daya yang tidak mencukupi. *message*. | *message*Termasuk rincian tentang kegagalan. |
| kegagalan | RDS-ACAR-AVENT-0418 | Amazon RDS tidak dapat mengakses kunci KMS enkripsi untuk instance *name* database. Ini kemungkinan karena kunci dinonaktifkan atau Amazon RDS tidak dapat mengaksesnya. Jika ini berlanjut, database akan ditempatkan ke dalam keadaan tidak dapat diakses. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS. | Tidak ada |
| kegagalan | RDS-ACAR-AVENT-0420 | Amazon RDS sekarang dapat berhasil mengakses kunci KMS enkripsi untuk instance *name* database. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| penyimpanan rendah | RDS-EVENT-0007 | Penyimpanan yang dialokasikan telah habis. Alokasikan penyimpanan tambahan untuk mengatasi masalah. | Penyimpanan yang dialokasikan untuk instans DB telah dipakai. Untuk mengatasi masalah ini, alokasikan penyimpanan tambahan untuk instans DB. Untuk mengetahui informasi selengkapnya, lihat Pertanyaan Umum tentang RDS. Anda dapat memantau ruang penyimpanan untuk instans DB menggunakan metrik Ruang Penyimpanan Kosong. |
| penyimpanan rendah | RDS-EVENT-0089 | Kapasitas penyimpanan gratis untuk instans DB: rendah pada penyimpanan *percentage* yang *name* disediakan [Provisioned Storage:*size*, Free Storage:]. *size* Anda mungkin ingin menambah penyimpanan yang disediakan untuk mengatasi masalah ini. | Instans DB telah menggunakan lebih dari 90% dari penyimpanan yang dialokasikan. Anda dapat memantau ruang penyimpanan untuk instans DB menggunakan metrik Ruang Penyimpanan Kosong. |
| penyimpanan rendah | RDS-EVENT-0227 | Penyimpanan cluster Aurora Anda sangat rendah dengan hanya terabyte yang tersisa. *amount* Ambil tindakan untuk mengurangi muatan penyimpanan pada klaster Anda. | Ruang subsistem penyimpanan Aurora hampir habis. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| pemeliharaan | RDS-EVENT-0026 | Menerapkan patch off-line ke instans DB. | Pemeliharaan offline instans DB sedang berlangsung. Instans DB saat ini tidak tersedia. |
| pemeliharaan | RDS-EVENT-0027 | Menyelesaikan penerapan patch off-line ke instans DB. | Pemeliharaan offline instans DB selesai. Instans DB kini tersedia. |
| pemeliharaan | RDS-EVENT-0047 | Instans basis data di-patch. | Tidak ada |
| pemeliharaan | RDS-EVENT-0155 | Instans DB memiliki peningkatan versi minor mesin DB yang tersedia. | Tidak ada |
| pemeliharaan | RDS-ACARA-0178 | Upgrade instance database sedang berlangsung. | Tidak ada |
| pemeliharaan | RDS-ACAR-AVENT-0422 | RDS akan menggantikan host instans DB *name* karena tindakan pemelihar aan yang tertunda. | Tidak ada |
| pemberita huan | RDS-EVENT-0044 | *message* | Ini adalah pemberita huan yang dikeluarkan operator. Untuk mengetahu i informasi selengkapnya, lihat pesan peristiwa. |
| pemberita huan | RDS-EVENT-0048 | Menunda peningkatan mesin basis data karena instans ini telah membaca replika yang perlu ditingkat kan terlebih dahulu. | Patching instans DB telah ditunda. |
| pemberita huan | RDS-EVENT-0087 | Instans DB dihentikan. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemberitahuan | RDS-EVENT-0088 | Instans DB dimulai. | Tidak ada |
| pemberitahuan | RDS-ACARA-0365 | File zona waktu diperbarui. Mulai ulang instans RDS Anda agar perubahan diterapkan. | Tidak ada |
| pemberitahuan, tanpa server | RDS-EVENT-0370 | Diawali jeda untuk instans DB. | Upaya baru untuk menjeda instans Aurora Serverless v2 DB idle telah dimulai. |
| pemberitahuan, tanpa server | RDS-ACARA-0371 | Jeda dibatalkan untuk instans DB. | Upaya untuk menjeda instans Aurora Serverless v2 DB yang menganggur tidak berhasil, karena beban kerja. |
| pemberitahuan, tanpa server | RDS-ACARA-0372 | Berhasil menjeda instans DB. | Instans Aurora Serverless v2 DB dijeda. |
| pemberitahuan, tanpa server | RDS-ACARA-0373 | Resume yang dimulai untuk instans DB. | Instans Aurora Serverless v2 DB mulai dilanjutkan, karena beban kerja baru atau aktivitas administratif atau pemeliharaan. |
| pemberitahuan, tanpa server | RDS-ACARA-0374 | Berhasil melanjutkan instance DB. | Instans Aurora Serverless v2 DB dilanjutkan. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemberita huan | RDS-ACARA-0385 | Topologi cluster diperbarui. | Ada perubahan DNS pada cluster DB untuk instance DB. Ini termasuk ketika instans DB baru ditambahk an atau dihapus, atau ada failover. |
| pemberita huan, basis data global | RDS-EVENT-0390 | Mencoba memblokir penulisan untuk cluster DB *cluster_id* di Wilayah *region_id* berhasil. | Aurora mulai memblokir penulisan di lapisan penyimpanan sebagai persiapan untuk peralihan atau kegagalan database global Aurora. |
| pemberita huan, basis data global | RDS-ACARA-0391 | Mencoba memblokir penulisan untuk cluster DB *cluster_id* di Wilayah yang habis *region_id* waktunya. | Aurora tidak dapat memblokir penulisan di lapisan penyimpanan sebagai persiapan untuk peralihan atau failover dari database global Aurora. Switchover atau failover akan dilanjutkan tetapi Anda mungkin perlu memulihkan data yang baru ditulis dari snapshot cluster primer asli. |
| replika baca | RDS-EVENT-0045 | Replikasi telah berhenti. | Pesan ini muncul ketika ada kesalahan selama replikasi. Untuk menentuka n jenis kesalahan, lihat Memecahkan masalah replika baca MySQL. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| replika baca | RDS-EVENT-0046 | Replikasi untuk Replika Baca dilanjutkan. | Pesan ini muncul saat Anda pertama kali membuat replika baca, atau sebagai pesan pemantaua n yang mengonfirmasi bahwa replikasi berfungsi dengan benar. Jika pesan ini mengikuti pemberita huan `RDS-EVENT-0045` , replikasi telah dilanjutkan setelah kesalahan atau setelah replikasi dihentikan. |
| replika baca | RDS-EVENT-0057 | Streaming replikasi telah dihentikan. | Tidak ada |
| pemulihan | RDS-EVENT-0020 | Pemulihan instans DB telah dimulai. Waktu pemulihan beragam dengan jumlah data yang akan dipulihkan. | Tidak ada |
| pemulihan | RDS-EVENT-0021 | Pemulihan instans DB selesai. | Tidak ada |
| pemulihan | RDS-EVENT-0023 | Permintaan Snapshot Muncul:. *`message`* | Pencadangan manual telah diminta, tetapi Amazon RDS saat ini dalam proses pembuatan snapshot DB. Kirim permintaan lagi setelah Amazon RDS menyelesaikan snapshot DB. |
| pemulihan | RDS-EVENT-0052 | Pemulihan instans Multi-AZ dimulai. | Waktu pemulihan beragam dengan jumlah data yang akan dipulihkan. |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemulihan | RDS-EVENT-0053 | Pemulihan instans Multi-AZ selesai. Failover atau aktivasi tertunda. | Pesan ini menunjukkan bahwa Amazon RDS telah menyiapkan instans DB Anda untuk memulai failover ke instance sekunder jika perlu. |
| pemulihan | RDS-ACARA-0361 | Pemulihan instans DB siaga telah dimulai. | Instans DB siaga dibangun kembali selama proses pemulihan. Kinerja database terpengaruh selama proses pemulihan. |
| pemulihan | RDS-ACARA-0362 | Pemulihan instans DB siaga telah selesai. | Instans DB siaga dibangun kembali selama proses pemulihan. Kinerja database terpengaruh selama proses pemulihan. |
| pemulihan | RDS-EVENT-0019 | Dipulihkan dari instans DB *name* ke*name*. | Instans DB telah dipulihkan dari point-in-time cadangan. |
| patching keamanan | RDS-EVENT-0230 | Pembaruan sistem tersedia untuk instans DB Anda. Untuk mengetahui informasi tentang cara menerapkan pembaruan, lihat 'Mempertahankan instans DB' di Panduan Pengguna RDS. | Patch sistem operasi versi minor baru tersedia untuk instans DB Anda. Untuk mengetahui informasi tentang cara menerapkan pembaruan, lihat Pembaruan sistem operasi untuk cluster Aurora DB. |

## Peristiwa grup parameter DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat grup parameter DB merupakan jenis sumber.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| perubahan konfigurasi | RDS-EVENT-0037 | Parameter *name* yang diperbarui *value* dengan metode terapkan*method*. | Tidak ada |

## Peristiwa grup keamanan DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat grup keamanan DB merupakan jenis sumber.

> **Note**
>
> Grup keamanan DB adalah sumber daya untuk EC2 -Classic. EC2-Classic pensiun pada 15 Agustus 2022. Jika Anda belum bermigrasi dari EC2 -Classic ke VPC, kami sarankan Anda bermigrasi sesegera mungkin. Untuk informasi selengkapnya, lihat Memigrasi dari EC2 - Klasik ke VPC di Panduan EC2 Pengguna Amazon dan blog EC2-Jaringan Klasik Pensiun - Inilah Cara Mempersiapkannya.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| perubahan konfigurasi | RDS-EVENT-0038 | Menerapkan perubahan ke grup keamanan. | Tidak ada |
| kegagalan | RDS-EVENT-0039 | Mencabut otorisasi sebagai. *user* | Kelompok keamanan yang dimiliki oleh *user* tidak ada. Otorisasi untuk grup keamanan telah dicabut karena tidak valid. |

## Acara grup shard DB

Tabel berikut menunjukkan kategori acara dan daftar peristiwa ketika grup shard DB adalah tipe sumber.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| kegagalan | RDS-EVENT-0324 | Pekerjaan pemuatan data gagal. | Pekerjaan pemuatan data gagal karena alasan yang muncul dalam pesan kesalahan. |
| pemeliharaan | RDS-ACARA-0302 | Tindakan Anda diperlukan untuk menyelesaikan pekerjaan pecahan terpisah yang tertunda *job_ID* untuk ID subcluster *number* di grup shard DB. *name*<br><br>Untuk menyelesaikan operasi memanggil SQL:<br><br>`SELECT rds_aurora.limitless_finalize_split_shard( job_ID);` | Tidak ada |
| pemeliharaan | RDS-ACARA-0303 | Finalisasi untuk pekerjaan pecahan terpisah *job_ID* telah dimulai untuk ID subcluster *number* di grup shard DB. *name* | Tidak ada |
| pemeliharaan | RDS-ACAR-AVENT-0304 | Pekerjaan pecahan terpisah *job_ID* telah berhasil diselesaikan untuk ID subcluster *number* | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| | | di grup shard DB. *name* Pecahan baru dengan ID subcluster *number* ditambahkan ke grup shard DB. *name* | |
| pemeliharaan | RDS-ACARA-0305 | Pekerjaan pecahan terpisah *job_ID* telah gagal untuk ID subcluster *number* di grup shard DB. *name* | Tidak ada |
| pemeliharaan | RDS-ACARA-0366 | Pekerjaan pecahan terpisah *job_ID* telah dimulai untuk ID subcluste r *number* di grup shard DB. *name* | Tidak ada |
| pemeliharaan | RDS-ACARA-0367 | Tambahkan pekerjaan router *job_ID* telah dimulai di grup *name* shard DB. | Tidak ada |
| pemeliharaan | RDS-ACAR-0368 | Tambahkan pekerjaan router *job_ID* telah berhasil diselesaikan. Router baru dengan ID subcluster *number* ditambahkan ke grup shard DB. *name* | Tidak ada |
| pemeliharaan | RDS-ACARA-0369 | Menambahkan pekerjaan router *job_ID* telah gagal di grup *name* shard DB. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| pemeliharaan | RDS-ACARA-0394 | Tambahkan pekerjaan router *job_ID* telah dibatalkan di grup pecahan DB. *name* | Tidak ada |
| pemeliharaan | RDS-EVENT-0395 | Pekerjaan pecahan terpisah *job_ID* telah dibatalkan di grup pecahan DB. *name* | Tidak ada |
| pemberitahuan | RDS-EVENT-0321 | Inisialisasi infrastruktur untuk pekerjaan Pemuatan Data. | Tidak ada |
| pemberitahuan | RDS-EVENT-0322 | Pekerjaan pemuatan data sedang berlangsung. | Tidak ada |
| pemberitahuan | RDS-EVENT-0323 | Pekerjaan pemuatan data berhasil diselesaikan. | Tidak ada |
| pemberitahuan | RDS-EVENT-0325 | Membatalkan pekerjaan pemuatan Data sesuai permintaan pelanggan. | Tidak ada |
| pemberitahuan | RDS-EVENT-0326 | Pekerjaan pemuatan data dibatalkan. | Tidak ada |

## Peristiwa Proksi RDS

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat Proksi RDS berupa jenis sumber.

| Kategori | ID peristiwa RDS | Pesan | Catatan |
|---|---|---|---|
| perubahan konfigurasi | RDS-EVENT-0204 | Proksi *name* DB yang dimodifikasi RDS. | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| perubahan konfigurasi | RDS-EVENT-0207 | RDS memodifikasi titik akhir proxy *name* DB. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0213 | RDS mendeteksi penambahan instans DB dan secara otomatis menambahkannya ke grup target proxy *name* DB. | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0214 | RDS mendeteksi penghapusan instans DB *name* dan secara otomatis menghapusnya dari kelompok target proxy DB*name*. *name* | Tidak ada |
| perubahan konfigurasi | RDS-EVENT-0215 | RDS mendeteksi penghapusan cluster DB *name* dan secara otomatis menghapusnya dari kelompok target proxy DB*name*. *name* | Tidak ada |
| pembuatan | RDS-EVENT-0203 | RDS membuat proxy *name* DB. | Tidak ada |
| pembuatan | RDS-EVENT-0206 | RDS membuat titik akhir *name* untuk proxy DB. *name* | Tidak ada |
| penghapusan | RDS-EVENT-0205 | RDS menghapus proxy *name* DB. | Tidak ada |
| penghapusan | RDS-EVENT-0208 | RDS menghapus titik akhir *name* untuk proxy DB. *name* | Tidak ada |

| Kategori | ID peristiwa RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| kegagalan | RDS-EVENT-0243 | RDS gagal menyediak an kapasitas untuk proxy *name* karena tidak ada cukup alamat IP yang tersedia di subnet Anda:. *name* Untuk memperbai ki masalah ini, pastikan subnet Anda memiliki jumlah minimum alamat IP yang tidak digunakan seperti yang direkomen dasikan dalam dokumenta si Proksi RDS. | Untuk menentukan jumlah yang direkomendasikan untuk kelas instans Anda, lihat Perencanaan untuk kapasitas alamat IP. |
| kegagalan | RDS-EVENT-0275 | RDS membatasi beberapa koneksi ke proxy DB. *name* Jumlah permintaan koneksi simultan dari klien ke proxy telah melampaui batas. | Tidak ada |

## Peristiwa deployment blue/green

Tabel berikut menunjukkan kategori acara dan daftar peristiwa ketika blue/green penyebaran adalah jenis sumber.

Untuk informasi selengkapnya tentang blue/green penerapan, lihat. Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database

| Kategori | ID peristiwa Amazon RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pembuatan | RDS-EVENT-0244 | Tugas deployment blue/ green selesai. Anda dapat membuat lebih banyak modifikasi pada basis | Tidak ada |

| Kategori | ID peristiwa Amazon RDS | Pesan | Catatan |
|---|---|---|---|
| | | data lingkungan hijau atau beralih antar deployment. | |
| kegagalan | RDS-EVENT-0245 | Pembuatan blue/gree n penerapan gagal karena*reason*. | Tidak ada |
| penghapus an | RDS-EVENT-0246 | Deployment blue/green dihapus. | Tidak ada |
| pemberita huan | RDS-EVENT-0247 | Peralihan dari awal*blue*. *green* | Tidak ada |
| pemberita huan | RDS-EVENT-0248 | Switchover selesai pada blue/green penerapan. | Tidak ada |
| kegagalan | RDS-EVENT-0249 | Switchover dibatalkan saat penerapan. blue/green | Tidak ada |
| pemberita huan | RDS-EVENT-0259 | Peralihan dari cluster DB memulai. *blue green* | Tidak ada |
| pemberita huan | RDS-EVENT-0260 | Peralihan dari klaster DB selesai. *blue green* Berganti nama *blue* menjadi *blue-old* dan *green* ke*blue*. | Tidak ada |
| kegagalan | RDS-EVENT-0261 | Peralihan dari cluster DB *blue* ke *green* dibatalkan karena. *reason* | Tidak ada |

| Kategori | ID peristiwa Amazon RDS | Pesan | Catatan |
| --- | --- | --- | --- |
| pemberitahuan | RDS-EVENT-0311 | Sinkronisasi urutan untuk peralihan cluster DB *green* telah *blue* dimulai. Switchover saat menggunakan urutan dapat menyebabkan waktu henti yang diperpanjang. | Tidak ada |
| pemberitahuan | RDS-EVENT-0312 | Sinkronisasi urutan untuk peralihan cluster *blue* DB *green* telah selesai. | Tidak ada |
| kegagalan | RDS-EVENT-0314 | Sinkronisasi urutan untuk peralihan cluster DB *blue* ke *green* dibatalkan karena urutan gagal disinkronkan. | Tidak ada |
| pemberitahuan | RDS-ACARA-0409 | *message* | Tidak ada |

# Memantau log Amazon Aurora Amazon

Setiap mesin RDS database menghasilkan log yang dapat Anda akses untuk audit dan pemecahan masalah. Jenis log ini bergantung pada mesin basis data Anda.

Anda dapat mengakses log database untuk instans DB menggunakan AWS Management Console, the AWS Command Line Interface (AWS CLI), atau Amazon RDSAPI. Anda tidak dapat menampilkan, melihat, atau mengunduh log transaksi.

> **Note**
>
> Dalam beberapa kasus, log berisi data tersembunyi. Oleh karena itu, AWS Management Console mungkin menampilkan konten dalam file log, tetapi file log mungkin kosong saat Anda mengunduhnya.

Topik


- Melihat dan mencantumkan file log basis data
- Mengunduh file log basis data
- Melihat file log basis data
- Menerbitkan log database ke Amazon CloudWatch Logs
- Membaca isi file log menggunakan REST
- File log basis data Aurora MySQL
- File log basis data Aurora PostgreSQL


## Melihat dan mencantumkan file log basis data

Anda dapat melihat file log database untuk mesin Aurora DB Anda dengan menggunakan file. AWS Management Console Anda dapat membuat daftar file log apa yang tersedia untuk diunduh atau dipantau dengan menggunakan AWS CLI atau Amazon RDSAPI.

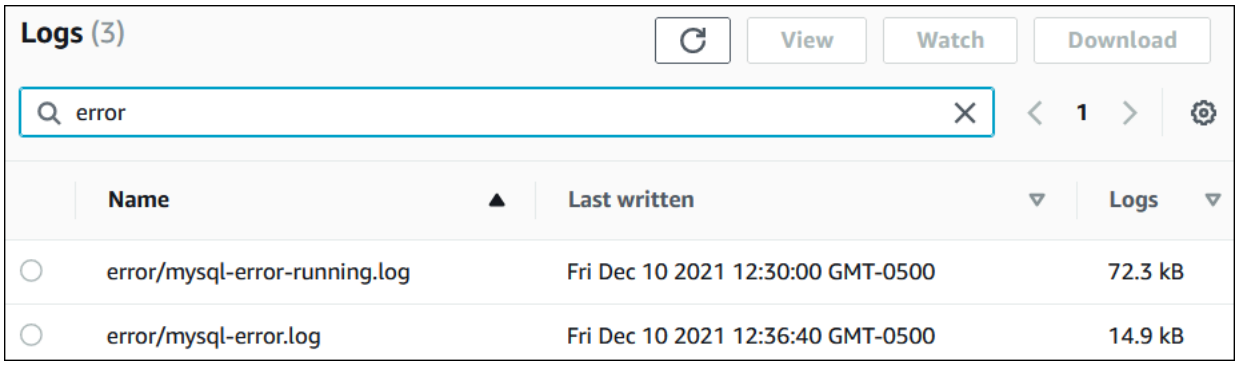
> **Note**
>
> Anda tidak dapat melihat file log untuk cluster Aurora Serverless v1 DB di RDS konsol. Namun, Anda dapat melihatnya di CloudWatch konsol Amazon di https://console.aws.amazon.com/cloudwatch/.

Konsol

Untuk melihat file log basis data

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih nama instans DB yang memiliki file log yang ingin dilihat.
4. Pilih tab Log & peristiwa.
5. Gulir ke bawah ke bagian Log.
6. (Opsional) Masukkan istilah pencarian untuk memfilter hasil.

   Contoh berikut mencantumkan log yang difilter dengan teks **`error`**.




7. Pilih log yang ingin dilihat, lalu pilih Lihat.

AWS CLI

Untuk membuat daftar file log database yang tersedia untuk instance DB, gunakan AWS CLI `describe-db-log-files`perintah.

Contoh berikut menampilkan daftar file log untuk instans DB yang bernama `my-db-instance`.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

Untuk mencantumkan file log database yang tersedia untuk instans DB, gunakan RDS API `DescribeDBLogFiles`tindakan Amazon.

# Mengunduh file log basis data

Anda dapat menggunakan AWS Management Console, AWS CLI, atau API untuk mengunduh file log database.

Konsol

Untuk mengunduh file log basis data

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih nama instans DB yang memiliki file log yang ingin dilihat.
4. Pilih tab Log & peristiwa.
5. Gulir ke bawah ke bagian Log.
6. Di bagian Log pilih tombol di samping log yang ingin diunduh, lalu pilih Unduh.
7. Buka menu konteks (klik kanan) untuk tautan yang diberikan, lalu pilih Simpan Tautan Sebagai. Masukkan lokasi tempat file log ingin disimpan, lalu pilih Simpan.

AWS CLI

Untuk mengunduh file log database, gunakan AWS CLI perintah `download-db-log-file-portion`. Secara default, perintah ini hanya mengunduh bagian terbaru dari file log. Namun, Anda dapat mengunduh seluruh file dengan menentukan parameter `--starting-token 0`.

Contoh berikut menunjukkan bagaimana untuk men-download seluruh isi dari file log yang disebut log/ ERROR .4 dan menyimpannya dalam file lokal bernama errorlog.txt.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

Untuk Windows:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

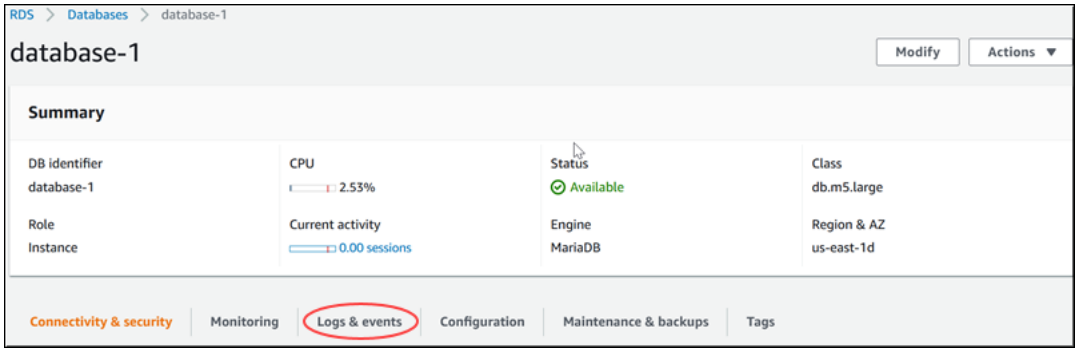
Untuk mengunduh file log database, gunakan RDS API DownloadDBLogFilePortiontindakan Amazon.

## Melihat file log basis data

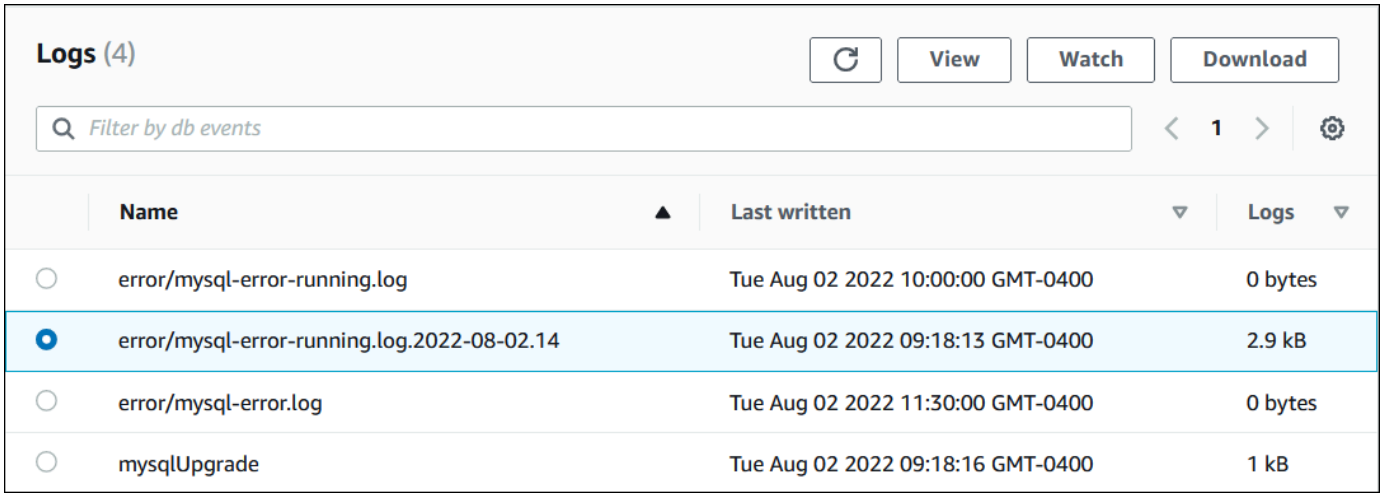
Menonton file log database setara dengan membuntuti file pada sistem UNIX atau Linux. Anda dapat melihat file log dengan menggunakan AWS Management Console. RDSmenyegarkan ekor log setiap 5 detik.

Untuk melihat file log basis data

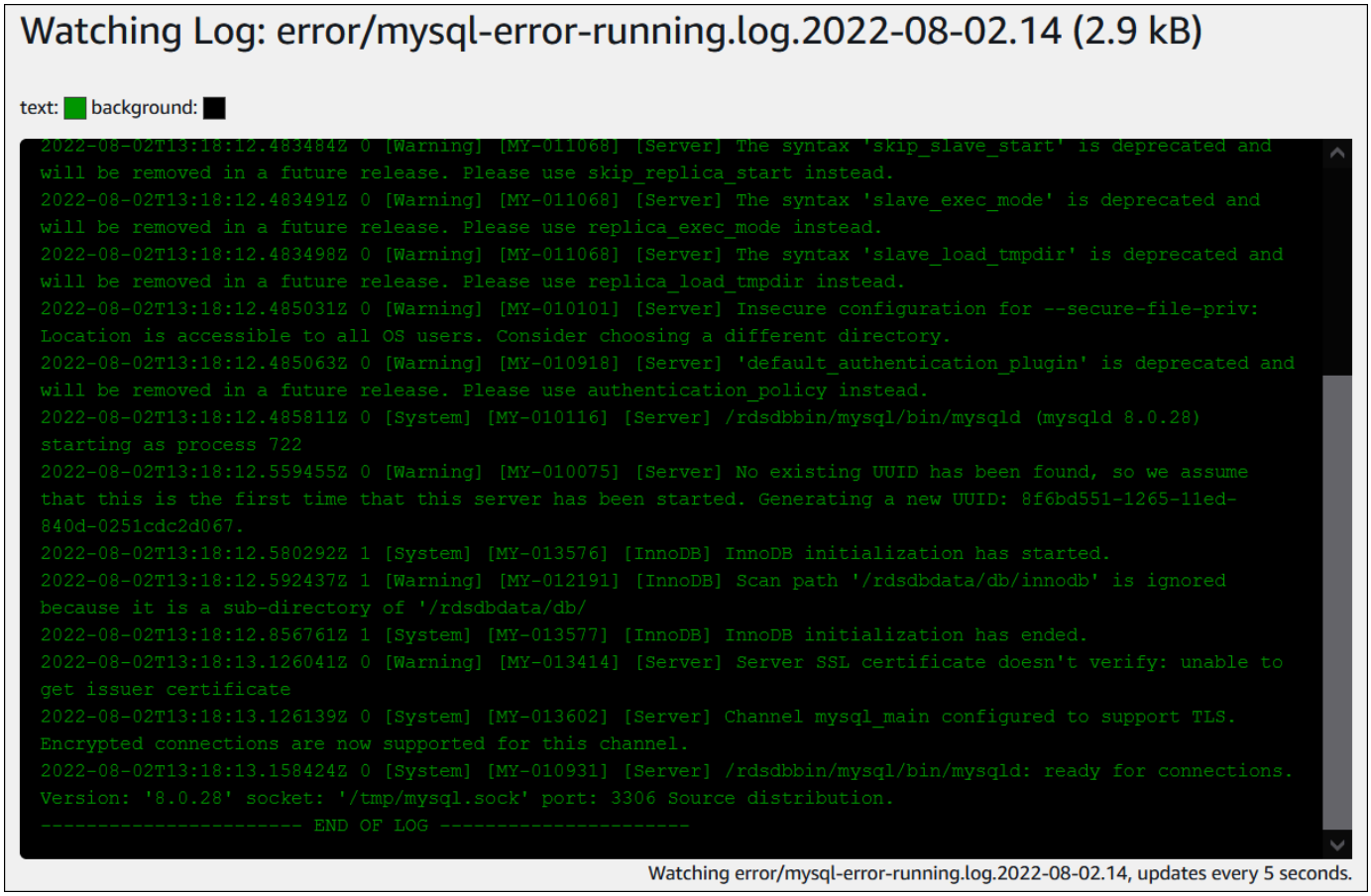
1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih nama instans DB yang memiliki file log yang ingin dilihat.
4. Pilih tab Log & peristiwa.




5. Di bagian Log, pilih file log, lalu pilih Lihat.




RDSmenunjukkan ekor log, seperti pada SQL contoh saya berikut.




## Menerbitkan log database ke Amazon CloudWatch Logs

Dalam basis data on-premise, log basis data berada pada sistem file. Amazon RDS tidak menyediakan akses host ke log basis data pada sistem file klaster DB Anda. Untuk alasan ini, Amazon RDS memungkinkan Anda mengekspor log database ke Amazon CloudWatch Logs. Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log. Anda juga dapat menyimpan data dalam penyimpanan yang sangat tahan lama dan mengelola data dengan Agen CloudWatch Log.

Topik

- Ikhtisar integrasi RDS dengan CloudWatch Log
- Memutuskan log mana yang akan dipublikasikan ke CloudWatch Log
- Menentukan log untuk dipublikasikan ke CloudWatch Log
- Mencari dan memfilter log Anda di CloudWatch Log

## Ikhtisar integrasi RDS dengan CloudWatch Log

Di CloudWatch Log, aliran log adalah urutan peristiwa log yang berbagi sumber yang sama. Setiap sumber log terpisah di CloudWatch Log membentuk aliran log terpisah. Grup log adalah grup log stream yang berbagi pengaturan retensi, pemantauan, dan kontrol akses yang sama.

Amazon Aurora terus mengalirkan data log klaster ke grup log. Misalnya, Anda memiliki grup log `/aws/rds/cluster/`*`cluster_name`*`/`*`log_type`* untuk setiap jenis log yang Anda terbitkan. Grup log ini berada di AWS Wilayah yang sama dengan instance database yang menghasilkan log.

AWS menyimpan data log yang dipublikasikan ke CloudWatch Log untuk jangka waktu yang tidak terbatas kecuali Anda menentukan periode retensi. Untuk informasi selengkapnya, lihat Ubah penyimpanan data log dalam CloudWatch Logs.

## Memutuskan log mana yang akan dipublikasikan ke CloudWatch Log

Setiap mesin basis data RDS mendukung kumpulan log-nya sendiri. Untuk mempelajari opsi untuk mesin basis data Anda, baca topik berikut:

- the section called “Menerbitkan log Aurora MySQL ke Log CloudWatch ”
- the section called “Menerbitkan log PostgreSQL Aurora ke Log CloudWatch ”

## Menentukan log untuk dipublikasikan ke CloudWatch Log

Tentukan log yang akan diterbitkan di konsol. Pastikan Anda memiliki peran terkait layanan di AWS Identity and Access Management (IAM). Untuk mengetahui informasi selengkapnya tentang peran terkait layanan, lihat Menggunakan peran terkait layanan untuk Amazon Aurora.

Untuk menentukan log yang akan diterbitkan

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Lakukan salah satu dari langkah berikut:
   - Pilih Buat basis data.
   - Pilih basis data dari daftar, lalu pilih Ubah.
4. Di Ekspor log, pilih log yang akan diterbitkan.

### Mencari dan memfilter log Anda di CloudWatch Log

Anda dapat mencari entri log yang memenuhi kriteria tertentu menggunakan konsol CloudWatch Log. Anda dapat mengakses log baik melalui konsol RDS, yang mengarahkan Anda ke konsol CloudWatch Log, atau dari konsol CloudWatch Log secara langsung.

Untuk mencari log RDS menggunakan konsol RDS

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB atau instans DB.
4. Pilih Konfigurasi.
5. Di bagian bawah Log yang diterbitkan, pilih log basis data yang ingin dilihat.

Untuk mencari log RDS Anda menggunakan konsol CloudWatch Log

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pada panel navigasi, pilih Grup log.
3. Di kotak filter, masukkan **`/aws/rds`**.
4. Untuk Grup Log, pilih nama grup log yang berisi log stream yang akan dicari.
5. Untuk Log Stream, pilih nama log stream yang akan dicari.
6. Di bagian Peristiwa log, masukkan sintaksis filter yang akan digunakan.

Untuk informasi selengkapnya, lihat Mencari dan memfilter data log di Panduan Pengguna CloudWatch Log Amazon. Untuk tutorial blog yang menjelaskan cara memantau log RDS, lihat Membangun pemantauan basis data proaktif untuk Amazon RDS dengan Amazon CloudWatch Logs, AWS Lambda, dan Amazon SNS.

## Membaca isi file log menggunakan REST

Amazon RDS menyediakan REST endpoint yang memungkinkan akses ke file log instans DB. Ini berguna jika Anda perlu menulis aplikasi untuk melakukan streaming konten file RDS log Amazon.

Sintaksnya adalah:

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
```

```
Content-type: application/json
host: rds.region.amazonaws.com
```

Parameter-parameter berikut diperlukan:

- *DBInstanceIdentifier*—nama instans basis data yang berisi file log yang ingin Anda unduh.
- *LogFileName*—nama file log yang akan diunduh.

Respons akan mengandung konten file log yang diminta, berupa sebuah aliran.

Contoh berikut mengunduh file log bernama log/ ERROR .6 untuk instance DB bernama sample-sql di wilayah us-west-2.

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

Jika Anda menentukan suatu instans basis data yang tidak ada, respons akan terdiri atas kesalahan berikut:

- `DBInstanceNotFound`—*DBInstanceIdentifier* tidak mengacu ke instans basis data yang ada. (kode HTTP status: 404)

# File log basis data Aurora MySQL

Anda dapat memantau log Aurora MySQL langsung melalui konsol Amazon RDS, Amazon RDS API, atau. AWS CLI AWS SDKs Anda juga dapat mengakses log MySQL dengan mengarahkan log ke tabel basis data di basis data utama dan mengueri tabel tersebut. Anda dapat menggunakan utilitas mysqlbinlog untuk mengunduh log biner.

Untuk mengetahui informasi selengkapnya tentang cara melihat, mengunduh, dan melihat log basis data berbasis file, lihat Memantau log Amazon Aurora Amazon.

Topik

- Ringkasan log basis data Aurora MySQL
- Mengirim Aurora Keluaran SQL log saya ke tabel
- Mengkonfigurasi Aurora untuk database Single-AZ
- Mengakses log biner MySQL

## Ringkasan log basis data Aurora MySQL

Anda dapat memantau jenis file log Aurora MySQL berikut:

- Log kesalahan
- Log kueri lambat
- Log umum
- Log audit
- Log contoh
- Log kesalahan otentikasi basis data IAM

Log kesalahan Aurora MySQL dihasilkan secara default. Anda dapat membuat kueri lambat dan log umum dengan mengatur parameter di grup parameter DB Anda.

Topik

- Log kesalahan Aurora MySQL
- Log umum dan kueri lambat Aurora MySQL
- Log audit Aurora MySQL
- Log contoh MySQL Aurora

- Rotasi dan retensi log untuk Aurora MySQL
- Menerbitkan log Aurora MySQL ke Amazon Logs CloudWatch

### Log kesalahan Aurora MySQL

Kesalahan tulis Aurora MySQL dalam file `mysql-error.log`. Setiap file log memiliki jam pembuatan (dalam UTC) yang ditambahkan pada namanya. File log juga memiliki stempel waktu yang membantu Anda menentukan kapan entri log ditulis.

Aurora MySQL ditulis ke log kesalahan hanya saat dinyalakan, dimatikan, dan saat terjadi kesalahan. Instan DB dapat memakan waktu berjam-jam atau berhari-hari tanpa perlu menulis entri baru ke log kesalahan. Jika Anda melihat tidak ada entri terbaru, berarti server tidak mengalami kesalahan yang akan mengakibatkan entri log.

Secara desain, log kesalahan difilter sehingga hanya peristiwa tak terduga seperti kesalahan yang ditampilkan. Namun, log kesalahan juga berisi beberapa informasi basis data tambahan, misalnya kemajuan kueri, yang tidak ditampilkan. Oleh karena itu, bahkan tanpa kesalahan aktual, ukuran log kesalahan mungkin meningkat dikarenakan aktivitas basis data yang sedang berlangsung. Dan sementara Anda mungkin melihat ukuran tertentu dalam byte atau kilobyte untuk log kesalahan di AWS Management Console, mereka mungkin memiliki 0 byte saat Anda mengunduhnya.

Aurora MySQL menulis `mysql-error.log` ke disk setiap 5 menit. Ini menambahkan konten log ke `mysql-error-running.log`.

Aurora MySQL merotasi file `mysql-error-running.log` setiap jam.

> **Note**
>
> Periode retensi log berbeda antara Amazon RDS dan Aurora.

### Log umum dan kueri lambat Aurora MySQL

Anda dapat menulis log umum dan log kueri lambat Aurora MySQL ke file atau tabel basis data. Untuk melakukannya, atur parameter di grup parameter DB Anda. Untuk mengetahui informasi tentang cara membuat dan memodifikasi grup parameter DB, lihat . Anda harus mengatur parameter ini sebelum dapat melihat log kueri lambat atau log umum di konsol Amazon RDS atau dengan menggunakan Amazon RDS API, Amazon RDS CLI, atau. AWS SDKs

Anda dapat mengontrol pencatatan log Aurora MySQL dengan menggunakan parameter dalam daftar ini:

- `slow_query_log`: Untuk membuat log kueri lambat, atur ke 1. Default-nya adalah 0.
- `general_log`: Untuk membuat log umum, atur ke 1. Default-nya adalah 0.
- `long_query_time`: Untuk mencegah pencatatan log kueri yang berjalan cepat dalam log kueri lambat, tentukan nilai untuk runtime kueri terpendek yang akan dicatat, dalam detik. Nilai default-nya adalah 10 detik; nilai minimumnya adalah 0. Jika log_output = FILE, Anda dapat menentukan nilai titik mengambang yang masuk ke resolusi mikrodetik. Jika log_output = TABLE, Anda harus menentukan nilai integer dengan resolusi kedua. Hanya kueri yang runtime-nya melebihi nilai `long_query_time` yang akan dicatat. Misalnya, mengatur `long_query_time` ke 0,1 akan mencegah pencatatan log kueri apa pun yang berjalan kurang dari 100 milidetik.
- `log_queries_not_using_indexes`: Untuk mencatat semua kueri yang tidak menggunakan indeks pada log kueri lambat, atur ke 1. Kueri yang tidak menggunakan indeks dicatat meskipun runtime-nya kurang dari nilai parameter `long_query_time`. Default-nya adalah 0.
- `log_output` *`option`*: Anda dapat menentukan salah satu opsi berikut untuk parameter `log_output`.
  - TABLE – Menulis kueri umum ke tabel `mysql.general_log`, dan kueri lambat ke tabel `mysql.slow_log`.
  - FILE – Menulis log umum dan log kueri lambat ke sistem file.
  - TIDAK ADA – Menonaktifkan pencatatan log.

  Untuk Aurora MySQL versi 2 dan 3, defaultnya adalah. `log_output FILE`

Agar data kueri lambat muncul di CloudWatch Log Amazon, kondisi berikut harus dipenuhi:

- CloudWatch Log harus dikonfigurasi untuk menyertakan log kueri yang lambat.
- `slow_query_log`harus diaktifkan.
- `log_output` harus diatur ke `FILE`.
- Kueri harus memakan waktu lebih lama dari waktu yang dikonfigurasi`long_query_time`.

Untuk mengetahui informasi selengkapnya tentang log umum dan kueri lambat, buka topik berikut di dokumentasi MySQL:

- Log kueri lambat

- Log kueri umum

### Log audit Aurora MySQL

Pencatatan log audit untuk Aurora MySQL disebut Audit Lanjutan. Untuk mengaktifkan Audit Lanjutan, tetapkan parameter klaster DB tertentu. Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.

### Log contoh MySQL Aurora

Aurora membuat file log terpisah untuk instans DB yang mengaktifkan jeda otomatis. File instance.log ini mencatat alasan mengapa instance DB ini tidak dapat dijeda saat diharapkan. Untuk informasi selengkapnya tentang perilaku file log instance dan kemampuan jeda otomatis Aurora, lihat Memantau aktivitas jeda dan melanjutkan Aurora Tanpa Server v2.

### Rotasi dan retensi log untuk Aurora MySQL

Saat pencatatan log diaktifkan, Amazon Aurora merotasi atau menghapus file log secara berkala. Langkah ini merupakan tindakan pencegahan untuk mengurangi kemungkinan file log besar memblokir penggunaan basis data atau memengaruhi performa. Aurora MySQL menangani rotasi dan penghapusan sebagai berikut:

- Ukuran file log kesalahan Aurora MySQL dibatasi hingga tidak lebih dari 15 persen dari penyimpanan lokal untuk instans DB. Untuk mempertahankan ambang batas ini, log secara otomatis dirotasi setiap jam. Aurora MySQL menghapus log setelah 30 hari atau ketika 15% dari ruang disk tercapai. Jika ukuran file log gabungan melebihi ambang batas setelah file log lama dihapus, file log paling lama akan dihapus hingga ukuran file log tidak lagi melebihi ambang batas.
- Aurora MySQL menghapus log audit, umum, dan kueri lambat setelah 24 jam atau ketika 15% dari penyimpanan telah terpakai.
- Saat pencatatan log `FILE` diaktifkan, file log umum dan kueri lambat akan diperiksa setiap jam dan file log yang berusia lebih dari 24 jam akan dihapus. Dalam beberapa kasus, ukuran file log gabungan yang tersisa setelah penghapusan mungkin melebihi ambang batas 15 persen dari ruang lokal instans DB. Dalam kasus ini, file log yang paling lama akan dihapus hingga ukuran file log tidak lagi melebihi ambang batas.
- Saat pencatatan log `TABLE` diaktifkan, tabel log tidak dirotasi atau dihapus. Tabel log akan dipotong jika ukuran semua log yang digabungkan terlalu besar. Anda dapat berlangganan kategori `low storage` acara untuk diberi tahu ketika tabel log harus diputar atau dihapus secara manual

untuk mengosongkan ruang. Untuk informasi selengkapnya, lihat Bekerja dengan pemberitahuan RDS acara Amazon.

Anda dapat merotasi tabel `mysql.general_log` secara manual dengan memanggil prosedur `mysql.rds_rotate_general_log`. Anda dapat merotasi tabel `mysql.slow_log` dengan mengikuti prosedur `mysql.rds_rotate_slow_log`.

Saat Anda merotasi tabel log secara manual, tabel log saat ini disalin ke tabel log cadangan dan entri di tabel log saat ini dihapus. Jika sudah ada, tabel log cadangan akan dihapus sebelum tabel log saat ini disalin ke cadangan. Anda dapat meminta tabel log cadangan jika diperlukan. Tabel log cadangan untuk tabel `mysql.general_log` bernama `mysql.general_log_backup`. Tabel log cadangan untuk tabel `mysql.slow_log` bernama `mysql.slow_log_backup`.

- Log audit Aurora MySQL dirotasi saat ukuran file mencapai 100 MB, dan dihapus setelah 24 jam.
- Amazon RDS memutar file log kesalahan otentikasi database IAM yang lebih besar dari 10 MB. Amazon RDS menghapus file log kesalahan otentikasi database IAM yang lebih tua dari lima hari atau lebih besar dari 100 MB.

Untuk bekerja dengan log dari konsol Amazon RDS, Amazon RDS API, Amazon RDS CLI, AWS SDKs atau, atur `log_output` parameter ke FILE. Seperti log kesalahan Aurora MySQL, file log ini dirotasi setiap jam. File log yang dihasilkan selama 24 jam sebelumnya akan dipertahankan. Perhatikan bahwa periode retensi log berbeda antara Amazon RDS dan Aurora.

Menerbitkan log Aurora MySQL ke Amazon Logs CloudWatch

Anda dapat mengonfigurasi cluster DB MySQL Aurora Anda untuk mempublikasikan data log ke grup log di Amazon Logs. CloudWatch Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Anda dapat menggunakan CloudWatch Log untuk menyimpan catatan log Anda dalam penyimpanan yang sangat tahan lama. Lihat informasi yang lebih lengkap di Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .

## Mengirim Aurora Keluaran SQL log saya ke tabel

Anda dapat mengarahkan log umum dan log kueri lambat ke tabel di instans DB dengan membuat grup parameter DB dan menetapkan parameter server `log_output` ke `TABLE`. Kueri umum lalu dicatat ke tabel `mysql.general_log` dan kueri lambat dicatat ke tabel `mysql.slow_log`. Anda dapat mengueri tabel untuk mengakses informasi log. Mengaktifkan pencatatan log ini akan meningkatkan jumlah data yang akan ditulis ke basis data, yang dapat menurunkan performa.

Log umum dan log kueri lambat dinonaktifkan secara default. Untuk mengaktifkan pencatatan log ke tabel, Anda juga harus mengatur parameter server `general_log` dan `slow_query_log` ke `1`.

Tabel log terus bertambah hingga aktivitas pencatatan log masing-masing dinonaktifkan dengan mengatur ulang parameter yang sesuai ke `0`. Banyak data yang sering terakumulasi seiring berjalannya waktu, yang dapat menghabiskan banyak persentase ruang penyimpanan yang dialokasikan. Amazon Aurora tidak memungkinkan Anda memotong tabel log, tetapi Anda dapat memindahkan isinya. Merotasi tabel akan menyimpan kontennya ke tabel cadangan, lalu membuat tabel log kosong yang baru. Anda dapat merotasi tabel log secara manual dengan mengikuti prosedur perintah berikut, tempat permintaan perintah ditunjukkan oleh `PROMPT>`:

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

Untuk menghapus data lama sepenuhnya dan mengosongkan kembali ruang disk, panggil prosedur yang sesuai dua kali secara berurutan.

## Mengkonfigurasi Aurora untuk database Single-AZ

Log biner adalah sekumpulan file log yang berisi informasi tentang modifikasi data yang dibuat ke instans server Aurora MySQL. Log biner berisi informasi seperti berikut:

- Peristiwa yang menggambarkan perubahan basis data seperti pembuatan tabel atau modifikasi baris
- Informasi tentang durasi setiap pernyataan yang memperbarui data
- Peristiwa untuk pernyataan yang bisa saja memperbarui data, tetapi tidak

Log biner mencatat pernyataan yang dikirim selama replikasi. Log ini juga diperlukan untuk beberapa operasi pemulihan. Untuk informasi selengkapnya, lihat Log Biner dalam dokumentasi MySQL.

Log biner hanya dapat diakses dari instans DB primer, bukan dari replika.

MySQL di Amazon Aurora mendukung format pencatatan log biner berbasis baris, berbasis pernyataan, dan campuran. Kami merekomendasikan campuran kecuali Anda memerlukan format binlog tertentu. Untuk detail tentang berbagai format log biner Aurora MySQL, lihat Format Pencatatan Biner dalam dokumentasi MySQL.

Jika Anda berencana menggunakan replikasi, format pencatatan log biner diperlukan karena menentukan catatan perubahan data yang dicatat di sumber dan dikirim ke target replikasi. Untuk

informasi tentang kelebihan dan kekurangan format logging biner yang berbeda untuk replikasi, lihat Keuntungan dan Kerugian Replikasi Berbasis Pernyataan dan Berbasis Baris dalam dokumentasi MySQL.

> **Important**
>
> Dengan MySQL 8.0.34, MySQL menghentikan parameter. `binlog_format` Di versi MySQL selanjutnya, MySQL berencana untuk menghapus parameter dan hanya mendukung replikasi berbasis baris. Sebagai hasilnya, sebaiknya gunakan pencatatan berbasis baris untuk pengaturan replikasi MySQL baru. Untuk informasi selengkapnya, lihat binlog_format dalam dokumentasi MySQL.
> MySQL versi 8.0 dan 8.4 menerima parameter. `binlog_format` Saat menggunakan parameter ini, MySQL mengeluarkan peringatan penghentian. Dalam rilis utama future, MySQL akan menghapus parameter. `binlog_format`
> Replikasi berbasis pernyataan dapat menyebabkan inkonsistensi antara klaster DB dan replika baca. Untuk informasi selengkapnya, lihat Penentuan Pernyataan Aman dan Tidak Aman di Binary Logging di dokumentasi MySQL.
> Mengaktifkan pencatatan log biner akan meningkatkan jumlah operasi I/O disk tulis untuk klaster DB. Anda dapat memantau penggunaan IOPS dengan `VolumeWriteIOPs` CloudWatch metrik.

Untuk mengatur format pencatatan log biner MySQL

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Pilih grup parameter klaster DB, yang terkait dengan klaster DB, yang ingin dimodifikasi.

   Anda tidak dapat mengubah grup parameter default. Jika klaster DB menggunakan grup parameter default, buat grup parameter baru dan hubungkan dengan klaster DB.

   Untuk mengetahui informasi selengkapnya tentang grup parameter, lihat .
4. Dari Tindakan, pilih Edit.
5. Atur parameter `binlog_format` ke format pencatatan log biner pilihan Anda (`ROW`, `STATEMENT`, atau `MIXED`). Anda juga dapat menggunakan nilai `OFF` untuk menonaktifkan pencatatan log biner.

> **Note**
>
> Pengaturan `binlog_format` ke `OFF` dalam kelompok parameter cluster DB menonaktifkan variabel `log_bin` sesi. Ini menonaktifkan logging biner pada cluster DB MySQL Aurora, yang pada gilirannya `binlog_format` mengatur ulang variabel sesi ke nilai default dalam database. `ROW`

6. Pilih Simpan perubahan untuk menyimpan pembaruan ke grup parameter klaster DB.

Setelah melakukan langkah-langkah ini, Anda harus mem-boot ulang instans penulis di klaster DB untuk menerapkan perubahan. Dalam Aurora MySQL versi 2.09 dan yang lebih rendah, saat Anda mem-boot ulang instans penulis, semua instans pembaca di klaster DB juga akan di-boot ulang. Dalam Aurora MySQL versi 2.10 dan yang lebih baru, Anda harus mem-boot ulang semua instans pembaca secara manual. Untuk informasi selengkapnya, lihat Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora.

> **Important**
>
> Mengubah grup parameter klaster DB memengaruhi semua klaster DB yang menggunakan grup parameter tersebut. Jika Anda ingin menentukan format logging biner yang berbeda untuk cluster Aurora MySQL DB yang berbeda di AWS Wilayah, cluster DB harus menggunakan kelompok parameter cluster DB yang berbeda. Grup parameter ini mengidentifikasi format pencatatan log yang berbeda. Tetapkan grup parameter klaster DB yang sesuai ke masing-masing klaster DB. Untuk mengetahui informasi selengkapnya tentang parameter Aurora MySQL, lihat Parameter konfigurasi Aurora MySQL.

## Mengakses log biner MySQL

Anda dapat menggunakan utilitas mysqlbinlog untuk mengunduh atau mengalirkan log biner dari instans DB RDS for MySQL. Log biner diunduh ke komputer lokal, tempat Anda dapat melakukan tindakan seperti memutar ulang log menggunakan utilitas mysql. Untuk mengetahui informasi selengkapnya tentang cara menggunakan utilitas mysqlbinlog, lihat Using mysqlbinlog to back up binary log files dalam dokumentasi MySQL.

Untuk menjalankan utilitas mysqlbinlog terhadap instans Amazon RDS, gunakan opsi berikut:

- `--read-from-remote-server` – Wajib diisi.
- `--host` – Nama DNS dari titik akhir instans.
- `--port` – Port yang digunakan oleh instans.
- `--user` – Pengguna MySQL yang telah diberi izin `REPLICATION SLAVE`.
- `--password` – Kata sandi untuk pengguna MySQL, atau hapus nilai kata sandi agar utilitas meminta Anda memasukkan kata sandi.
- `--raw` – Mengunduh file dalam format biner.
- `--result-file` – File lokal untuk menerima output mentah.
- `--stop-never` – Mengalirkan file log biner.
- `--verbose` – Jika Anda menggunakan format binlog `ROW`, sertakan opsi ini untuk melihat peristiwa baris sebagai pernyataan pseudo-SQL. Untuk mengetahui informasi selengkapnya tentang opsi `--verbose`, lihat mysqlbinlog row event display dalam dokumentasi MySQL.
- Tentukan nama satu atau beberapa file log biner. Untuk mendapatkan daftar log yang tersedia, gunakan perintah SQL `SHOW BINARY LOGS`.

Untuk mengetahui informasi selengkapnya tentang opsi mysqlbinlog, lihat mysqlbinlog — Utility for processing binary log files dalam dokumentasi MySQL.

Contoh berikut menunjukkan cara menggunakan utilitas mysqlbinlog.

Untuk Linux, macOS, atau Unix:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

Untuk Windows:

```
mysqlbinlog ^
    --read-from-remote-server ^
```

```
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

Log biner harus tetap tersedia pada instance DB untuk utilitas mysqlbinlog untuk mengaksesnya. Untuk memastikan ketersediaannya, gunakan prosedur yang mysql.rds_set_configuration disimpan dan tentukan periode dengan waktu yang cukup bagi Anda untuk mengunduh log. Jika konfigurasi ini tidak disetel, Amazon RDS membersihkan log biner sesegera mungkin, yang menyebabkan celah di log biner yang diambil oleh utilitas mysqlbinlog.

Contoh berikut menetapkan periode retensi ke 1 hari.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

Untuk menampilkan pengaturan saat ini, gunakan prosedur tersimpan mysql.rds_show_configuration.

```
call mysql.rds_show_configuration;
```

# File log basis data Aurora PostgreSQL

Anda dapat memantau jenis file log Aurora PostgreSQL berikut:

- Log PostgreSQL
- Log contoh
- Log kesalahan otentikasi basis data IAM

> **Note**
>
> Untuk informasi selengkapnya tentang mengaktifkan autentikasi database IAM, lihat. Mengaktifkan dan menonaktifkan autentikasi basis data IAM

Aurora PostgreSQL mencatat aktivitas basis data ke file log PostgreSQL default. Untuk instans DB PostgreSQL on-premise, pesan ini disimpan secara titik waktu di `log/postgresql.log`. Untuk klaster DB Aurora PostgreSQL, file log tersedia di klaster Aurora. Log ini juga dapat diakses melalui AWS Management Console, di mana Anda dapat melihat atau mengunduhnya. Tingkat pencatatan log default menangkap kegagalan masuk, kesalahan server fatal, kebuntuan, dan kegagalan kueri.

Untuk mengetahui informasi selengkapnya tentang cara menampilkan, mengunduh, dan melihat log basis data berbasis file, lihat Memantau log Amazon Aurora Amazon. Untuk mempelajari selengkapnya tentang log PostgreSQL, lihat Menggunakan log Amazon RDS dan Aurora PostgreSQL: Bagian 1 dan Menggunakan log Amazon RDS dan Aurora PostgreSQL: Bagian 2.

Selain log PostgreSQL standar yang dibahas dalam topik ini, Aurora PostgreSQL juga mendukung ekstensi PostgreSQL Audit (`pgAudit`). Sebagian besar industri dan lembaga pemerintah yang teregulasi perlu mempertahankan log audit atau jejak audit perubahan yang dibuat pada data untuk memenuhi persyaratan hukum. Untuk mengetahui informasi tentang cara menginstal dan menggunakan pgAudit, lihat Menggunakan pgAudit untuk mencatat aktivitas database.

Aurora membuat file log terpisah untuk instans DB yang mengaktifkan jeda otomatis. File instance.log ini mencatat alasan mengapa instance DB ini tidak dapat dijeda saat diharapkan. Untuk informasi selengkapnya tentang perilaku file log instance dan kemampuan jeda otomatis Aurora, lihat Memantau aktivitas jeda dan melanjutkan Aurora Tanpa Server v2.

Topik

•

- Mengaktifkan pencatatan log kueri untuk klaster DB Aurora PostgreSQL

Anda dapat menyesuaikan perilaku pencatatan log untuk klaster DB Aurora PostgreSQL dengan mengubah berbagai parameter. Dalam tabel berikut, Anda dapat menemukan parameter yang memengaruhi durasi penyimpanan log, waktu untuk memutar log, dan apakah akan menampilkan log sebagai format CSV (nilai yang dipisahkan koma). Anda juga dapat menemukan output teks yang dikirim ke STDERR, di antara pengaturan lainnya. Untuk mengubah pengaturan parameter yang dapat diubah, gunakan grup parameter klaster DB kustom untuk klaster DB Aurora PostgreSQL. Untuk mengetahui informasi selengkapnya, lihat .

| Parameter | Default | Deskripsi |
|---|---|---|
| log_destination | stderr | Menetapkan format output untuk log. Default-nya adalah `stderr`, tetapi Anda juga dapat menentukan nilai dipisahkan koma (CSV) dengan menambahkan `csvlog` ke pengaturan. Untuk informasi selengkapnya, lihat Mengatur tujuan log (`stderr, csvlog`). |
| log_filename | postgresql.log.%Y-%m-%d-%H%M | Menentukan pola untuk nama file log. Selain default, parameter ini mendukung `postgresql.log.%Y-%m-%d` dan `postgresql.log.%Y-%m-%d-%H` untuk pola nama file. Untuk Aurora PostgreSQL versi 17.4 dan yang lebih baru, Anda tidak dapat mengubah parameter ini. |
| log_line_prefix | %t:%r:%u@%d:[%p]: | Mendefinisikan awalan untuk setiap baris log yang akan ditulis ke `stderr`, untuk mencatat waktu (%t), host jarak jauh (%r), pengguna (%u), basis data (%d), dan ID proses (%p). |
| log_rotation_age | 60 | Menit setelah file log dirotasi secara otomatis. Anda dapat mengubah nilai ini dalam kisaran 1 dan 1440 menit. Untuk informasi selengkapnya, lihat Mengatur rotasi file log. |

| Parameter | Default | Deskripsi |
| --- | --- | --- |
| log_rotation_size | – | Ukuran (kB) tempat log dirotasi secara otomatis. Anda dapat mengubah nilai ini dalam kisaran 50.000 hingga 1.000.000 kilobyte. Untuk mempelajari selengkapnya, lihat Mengatur rotasi file log. |
| rds.log_retention_period | 4320 | Log PostgreSQL yang lebih lama dari jumlah menit yang ditentukan dihapus. Nilai default 4320 menit menghapus file log setelah 3 hari. Untuk informasi selengkapnya, lihat Mengatur periode retensi log. |

Untuk mengidentifikasi masalah aplikasi, Anda dapat mencari kegagalan kueri, kegagalan masuk, kebuntuan, dan kesalahan server fatal di log. Misalnya, anggaplah Anda mengonversi aplikasi lama dari Oracle ke Aurora PostgreSQL, tetapi tidak semua kueri dikonversi dengan benar. Kueri yang salah format ini menghasilkan pesan kesalahan yang dapat Anda temukan di log untuk membantu mengidentifikasi masalah. Untuk mengetahui informasi selengkapnya tentang pencatatan log kueri, lihat Mengaktifkan pencatatan log kueri untuk klaster DB Aurora PostgreSQL.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara mengatur berbagai parameter yang mengontrol detail dasar untuk log PostgreSQL Anda.

Topik

- Mengatur periode retensi log
- Mengatur rotasi file log
- Mengatur tujuan log (stderr, csvlog)
- Memahami parameter log_line_prefix

### Mengatur periode retensi log

Parameter `rds.log_retention_period` menentukan berapa lama klaster DB Aurora PostgreSQL menyimpan file log-nya. Pengaturan default-nya adalah 3 hari (4.320 menit), tetapi Anda dapat mengatur nilai ini ke mana saja dari 1 hari (1.440 menit) hingga 7 hari (10.080 menit). Pastikan

bahwa klaster DB Aurora PostgreSQL Anda memiliki penyimpanan yang cukup untuk menyimpan file log selama periode waktu tertentu.

Kami menyarankan agar log Anda dipublikasikan secara rutin ke Amazon CloudWatch Logs sehingga Anda dapat melihat dan menganalisis data sistem lama setelah log dihapus dari cluster DB PostgreSQL Aurora Anda. Untuk mengetahui informasi selengkapnya, lihat Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch . Setelah Anda mengatur CloudWatch penerbitan, Aurora tidak menghapus log sampai setelah diterbitkan ke CloudWatch Log.

Amazon Aurora mengompresi log PostgreSQL lama ketika penyimpanan untuk instans DB mencapai ambang batas. Aurora mengompresi file menggunakan utilitas kompresi gzip. Untuk mengetahui informasi selengkapnya, lihat situs web gzip.

Saat penyimpanan instans DB rendah dan semua log yang tersedia dikompresi, Anda akan mendapatkan peringatan seperti berikut:

```
Warning: local storage for PostgreSQL log files is critically low for
this Aurora PostgreSQL instance, and could lead to a database outage.
```

Jika tidak ada cukup penyimpanan, Aurora dapat menghapus log PostgreSQL yang terkompresi sebelum akhir periode retensi tertentu. Jika hal ini terjadi, Anda akan melihat pesan yang mirip dengan pesan berikut:

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

Mengatur rotasi file log

Aurora membuat file log baru setiap jam secara default. Waktu dikontrol oleh parameter `log_rotation_age`. Parameter ini memiliki nilai default 60 (menit), tetapi Anda dapat mengaturnya ke mana saja dari 1 menit hingga 24 jam (1.440 menit). Ketika tiba waktunya rotasi, file log baru yang berbeda akan dibuat. File ini diberi nama sesuai dengan pola yang ditentukan oleh parameter `log_filename`.

File log juga dapat dirotasi sesuai dengan ukurannya, seperti yang ditentukan dalam parameter `log_rotation_size`. Parameter ini menentukan bahwa log harus dirotasi saat mencapai ukuran yang ditentukan (dalam kilobyte). Nilai default `log_rotation_size` adalah 100000 kB (kilobyte) untuk klaster DB Aurora PostgreSQL, tetapi Anda dapat mengatur nilai ini ke mana pun dari 50.000 hingga 1.000.000 kilobyte.

Nama file log didasarkan pada pola nama file yang ditentukan dalam parameter `log_filename`. Pengaturan yang tersedia untuk parameter ini adalah sebagai berikut:

- `postgresql.log.%Y-%m-%d` – Format default untuk nama file log. Termasuk tahun, bulan, dan tanggal dalam nama file log.
- `postgresql.log.%Y-%m-%d-%H` – Termasuk jam dalam format nama file log.
- `postgresql.log.%Y-%m-%d-%H%M` – Termasuk jam:menit dalam format nama file log.

Jika Anda mengatur parameter `log_rotation_age` ke kurang dari 60 menit, atur parameter `log_filename` ke format menit.

Untuk mengetahui informasi selengkapnya, lihat `log_rotation_age` dan `log_rotation_size` dalam dokumentasi PostgreSQL.

Mengatur tujuan log (**`stderr`**, **`csvlog`**)

Secara default, Aurora PostgreSQL menghasilkan log dalam format kesalahan standar (stderr). Format ini adalah pengaturan default untuk parameter `log_destination`. Setiap pesan diawali menggunakan pola yang ditentukan dalam parameter `log_line_prefix`. Untuk mengetahui informasi selengkapnya, lihat Memahami parameter log_line_prefix.

Aurora PostgreSQL juga dapat menghasilkan log dalam format `csvlog`. `csvlog` berguna untuk menganalisis data log sebagai data nilai yang dipisahkan koma (CSV). Misalnya, anggaplah Anda menggunakan ekstensi `log_fdw` untuk bekerja dengan log Anda sebagai tabel asing. Tabel asing yang dibuat pada file log `stderr` berisi satu kolom dengan data peristiwa log. Dengan menambahkan `csvlog` ke parameter `log_destination`, Anda mendapatkan file log dalam format CSV dengan demarkasi untuk beberapa kolom tabel asing. Anda kini dapat mengurutkan dan menganalisis log dengan lebih mudah.

Jika Anda menentukan `csvlog` untuk parameter ini, perhatikan bahwa file `stderr` dan `csvlog` dihasilkan. Pastikan untuk memantau penyimpanan yang digunakan oleh log, dengan mempertimbangkan `rds.log_retention_period` dan pengaturan lain yang memengaruhi penyimpanan log dan omset. Menggunakan `stderr` dan `csvlog` lebih dari dua kali lipat penyimpanan yang digunakan oleh log.

Jika Anda menambahkan `csvlog` ke `log_destination` dan ingin kembali ke `stderr` sendiri, Anda perlu mengatur ulang parameter. Untuk melakukannya, buka Konsol Amazon RDS, lalu buka grup parameter klaster DB untuk instans Anda. Pilih parameter `log_destination`, pilih Edit parameter, lalu pilih Atur ulang.

Untuk mengetahui informasi selengkapnya tentang cara mengonfigurasi pencatatan log, lihat Menggunakan log Amazon RDS dan Aurora PostgreSQL: Bagian 1.

Memahami parameter log_line_prefix

Format `stderr` log awalan setiap pesan log dengan rincian yang ditentukan oleh parameter. `log_line_prefix` Nilai defaultnya adalah:

```
%t:%r:%u@%d:[%p]:t
```

Mulai dari Aurora PostgreSQL versi 16, Anda juga dapat memilih:

```
%m:%r:%u@%d:[%p]:%l:%e:%s:%v:%x:%c:%q%a
```

Setiap entri log yang dikirim ke stderr mencakup informasi berikut berdasarkan nilai yang dipilih:

- `%t`— Waktu entri log tanpa milidetik
- `%m`— Waktu entri log dengan milidetik
- `%r` – Alamat host jarak jauh
- `%u@%d` – Nama pengguna @ nama basis data
- `[%p]` – ID Proses jika tersedia
- `%l`— Nomor baris log per sesi
- `%e`— Kode kesalahan SQL
- `%s`— Stempel waktu mulai proses
- `%v`— Id transaksi virtual
- `%x`— ID Transaksi
- `%c`— ID Sesi
- `%q`— Terminator non-sesi
- `%a`— Nama aplikasi

## Mengaktifkan pencatatan log kueri untuk klaster DB Aurora PostgreSQL

Anda dapat mengumpulkan informasi yang lebih mendetail tentang aktivitas basis data, termasuk kueri, kueri yang menunggu kunci, titik pemeriksaan, dan banyak detail lainnya dengan mengatur beberapa parameter yang tercantum dalam tabel berikut. Topik ini berfokus pada kueri pencatatan log.

| Parameter | Default | Deskripsi |
| --- | --- | --- |
| log_connections | – | Mencatat log setiap koneksi yang berhasil. Untuk mempelajari cara menggunak an parameter ini dengan `log_disco nnections` untuk mendeteksi churn koneksi, lihatMengelola churn koneksi Aurora PostgreSQL dengan pooling . |
| log_disconnections | – | Mencatat log akhir setiap sesi dan durasinya . Untuk mempelajari cara menggunak an parameter ini dengan `log_conne ctions` untuk mendeteksi churn koneksi, lihatMengelola churn koneksi Aurora PostgreSQL dengan pooling . |
| log_checkpoints | — | Tidak berlaku untuk Aurora PostgreSQL |
| log_lock_waits | – | Mencatat log waktu tunggu kunci yang panjang. Secara default, parameter ini tidak diatur. |
| log_min_duration_s ample | – | (md) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat sampel pernyataan dicatat. Ukuran sampel diatur menggunakan parameter `log_state ment_sample_rate` . |
| log_min_duration_s tatement | – | Setiap pernyataan SQL yang berjalan setidakny a selama periode waktu tertentu atau lebih lama akan dicatat. Secara default, parameter ini tidak diatur. Mengaktifkan parameter ini dapat membantu Anda menemukan kueri yang belum dioptimalkan. |
| log_statement | – | Mengatur jenis pernyataan yang dicatat. Secara default, parameter ini tidak diatur, tetapi Anda dapat mengubahnya ke `all`, `ddl`, atau `mod` untuk menentukan jenis pernyataan SQL |

| Parameter | Default | Deskripsi |
| --- | --- | --- |
| | | yang ingin Anda catat. Jika menentukan apa pun selain none untuk parameter ini, Anda juga harus mengambil langkah-langkah tambahan untuk mencegah eksposur kata sandi dalam file log. Untuk informasi selengkapnya, lihat Mengurangi risiko eksposur kata sandi saat menggunakan pencatatan log kueri. |
| log_statement_sample_rate | – | Persentase pernyataan melebihi waktu yang ditentukan dalam `log_min_duration_sample` untuk dicatat, yang dinyatakan sebagai nilai titik mengambang antara 0,0 dan 1,0. |
| log_statement_stats | – | Menulis statistik performa kumulatif ke log server. |

Menggunakan pencatatan log untuk menemukan kueri performa lambat

Anda dapat mencatat pernyataan dan kueri SQL untuk membantu menemukan kueri performa lambat. Anda mengaktifkan kemampuan ini dengan memodifikasi pengaturan di `log_statement` dan parameter `log_min_duration` seperti yang diuraikan dalam bagian ini. Sebelum mengaktifkan pencatatan log kueri untuk klaster DB Aurora PostgreSQL, Anda harus mengetahui kemungkinan eksposur kata sandi di dalam log dan cara mengurangi risiko ini. Untuk informasi selengkapnya, lihat Mengurangi risiko eksposur kata sandi saat menggunakan pencatatan log kueri.

Berikut ini, Anda dapat menemukan informasi referensi tentang parameter `log_statement` dan `log_min_duration`.

log_statement

Parameter ini menentukan jenis pernyataan SQL yang harus dikirim ke log. Nilai default-nya adalah `none`. Jika Anda mengubah parameter ini ke `all`, `ddl`, atau `mod`, pastikan untuk menerapkan tindakan yang disarankan untuk mengurangi risiko eksposur kata sandi di dalam log. Untuk informasi selengkapnya, lihat Mengurangi risiko eksposur kata sandi saat menggunakan pencatatan log kueri.

all

Mencatat log semua pernyataan. Pengaturan ini direkomendasikan untuk tujuan debugging.

ddl

Mencatat log semua pernyataan bahasa definisi data (DDL), seperti CREATE, ALTER, DROP, dan seterusnya.

mod

Mencatat log semua pernyataan DDL dan pernyataan bahasa manipulasi data (DL), seperti INSERT, UPDATE, dan DELETE, yang mengubah data.

none

Tidak ada pernyataan SQL yang dicatat. Kami merekomendasikan pengaturan ini untuk menghindari risiko eksposur kata sandi di dalam log.

log_min_duration_statement

Setiap pernyataan SQL yang berjalan setidaknya selama periode waktu tertentu atau lebih lama akan dicatat. Secara default, parameter ini tidak diatur. Mengaktifkan parameter ini dapat membantu Anda menemukan kueri yang belum dioptimalkan.

–1–2147483647

Jumlah milidetik (md) runtime di mana pernyataan dicatat.

Untuk menyiapkan pencatatan log kueri

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL Anda menggunakan grup parameter klaster kustom.

1. Atur parameter `log_statement` ke `all`. Contoh berikut ini menunjukkan informasi yang ditulis ke file `postgresql.log` dengan pengaturan parameter ini.

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
```

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
----------------------- END OF LOG ----------------------
```

2. Atur parameter `log_min_duration_statement`. Contoh berikut ini menunjukkan informasi yang ditulis ke file `postgresql.log` saat pengaturan parameter ini diatur ke 1.

   Kueri yang melebihi durasi yang ditentukan dalam parameter `log_min_duration_statement` dicatat. Bagian berikut menunjukkan satu contoh. Anda dapat melihat file log untuk klaster DB Aurora PostgreSQL di Konsol Amazon RDS.

   ```
   2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
    table comments;
   2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
    167.754 ms
   2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
   2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
    (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
    total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
    estimate=131028 kB
   ----------------------- END OF LOG ----------------------
   ```

### Mengurangi risiko eksposur kata sandi saat menggunakan pencatatan log kueri

Sebaiknya Anda tetap mengatur `log_statement` ke `none` agar tidak mengekspos kata sandi. Jika Anda mengatur `log_statement` ke `all`, `ddl`, atau `mod`, sebaiknya Anda mengambil satu atau beberapa langkah berikut.

- Untuk klien, enkripsi informasi sensitif. Untuk mengetahui informasi selengkapnya, lihat Opsi Enkripsi dalam dokumentasi PostgreSQL. Gunakan opsi `ENCRYPTED` (dan `UNENCRYPTED`) dari pernyataan `CREATE` dan `ALTER`. Untuk mengetahui informasi selengkapnya, lihat CREATE USER dalam dokumentasi PostgreSQL.
- Untuk klaster DB Aurora PostgreSQL, siapkan dan gunakan ekstensi PostgreSQL Audit (pgAaudit). Ekstensi ini menyunting informasi sensitif dalam pernyataan CREATE dan ALTER yang dikirim ke log. Untuk informasi selengkapnya, lihat Menggunakan pgAudit untuk mencatat aktivitas database.
- Batasi akses ke CloudWatch log.
- Gunakan mekanisme autentikasi yang lebih kuat seperti IAM.

# Memantau panggilan Amazon Aurora API AWS CloudTrail

AWS CloudTrail adalah AWS layanan yang membantu Anda mengaudit AWS akun Anda. AWS CloudTrail diaktifkan untuk AWS akun Anda saat Anda membuatnya. Untuk informasi selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

Topik


- CloudTrail
- Entri file RDS log Amazon Aurora


## CloudTrail

Semua RDS tindakan Amazon Aurora dicatat oleh. CloudTrail CloudTrail

### CloudTrail acara

CloudTrail menangkap API panggilan untuk Amazon Amazon sebagai acara. Peristiwa merepresentasikan satu permintaan dari sumber apa pun dan menyertakan informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. Acara termasuk panggilan dari RDS konsol Amazon dan dari panggilan kode ke RDS API operasi Amazon.

RDSAktivitas Amazon Aurora dicatat dalam suatu CloudTrail peristiwa dalam sejarah Acara. Anda dapat menggunakan CloudTrail konsol untuk melihat 90 hari terakhir API aktivitas dan peristiwa yang direkam di suatu AWS Wilayah. Untuk informasi selengkapnya, lihat Melihat peristiwa dengan riwayat CloudTrail acara.

### CloudTrail jalan setapak

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk Amazon Aurora , buat jejak. Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa ke bucket Amazon S3 tertentu. CloudTrail biasanya mengirimkan file log dalam waktu 15 menit dari aktivitas akun.

> **Note**
>
> Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara.

Anda dapat membuat dua jenis jalur untuk AWS akun: jejak yang berlaku untuk semua Wilayah, atau jejak yang berlaku untuk satu Wilayah. Secara default, ketika Anda membuat jejak di konsol, jejak diterapkan ke semua Region.

Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat:

- Ikhtisar untuk membuat jejak
- CloudTrail layanan dan integrasi yang didukung
- Mengonfigurasi SNS notifikasi Amazon untuk CloudTrail
- Menerima file CloudTrail log dari beberapa Wilayah dan Menerima file CloudTrail log dari beberapa akun

## Entri file RDS log Amazon Aurora

CloudTrail file log berisi satu atau lebih entri log. CloudTrail file log bukanlah jejak tumpukan yang dipesan dari API panggilan publik, sehingga tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `CreateDBInstance` tindakan.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
```

```
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
```

```
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
```

```
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Seperti ditunjukkan pada elemen `userIdentity` dalam contoh sebelumnya, setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan. Informasi identitas membantu Anda menentukan hal-hal berikut:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau IAM.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna terfederasi.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi lebih lanjut tentang`userIdentity`, lihat CloudTrail userIdentity elemen. Untuk informasi selengkapnya tentang `CreateDBInstance` dan RDS tindakan Amazon Aurora lainnya, lihat Referensi Amazon RDS API.

# Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data

Dengan menggunakan Aliran Aktivitas Basis Data, Anda dapat memantau aliran aktivitas basis data secara waktu nyaris nyata.

Topik


- Ikhtisar Aliran Aktivitas Basis Data
- Prasyarat jaringan untuk aliran aktivitas basis data Aurora MySQL
- Memulai aliran aktivitas basis data
- Mendapatkan status aliran aktivitas basis data
- Menghentikan aliran aktivitas basis data
- Memantau aliran aktivitas basis data
- Contoh kebijakan IAM untuk aliran aktivitas database


## Ikhtisar Aliran Aktivitas Basis Data

Sebagai seorang administrator basis data Amazon Aurora, Anda perlu melindungi basis data Anda dan memenuhi persyaratan kepatuhan dan peraturan. Satu strateginya adalah mengintegrasikan aliran aktivitas basis data dengan alat pemantauan.

Ancaman keamanan bersifat eksternal dan internal. Untuk melindungi terhadap ancaman internal, Anda dapat mengendalikan akses administrator ke aliran data dengan mengonfigurasikan fitur Aliran Aktivitas Basis Data. DBAs tidak memiliki akses ke pengumpulan, transmisi, penyimpanan, dan pemrosesan aliran.

Daftar Isi


- Cara kerja aliran aktivitas basis data
- Mode asinkron dan sinkron untuk aliran aktivitas basis data
- Persyaratan dan batasan untuk aliran aktivitas basis data
- Wilayah dan ketersediaan versi
- Kelas-kelas instans basis data yang didukung untuk aliran aktivitas basis data

## Cara kerja aliran aktivitas basis data

Di Amazon Aurora, Anda memulai aliran aktivitas basis data di tingkat klaster. Semua instans basis data dalam klaster Anda mengaktifkan aliran aktivitas basis data.

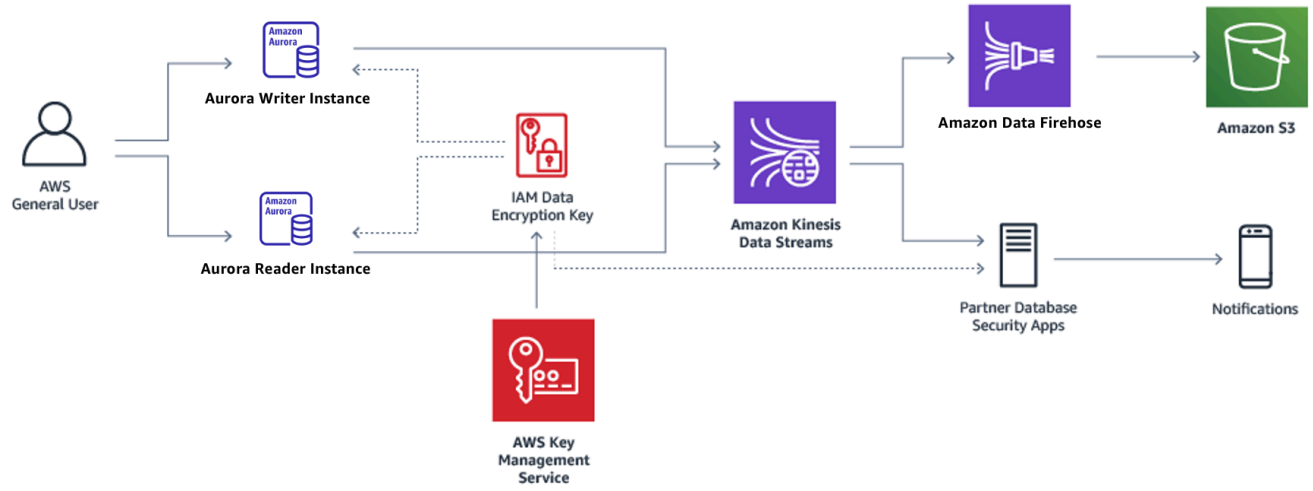
Klaster basis data Aurora Anda mendorong aktivitas ke aliran data Amazon Kinesis dalam waktu nyaris nyata. Aliran Kinesis dibuat secara otomatis. Dari Kinesis, Anda dapat mengonfigurasi AWS layanan seperti Amazon Data Firehose dan AWS Lambda menggunakan aliran dan menyimpan data.

> **Important**
>
> Penggunaan fitur aliran aktivitas basis data di Amazon RDS adalah gratis, tetapi Amazon Kinesis mengenakan biaya untuk aliran data. Lihat informasi yang lebih lengkap di struktur harga Amazon Kinesis Data Streams.

Jika Anda menggunakan basis data global Aurora, mulai secara terpisah aliran aktivitas basis data pada setiap klaster basis data. Setiap klaster mengirimkan data audit ke aliran Kinesisnya sendiri dalam Wilayah AWS-nya sendiri. Aliran aktivitas tidak beroperasi secara berbeda selama failover. Aliran akan terus mengaudit basis data global Anda seperti biasa.

Anda dapat mengonfigurasikan aplikasi untuk pengelolaan kepatuhan agar menggunakan aliran aktivitas basis data. Untuk Aurora PostgreSQL, aplikasi kepatuhan mencakup IBM Security Guardium dan Audit dan Perlindungan Database Imperva. SecureSphere  Aplikasi-aplikasi ini dapat menggunakan aliran untuk menghasilkan peringatan dan aktivitas audit pada klaster basis data Aurora.

Grafik berikut menunjukkan cluster Aurora DB yang dikonfigurasi dengan Amazon Data Firehose.




## Mode asinkron dan sinkron untuk aliran aktivitas basis data

Anda dapat memilih agar sesi basis data menangani peristiwa aktivitas basis data dalam salah satu mode berikut:

- Mode asinkron – Apabila sesi basis data menghasilkan peristiwa aliran aktivitas, sesi akan kembali dengan seketika ke aktivitas normal. Di latar belakang, peristiwa aliran aktivitas dijadikan catatan yang durabel. Jika kesalahan terjadi dalam tugas latar belakang, peristiwa RDS akan dikirim. Peristiwa ini menunjukkan awal dan akhir segala jendela waktu ketika catatan peristiwa aliran aktivitas mungkin telah hilang.

  Mode asinkron lebih memprioritaskan performa basis data daripada akurasi aliran aktivitas.

  > **Note**
  >
  > Mode asinkron tersedia untuk baik Aurora PostgreSQL maupun Aurora MySQL.

- Mode sinkron – Ketika sesi basis data menghasilkan peristiwa aliran aktivitas, sesi ini memblokir aktivitas-aktivitas lain sampai peristiwa itu dijadikan durabel. Jika peristiwa tidak dapat dijadikan durabel karena suatu alasan, sesi basis data akan kembali ke aktivitas normal. Namun, peristiwa RDS dikirim yang menunjukkan bahwa catatan aliran aktivitas mungkin hilang selama beberapa waktu. Peristiwa RDS kedua dikirim setelah sistem kembali berkondisi baik.

  Mode sinkron lebih memprioritaskan akurasi aliran aktivitas daripada performa basis data.

> **Note**
>
> Mode sinkron tersedia untuk Aurora PostgreSQL. Anda tidak dapat menggunakan mode sinkron dengan Aurora MySQL.

## Persyaratan dan batasan untuk aliran aktivitas basis data

Dalam Aurora, aliran aktivitas basis data memiliki persyaratan dan batasan berikut:

- Amazon Kinesis diharuskan untuk aliran aktivitas basis data.
- AWS Key Management Service (AWS KMS) diperlukan untuk aliran aktivitas database karena selalu dienkripsi.
- Menerapkan enkripsi tambahan ke aliran data Amazon Kinesis Anda tidak kompatibel dengan aliran aktivitas database, yang sudah dienkripsi dengan kunci Anda. AWS KMS
- Mulai aliran aktivitas basis data Anda di tingkat klaster basis data. Jika Anda menambahkan instans basis data ke klaster, Anda tidak perlu memulai aliran aktivitas pada instans: instans: aliran diaudit secara otomatis.
- Di basis data global Aurora, pastikan untuk memulai secara terpisah aliran aktivitas pada setiap klaster basis data. Setiap klaster mengirimkan data audit ke aliran Kinesisnya sendiri dalam Wilayah AWS-nya sendiri.
- Di Aurora PostgreSQL, pastikan untuk menghentikan aliran aktivitas database sebelum upgrade versi utama. Anda dapat memulai aliran aktivitas basis data setelah pemutakhiran selesai.

## Wilayah dan ketersediaan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi khusus dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Lihat informasi yang lebih lengkap tentang ketersediaan versi dan Wilayah dengan Aurora dan aliran aktivitas basis data di Daerah yang Didukung dan mesin Aurora DB untuk aliran aktivitas database.

## Kelas-kelas instans basis data yang didukung untuk aliran aktivitas basis data

Untuk Aurora MySQL, Anda dapat menggunakan aliran aktivitas basis data dengan kelas-kelas instans basis data berikut:

- db.r8g. * besar
- db.r7g.*large
- db.r7i. * besar
- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

Untuk Aurora PostgreSQL, Anda dapat menggunakan aliran aktivitas basis data dengan kelas-kelas instans basis data berikut:

- db.r8g. * besar
- db.r7i. * besar
- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large
- db.r5.*large
- db.r4.*large
- db.x2g.*

## Prasyarat jaringan untuk aliran aktivitas basis data Aurora MySQL

Di bagian berikut, Anda dapat menemukan cara mengonfigurasikan cloud privat virtual (VPC) untuk digunakan dengan aliran aktivitas basis data.

> **Note**
>
> Prasyarat jaringan MySQL Aurora berlaku untuk versi mesin berikut:
>
> - Aurora MySQL versi 2, hingga 2.11.3
> - Aurora MySQL versi 2.12.0
> - Aurora MySQL versi 3, hingga 3.04.2

Topik


- Prasyarat untuk titik akhir AWS KMS
- Prasyarat untuk ketersediaan publik
- Prasyarat untuk ketersediaan privat


## Prasyarat untuk titik akhir AWS KMS

Instance dalam klaster MySQL Aurora yang menggunakan aliran aktivitas harus dapat mengakses titik akhir. AWS KMS Pastikan bahwa persyaratan ini terpenuhi sebelum Anda mengaktifkan aliran aktivitas basis data untuk klaster Aurora MySQL. Jika klaster Aurora tersedia untuk umum, persyaratan ini terpenuhi secara otomatis.

> **Important**
>
> Jika klaster DB MySQL Aurora tidak dapat mengakses titik AWS KMS akhir, aliran aktivitas akan berhenti. Dalam hal itu, Aurora memberi tahu Anda tentang masalah lewat Peristiwa RDS.

## Prasyarat untuk ketersediaan publik

Agar menjadi publik, klaster basis data Aurora harus memenuhi persyaratan berikut:

- Dapat Diakses Publik adalah Ya di halaman detail AWS Management Console cluster.
- Klaster basis data berada di subnet publik Amazon VPC. Lihat informasi yang lebih lengkap tentang instans basis data yang dapat diakses publik di Menggunakan klaster DB di VPC. Lihat informasi yang lebih lengkap tentang subnet Amazon VPC publik di VPC dan Subnet Anda.

## Prasyarat untuk ketersediaan privat

Jika klaster Aurora basis data Anda berada di subnet publik VPC dan tidak dapat diakses publik, klaster itu bersifat privat. Untuk menjaga klaster tetap privat dan menggunakannya dengan aliran aktivitas basis data, Anda memiliki opsi-opsi berikut:

- Konfigurasikan Network Address Translation (NAT) di VPC Anda. Lihat informasi yang lebih lengkap di Gateway NAT.
- Buat AWS KMS titik akhir di VPC Anda. Opsi ini disarankan karena lebih mudah dikonfigurasikan.

Untuk membuat AWS KMS titik akhir di VPC Anda

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Di panel navigasi, pilih Titik Akhir.
3. Pilih Buat Titik Akhir.

   Halaman Buat Titik Akhir muncul.
4. Lakukan hal-hal berikut:
   - Dalam Kategori layanan, pilih Layanan AWS .
   - Di Nama Layanan, pilih com.amazonaws. ***region*** .kms, di mana *wilayah* adalah Wilayah AWS tempat cluster Anda berada.
   - Untuk VPC, pilih VPC tempat klaster Anda berada.
5. Pilih Buat Titik Akhir.

Lihat informasi yang lebih lengkap tentang cara mengonfigurasikan titik akhir VPC di Titik Akhir VPC.

## Memulai aliran aktivitas basis data

Untuk memantau aktivitas basis data bagi semua instans dalam klaster basis data Aurora, mulai aliran aktivitas pada tingkat klaster. Setiap instans basis data yang Anda tambahkan ke klaster turut dipantau secara otomatis. Jika Anda menggunakan basis data global Aurora, mulai secara terpisah aliran aktivitas basis data pada setiap klaster basis data. Setiap klaster mengirimkan data audit ke aliran Kinesisnya sendiri dalam Wilayah AWS-nya sendiri.

Ketika Anda memulai aliran aktivitas , setiap peristiwa aktivitas basis data yang Anda konfigurasikan dalam kebijakan audit akan menghasilkan peristiwa aliran aktivitas. Perintah-perintah SQL seperti `CONNECT` dan `SELECT` menghasilkan peristiwa akses. Perintah-perintah SQL seperti `CREATE` dan `INSERT` menghasilkan peristiwa perubahan.

Console

Untuk memulai aliran aktivitas basis data

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster basis data tempat Anda ingin memulai aliran aktivitas.

4. Untuk Tindakan, pilih Mulai aliran aktivitas.

   *name*Jendela Start database activity stream: muncul, di *name* mana cluster DB Anda.

5. Masukkan setelan berikut:

   - Untuk AWS KMS key, pilih kunci dari daftar AWS KMS keys.

     > **Note**
     >
     > Jika klaster Aurora MySQL Anda tidak dapat mengakses kunci KMS, ikuti petunjuk dalam Prasyarat jaringan untuk aliran aktivitas basis data Aurora MySQL untuk mengaktifkan dahulu akses itu.

     Aurora menggunakan kunci KMS untuk mengenkripsi kunci yang pada gilirannya mengenkripsi aktivitas basis data. Pilih kunci KMS selain kunci default. Untuk informasi selengkapnya tentang kunci enkripsi dan AWS KMS, lihat Apa itu AWS Key Management Service? di Panduan AWS Key Management Service Pengembang.

   - Untuk Mode aliran aktivitas basis data, pilih Asinkron atau Sinkron.

     > **Note**
     >
     > Pilihan ini hanya berlaku untuk Aurora PostgreSQL. Untuk Aurora MySQL, Anda hanya bisa menggunakan mode asinkron.

   - Pilih Segera.

     Bila Anda memilih Segera, klaster basis data memulai ulang dengan seketika. Jika Anda memilih Selama jendela pemeliharaan berikutnya, klaster basis data tidak seketika memulai ulang. Dalam hal ini, aliran aktivitas basis data tidak dimulai hingga jendela pemeliharaan berikutnya.

6. Pilih Mulai aliran aktivitas basis data.

   Status untuk klaster basis data menunjukkan bahwa aliran aktivitas dimulai.

> **Note**
>
> Jika Anda mendapatkan kesalahan `You can't start a database activity stream in this configuration`, periksa Kelas-kelas instans basis data yang didukung untuk aliran aktivitas basis data untuk melihat apakah klaster basis data Anda menggunakan kelas instans yang didukung.

AWS CLI

Untuk memulai aliran aktivitas database untuk cluster DB , konfigurasikan DB cluster menggunakan start-activity-stream AWS CLI perintah.

- `--resource-arn` *`arn`* – Menentukan Amazon Resource Name (ARN) klaster basis data.
- `--mode` *`sync-or-async`* – Menentukan mode sinkron (`sync`) atau asinkron (`async`). Untuk Aurora PostgreSQL, Anda dapat memilih salah satu nilai. Untuk Aurora MySQL, pilih `async`.
- `--kms-key-id` *`key`* – Menentukan pengidentifikasi kunci KMS untuk mengenkripsi pesan dalam aliran aktivitas basis data. Pengidentifikasi kunci AWS KMS adalah kunci ARN, ID kunci, alias ARN, atau nama alias untuk. AWS KMS key

Contoh berikut memulai aliran aktivitas basis data untuk sebuah klaster basis data dalam mode asinkron.

Untuk Linux, macOS, atau Unix:

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

Untuk Windows:

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
```

```
    --apply-immediately
```

Amazon RDS API

Untuk memulai aliran aktivitas database untuk , konfigurasikan cluster menggunakan StartActivityStreamoperasi.

Panggil tindakan dengan parameter-parameter di bawah:

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

> **Note**
>
> Jika Anda mendapatkan kesalahan yang menyatakan bahwa Anda tidak dapat memulai aliran aktivitas database dengan versi database Aurora PostgreSQL Anda saat ini, terapkan patch terbaru untuk Aurora PostgreSQL sebelum memulai aliran aktivitas database. Untuk informasi tentang meningkatkan database Aurora PostgreSQL Anda, lihat. Meningkatkan kluster Amazon Aurora DB
> Berikut ini adalah versi patch minimum untuk memulai aliran aktivitas database dengan Aurora PostgreSQL.
>
> - 3.4.15 (11.9.15), 11.21.10
> - 12.9.15, 12.15.9, 12.16.10, 12.17.7, 12.18.5, 12.19.4, 12.20.3, 12.22.3
> - 13.9.12, 13.11.9, 13.12.10, 13.13.7, 13.14.5, 13.15.4, 13.16.3, 13.18.3
> - 14.6.12, 14.8.9, 14.9.10, 14.10.7, 14.11.5, 14.12.4, 14.13.3, 14.15.3
> - 15.3.9, 15.4.10, 15.5.7, 15.6.5, 15.7.4, 15.8.3, 15.10.3
> - 16.1.7, 16.2.5, 16.3.4, 16.4.3, 16.6.3

## Mendapatkan status aliran aktivitas basis data

Anda bisa mendapatkan status aliran aktivitas menggunakan konsol atau AWS CLI.

Konsol

Untuk mendapatkan status aliran aktivitas basis data

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih tautan klaster DB.
3. Pilih tab Konfigurasi, dan periksa Aliran aktivitas basis data untuk statusnya.

AWS CLI

Anda bisa mendapatkan konfigurasi aliran aktivitas untuk klaster DB sebagai respons terhadap permintaan CLI describe-db-clusters .

Contoh berikut menjelaskan *my-cluster*.

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

Contoh berikut menunjukkan respons JSON. Bidang-bidang berikut ditampilkan:

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
-

Bidang-bidang ini sama untuk Aurora PostgreSQL dan Aurora MySQL, kecuali bahwa `ActivityStreamMode` selalu `async` untuk Aurora MySQL, sedangkan untuk Aurora PostgreSQL mungkin `sync` atau `async`.

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
```

```
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

API RDS

Anda bisa mendapatkan konfigurasi aliran aktivitas untuk klaster DB sebagai respons operasi DescribeDBClusters .

## Menghentikan aliran aktivitas basis data

Anda dapat menghentikan aliran aktivitas menggunakan konsol atau AWS CLI.

Jika Anda menghapus klaster basis data, aliran aktivitas dihentikan dan aliran Amazon Kinesis yang mendasari dihapus secara otomatis.

Konsol

Untuk mematikan aliran aktivitas

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih klaster basis data yang ingin Anda hentikan aliran aktivitas basis datanya.
4. Untuk Tindakan, pilih Hentikan aliran aktivitas. Jendela Aliran Aktivitas Basis Data akan muncul.
    a. Pilih Seketika.

        Bila Anda memilih Segera, klaster basis data memulai ulang dengan seketika. Jika Anda memilih Selama jendela pemeliharaan berikutnya, klaster basis data tidak seketika memulai ulang. Dalam hal ini, aliran aktivitas basis data tidak berhenti hingga jendela pemeliharaan berikutnya.
    b. Pilih Lanjutkan.

AWS CLI

Untuk menghentikan aliran aktivitas database untuk DB cluster , konfigurasikan menggunakan AWS CLI perintah stop-activity-stream. Tandai Kawasan AWS untuk klaster basis data dengan menggunakan parameter `--region`. Parameter `--apply-immediately` bersifat opsional.

Untuk Linux, macOS, atau Unix:

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

Untuk Windows:

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

API RDS

Untuk menghentikan aliran aktivitas database untuk DB cluster , konfigurasikan cluster menggunakan StopActivityStreamoperasi. Tandai Kawasan AWS untuk klaster basis data dengan menggunakan parameter `Region`. Parameter `ApplyImmediately` bersifat opsional.

## Memantau aliran aktivitas basis data

Aliran aktivitas basis data memantau dan melaporkan aktivitas. Aliran aktivitas dikumpulkan dan dikirim ke Amazon Kinesis. Dari Kinesis, Anda dapat memantau aliran aktivitas, atau layanan dan aplikasi lain dapat menggunakan aliran aktivitas untuk analisis lebih lanjut. Anda dapat menemukan nama aliran Kinesis yang mendasarinya dengan menggunakan AWS CLI perintah `describe-db-clusters`atau operasi. RDS API `DescribeDBClusters`

Aurora  mengelola aliran Kinesis untuk Anda sebagai berikut:

- Aurora  membuat aliran Kinesis secara otomatis dengan periode retensi 24 jam.
- Aurora  menskalakan aliran Kinesis jika perlu.
- Jika Anda menghentikan aliran aktivitas database atau menghapus cluster DB , Aurora menghapus aliran Kinesis.

Kategori-kategori aktivitas berikut dipantau dan dimasukkan ke dalam log audit aliran aktivitas:

- SQLperintah — Semua SQL perintah diaudit, dan juga pernyataan yang disiapkan, fungsi bawaan, dan fungsi di SQL PL/. Panggilan ke prosedur tersimpan akan diaudit. Setiap SQL pernyataan yang dikeluarkan dalam prosedur atau fungsi yang disimpan juga diaudit.
- Informasi database lainnya — Aktivitas yang dipantau mencakup SQL pernyataan lengkap, jumlah baris baris yang terpengaruh dari DML perintah, objek yang diakses, dan nama database yang unik. Untuk Aurora PostgreSQL, aliran aktivitas database juga memantau variabel pengikat dan parameter prosedur yang disimpan.

  > **Important**
  >
  > SQLTeks lengkap dari setiap pernyataan terlihat di log audit aliran aktivitas, termasuk data sensitif apa pun. Namun, kata sandi pengguna database disunting jika Oracle dapat menentukannya dari konteks, seperti dalam pernyataan berikut. SQL
  >
  > ```
  > ALTER ROLE role-name WITH password
  > ```

- Informasi koneksi – Aktivitas yang dipantau mencakup sesi dan informasi jaringan, ID proses server, dan kode keluar.

Jika aliran aktivitas mengalami kegagalan saat memantau instans DB Anda, Anda akan diberi tahu melalui RDS acara.

Di bagian berikut, Anda dapat mengakses, mengaudit, dan memproses aliran aktivitas basis data.

Topik

- Mengakses aliran aktivitas dari Amazon Kinesis
- Audit isi log dan contoh untuk aliran aktivitas database
- databaseActivityEventDaftar array JSON untuk aliran aktivitas database
- Memproses aliran aktivitas database menggunakan AWS SDK

## Mengakses aliran aktivitas dari Amazon Kinesis

Saat Anda mengaktifkan aliran aktivitas untuk klaster basis data, aliran Kinesis dibuat untuk Anda. Dari Kinesis, Anda dapat memantau aktivitas basis data Anda secara waktu nyata. Untuk menganalisis lebih lanjut aktivitas basis data, Anda dapat menghubungkan aliran Kinesis dengan aplikasi konsumen. Anda juga dapat menghubungkan aliran ke aplikasi manajemen kepatuhan

seperti IBMSecurity Guardium atau Audit SecureSphere Database Imperva dan Perlindungan Keamanan Guardium atau Audit dan Perlindungan IBM .

Anda dapat mengakses aliran Kinesis Anda baik dari RDS konsol atau konsol Kinesis.

Untuk mengakses aliran aktivitas dari Kinesis menggunakan konsol RDS

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih cluster DB tempat Anda memulai aliran aktivitas.
4. Pilih Konfigurasi.
5. Di bawah Aliran aktivitas basis data, pilih tautan di bawah Aliran Kinesis.
6. Di konsol Kinesis, pilih Pemantauan untuk mulai mengamati aktivitas basis data.

Untuk mengakses aliran aktivitas dari Kinesis dengan menggunakan konsol Kinesis

1. Buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih aliran aktivitas Anda dari daftar aliran Kinesis.

   Nama aliran aktivitas mencakup awalan `aws-rds-das-cluster-` diikuti dengan ID sumber daya klaster basis data. Berikut sebuah contohnya.

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

   Untuk menggunakan RDS konsol Amazon untuk menemukan ID sumber daya untuk DB cluster , pilih dari daftar database, lalu pilih tab Konfigurasi.

   Untuk menggunakan AWS CLI untuk menemukan nama aliran Kinesis lengkap untuk aliran aktivitas, gunakan describe-db-clustersCLIpermintaan dan catat nilai `ActivityStreamKinesisStreamName` dalam respons.
3. Pilih Pemantauan untuk mulai mengamati aktivitas basis data.

Lihat informasi yang lebih lengkap tentang penggunaan Amazon Kinesis di Apakah Amazon Kinesis Data Streams?.

## Audit isi log dan contoh untuk aliran aktivitas database

Peristiwa-peristiwa yang dipantau disajikan dalam aliran aktivitas basis data berupa string JSON. Strukturnya terdiri atas objek JSON yang berisi `DatabaseActivityMonitoringRecord`, yang selanjutnya berisi sebuah array peristiwa aktivitas `databaseActivityEventList`.

> **Note**
>
> Untuk aliran aktivitas database, array `paramList` JSON tidak menyertakan nilai null dari aplikasi Hibernate.

Topik

- Contoh-contoh log audit untuk aliran aktivitas
- Objek JSON DatabaseActivityMonitoringRecords
- databaseActivityEvents Objek JSON

### Contoh-contoh log audit untuk aliran aktivitas

Berikut adalah contoh log audit JSON terdekripsi dari catatan peristiwa aktivitas.

**Example Catatan peristiwa aktivitas pernyataan SQL CONNECT Aurora PostgreSQL**

Catatan peristiwa aktivitas berikut menunjukkan upaya masuk dengan penggunaan pernyataan SQL `CONNECT` (`command`) oleh klien psql (`clientApplication`).

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-10-30 00:39:49.940668+00",
          "logTime": "2019-10-30 00:39:49.990579+00",
          "statementId": 1,
          "substatementId": 1,
```

```
          "objectType": null,
          "command": "CONNECT",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "49804",
          "sessionId": "5ce5f7f0.474b",
          "rowCount": null,
          "commandText": null,
          "paramList": [],
          "pid": 18251,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "MISC",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Catatan peristiwa aktivitas pernyataan SQL CONNECT Aurora MySQL

Catatan peristiwa aktivitas berikut menunjukkan upaya masuk dengan penggunaan pernyataan SQL `CONNECT` (`command`) oleh klien mysql (`clientApplication`).

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
```

```
      "pid":2830,
      "dbUserName":"rdsadmin",
      "databaseName":"",
      "remoteHost":"localhost",
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```


Example Catatan peristiwa aktivitas pernyataan CREATE TABLE Aurora PostgreSQL

Contoh berikut menunjukkan peristiwa `CREATE TABLE` untuk Aurora PostgreSQL.


```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
```

```
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
          "statementId": 2,
          "substatementId": 1,
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Catatan peristiwa aktivitas pernyataan CREATE TABLE Aurora MySQL

Contoh berikut menunjukkan pernyataan `CREATE TABLE` untuk Aurora MySQL. Operasi ini disajikan sebagai dua catatan peristiwa terpisah. Satu peristiwa memiliki `"class":"MAIN"`. Peristiwa yang lain memiliki `"class":"AUX"`. Pesan-pesan mungkin tiba dengan sebarang urutan. Bidang `logTime` peristiwa `MAIN` selalu lebih awal dari bidang-bidang `logTime` untuk peristiwa `AUX` yang terkait.

Contoh berikut menunjukkan peristiwa dengan nilai `class` berupa `MAIN`.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Contoh berikut menunjukkan peristiwa yang bersangkutan dengan nilai `class` berupa AUX.

```
{
  "type":"DatabaseActivityMonitoringRecord",
```

```
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Example Catatan peristiwa aktivitas pernyataan SELECT Aurora PostgreSQL

Contoh berikut menunjukkan peristiwa `SELECT` .

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
```

```
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```



```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
```

```
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
```

```
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Catatan peristiwa aktivitas pernyataan SELECT Aurora MySQL

Contoh berikut menunjukkan peristiwa `SELECT`.

Contoh berikut menunjukkan peristiwa dengan nilai `class` berupa `MAIN`.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
```

```
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"726571",
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Contoh berikut menunjukkan peristiwa yang bersangkutan dengan nilai `class` berupa AUX.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"READ",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
```

```
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Objek JSON DatabaseActivityMonitoringRecords

Catatan peristiwa aktivitas basis data berada dalam objek JSON yang berisi informasi berikut.

| Bidang JSON | Tipe data | Deskripsi |
| --- | --- | --- |
| `type` | string | Jenis catatan JSON. Nilainya adalah `DatabaseActivityMonitoringRecords` . |
| `version` | string | Versi catatan pemantauan aktivitas basis data.<br><br>Versi catatan aktivitas basis data yang dihasilkan bergantung pada versi mesin klaster basis data:<br><br>• Catatan aktivitas basis data versi 1.1 dihasilkan untuk klaster basis data Aurora PostgreSQL yang menjalankan versi mesin 10.10 dan versi kecil yang lebih baru serta versi mesin 11.5 dan yang lebih baru. |

| Bidang JSON | Tipe data | Deskripsi |
| --- | --- | --- |
| | | • Catatan aktivitas basis data versi 1.0 dihasilkan untuk klaster basis data Aurora PostgreSQL yang menjalankan versi mesin 10.7 dan 11.4.<br><br>Semua bidang berikut ada dalam versi 1.0 dan versi 1.1 kecuali jika diberi catatan khusus. |
| databaseActivityEvents | string | Objek JSON yang berisi peristiwa aktivitas. |
| kunci | string | Kunci enkripsi yang Anda gunakan untuk mendekripsi databaseActivityEventDaftar array JSON |

databaseActivityEvents Objek JSON

Objek JSON `databaseActivityEvents` berisi informasi berikut.

Bidang-bidang tingkat atas dalam catatan JSON

Setiap peristiwa dalam log audit dibungkus dalam sebuah catatan dalam format JSON. Catatan ini berisi bidang-bidang berikut.

tipe

Bidang ini selalu memiliki nilai `DatabaseActivityMonitoringRecords`.

versi

Bidang ini mewakili versi protokol atau kontrak data aliran aktivitas basis data. Versi menentukan bidang-bidang yang tersedia.

Versi 1.0 mewakili dukungan aliran aktivitas data asli untuk Aurora PostgreSQL versi 10.7 dan 11.4. Versi 1.1 mewakili dukungan aliran aktivitas data untuk Aurora PostgreSQL versi 10.10 dan yang lebih tinggi serta Aurora PostgreSQL versi 11.5 dan yang lebih tinggi. Versi 1.1 mencakup bidang-bidang tambahan `errorMessage` dan `startTime`. Versi 1.2 mewakili dukungan aliran

aktivitas data untuk Aurora MySQL 2.08 dan lebih tinggi. Versi 1.2 mencakup bidang-bidang tambahan `endTime` dan `transactionId`.

databaseActivityEvents

String terenkripsi yang mewakili satu atau beberapa peristiwa aktivitas. String disajikan berupa array byte base64. Saat Anda mendekripsi string, hasilnya adalah catatan dalam format JSON dengan bidang-bidang seperti ditunjukkan dalam contoh di bagian ini.

kunci

Kunci data terenkripsi yang digunakan untuk mengenkripsi string `databaseActivityEvents`. Ini sama dengan AWS KMS key yang Anda berikan saat memulai aliran aktivitas database.

Contoh berikut menunjukkan format catatan ini.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

Lakukan langkah-langkah berikut untuk mendekripsi konten bidang `databaseActivityEvents`:

1. Lakukan dekripsi nilai dalam bidang JSON `key` dengan menggunakan kunci KMS yang Anda sediakan ketika memulai aliran aktivitas basis data. Melakukan hal itu akan menghasilkan kunci enkripsi data berupa teks jelas.
2. Base64 mendekode nilai dalam bidang JSON `databaseActivityEvents` untuk mendapatkan teks sandi, dalam format biner, dari Panduan Referensi audit.
3. Lakukan dekripsi teks sandi biner dengan kunci enkripsi data yang Anda dekode pada langkah pertama.
4. Lakukan dekompresi Panduan Referensi yang terdekripsi.
   - Muatan terenkripsi ada di bidang `databaseActivityEvents`.
   - Bidang `databaseActivityEventList` berisi array catatan audit. Bidang `type` dalam array dapat berupa `record` atau `heartbeat`.

Catatan peristiwa aktivitas log audit adalah objek JSON yang berisi informasi berikut.

| Bidang JSON | Tipe data | Deskripsi |
| --- | --- | --- |
| `type` | string | Jenis catatan JSON. Nilainya adalah `DatabaseActivityMonitoringRecord` . |
| `clusterId` | string | Pengidentifikasi sumber daya klaster basis data. Pengidentifikasi ini berkaitan dengan atribut klaster basis data `DbClusterResourceId` . |
| `instanceId` | string | Pengidentifikasi sumber daya instans basis data. Pengidentifikasi ini berkaitan dengan atribut instans basis data `DbiResourceId` . |
| databaseActivityEventDaftar array JSON | string | Array catatan audit aktivitas atau pesan heartbeat. |

## databaseActivityEventDaftar array JSON untuk aliran aktivitas database

Muatan log audit adalah array JSON `databaseActivityEventList` terenkripsi. Tabel-tabel berikut memerinci secara alfabetis bidang-bidang untuk setiap peristiwa aktivitas dalam array `DatabaseActivityEventList` terdekripsi sebuah log audit. Bidang-bidang akan berbeda tergantung pada apakah Anda menggunakan Aurora PostgreSQL atau Aurora MySQL. Rujuk ke tabel yang relevan dengan mesin basis data Anda.

> **Important**
>
> Struktur peristiwa dapat berubah sewaktu-waktu. Aurora mungkin menambahkan bidang-bidang baru ke peristiwa aktivitas di masa mendatang. Dalam aplikasi yang menguraikan data JSON, pastikan bahwa kode Anda dapat mengabaikan atau mengambil tindakan yang tepat untuk nama-nama bidang yang tidak dikenal.

### databaseActivityEventKolom daftar untuk Aurora PostgreSQL

Berikut ini adalah `databaseActivityEventList` bidang untuk Aurora PostgreSQL.

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `class` | string | Kelas peristiwa aktivitas. Nilai-nilai yang valid untuk Aurora PostgreSQL adalah:<br>• `ALL`<br>• `CONNECT` – Peristiwa koneksi atau pemutusan.<br>• `DDL` – Pernyataan DDL yang tidak termasuk dalam daftar pernyataan untuk kelas `ROLE`.<br>• `FUNCTION` – Panggilan fungsi atau blok `DO`.<br>• `MISC` – Perintah lain-lain, seperti `DISCARD`, `FETCH`, `CHECKPOINT` , atau `VACUUM`.<br>• `NONE`<br>• `READ` – Pernyataan `SELECT` atau `COPY` apabila sumbernya sebuah relasi atau kueri.<br>• `ROLE` – Pernyataan seputar peran dan privilese, yang meliputi `GRANT`, `REVOKE`, dan `CREATE/ALTER/DROP ROLE`.<br>• `WRITE` – Pernyataan `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, atau `COPY` ketika tujuan adalah relasi. |
| `clientApplication` | string | Aplikasi yang digunakan klien untuk menghubungi seperti dilaporkan oleh klien. Klien tidak wajib memberikan informasi ini, sehingga nilainya dapat null. |
| `command` | string | Nama perintah SQL tanpa perincian perintah sama sekali. |
| `commandText` | string | Pernyataan SQL sebenarnya yang diberikan oleh pengguna. Untuk Aurora PostgreSQL, nilainya identik dengan pernyataan SQL asli. Bidang ini digunakan untuk semua jenis catatan kecuali untuk menghubungkan atau memutus catatan, yang dalam hal ini nilainya null.<br>**Important**<br>Teks SQL lengkap setiap pernyataan, yang meliputi semua data sensitif, dapat dilihat di log audit aliran |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| | | aktivitas. Namun, kata sandi pengguna basis data disensor jika Aurora dapat menentukannya dari konteks, seperti dalam pernyataan SQL berikut.<br>`ALTER ROLE role-name WITH password` |
| `databaseName` | string | Basis data yang terhubung dengan pengguna. |
| `dbProtocol` | string | Protokol basis data, misalnya `Postgres 3.0`. |
| `dbUserName` | string | Pengguna basis data yang diautentikasi oleh klien. |
| `errorMessage`<br><br>(khusus catatan aktivitas basis data 1.1) | string | Jika ada kesalahan, bidang ini diisi dengan pesan kesalahan yang akan dihasilkan oleh server basis data. Nilai `errorMessage` adalah null untuk pernyataan normal yang tidak mengakibatkan kesalahan.<br><br>Kesalahan didefinisikan sebagai sebarang aktivitas yang akan menghasilkan peristiwa log kesalahan PostgreSQL yang dapat dilihat klien pada tingkat keparahan `ERROR` atau lebih tinggi. Lihat informasi yang lebih lengkap di Tingkat Keparahan Pesan PostgreSQL. Misalnya, kesalahan sintaks dan pembatalan kueri menghasilkan pesan kesalahan.<br><br>Kesalahan-kesalahan internal server PostgreSQL, seperti kesalahan proses penunjuk pemeriksaan latar belakang tidak menghasilkan pesan kesalahan. Namun, catatan untuk peristiwa seperti itu masih dipancarkan terlepas dari setelan level keparahan log. Hal ini mencegah penyerang mematikan pencatatan log untuk mencoba menghindari deteksi.<br><br>Lihat juga bidang `exitCode`. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `exitCode` | int | Nilai yang digunakan untuk catatan keluar sesi. Pada keluar yang mulus, ini berisi kode keluar. Kode keluar tidak selalu dapat diperoleh dalam beberapa skenario kegagalan. Contoh-contoh adalah jika PostgreSQL melakukan `exit()` atau jika operator melakukan perintah seperti `kill -9`.<br><br>Jika ada kesalahan apa pun, bidang `exitCode` menampilk an kode kesalahan SQL, `SQLSTATE`, sebagaimana tercantum dalam Kode Kesalahan PostgreSQL.<br><br>Lihat juga bidang `errorMessage` . |
| `logTime` | string | Stempel waktu seperti tercatat di jalur kode audit. Ini mewakili waktu akhir eksekusi pernyataan SQL. Lihat juga bidang `startTime` . |
| `netProtocol` | string | Protokol komunikasi jaringan. |
| `objectName` | string | Nama objek basis data jika pernyataan SQL beroperasi pada objek itu. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| objectType | string | Jenis objek basis data, seperti tabel, indeks, tampilan, dan sebagainya. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null. Nilai-nilai yang valid meliputi:<br>• COMPOSITE TYPE<br>• FOREIGN TABLE<br>• FUNCTION<br>• INDEX<br>• MATERIALIZED VIEW<br>• SEQUENCE<br>• TABLE<br>• TOAST TABLE<br>• VIEW<br>• UNKNOWN |
| paramList | string | Array parameter terpisah koma yang disampaikan ke pernyataan SQL. Jika pernyataan SQL tidak memiliki parameter, nilai ini adalah array kosong. |
| pid | int | ID proses untuk proses backend yang dialokasikan untuk melayani koneksi klien. |
| remoteHost | string | Alamat IP atau nama host klien. Untuk Aurora PostgreSQL, informasi yang digunakan bergantung pada setelan parameter log_hostname basis data. remoteHost Nilai juga termasuk [local] dan localhost yang menunjukkan aktivitas dari rdsadmin pengguna. |
| remotePort | string | Nomor port klien. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `rowCount` | int | Jumlah baris tabel yang terpengaruh atau diambil oleh pernyataan SQL. Bidang ini hanya digunakan untuk pernyataan SQL yang merupakan pernyataan bahasa manipulasi data (DML). Jika pernyataan SQL bukan pernyataan DML, nilai ini null. |
| `serverHost` | string | Alamat IP host server basis data. `serverHost` Nilai juga termasuk `[local]` dan `localhost` yang menunjukkan aktivitas dari `rdsadmin` pengguna. |
| `serverType` | string | Jenis server basis data, misalnya `PostgreSQL` . |
| `serverVersion` | string | Versi server basis data, misalnya `2.3.1` untuk Aurora PostgreSQL. |
| `serviceName` | string | Nama layanan, misalnya `Amazon Aurora PostgreSQL-Compatible edition` . |
| `sessionId` | int | Pengidentifikasi sesi unik semu. |
| `sessionId` | int | Pengidentifikasi sesi unik semu. |
| `startTime`<br><br>(khusus catatan aktivitas basis data 1.1) | string | Waktu ketika eksekusi dimulai untuk pernyataan SQL.<br><br>Untuk menghitung waktu eksekusi kira-kira pernyataan SQL, gunakan `logTime - startTime` . Lihat juga bidang `logTime`. |
| `statementId` | int | Pengidentifikasi untuk pernyataan SQL klien. Penghitung berada di tingkat sesi dan bertambah dengan setiap pernyataan SQL yang dimasukkan oleh klien. |
| `substatementId` | int | Pengidentifikasi untuk subpernyataan SQL. Nilai ini menghitung subpernyataan terkandung untuk setiap pernyataan SQL yang diidentifikasi oleh bidang `statementId` . |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `type` | string | Jenis peristiwa. Nilai-nilai yang valid adalah `record` atau `heartbeat` . |

databaseActivityEventDaftar bidang untuk Aurora MySQL

Berikut ini adalah `databaseActivityEventList` bidang untuk Aurora MySQL.

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `class` | string | Kelas peristiwa aktivitas.<br><br>Nilai-nilai yang valid untuk Aurora MySQL adalah:<br><br>• `MAIN` – Peristiwa utama yang mewakili pernyataan SQL.<br>• `AUX` – Peristiwa tambahan yang berisi detail tambahan. Misalnya, pernyataan yang mengganti nama objek mungkin memiliki peristiwa dengan kelas `AUX` yang mencerminkan nama baru.<br><br>Untuk menemukan peristiwa-peristiwa `MAIN` dan `AUX` yang berkaitan dengan pernyataan yang sama, periksa berbagai peristiwa yang memiliki nilai yang sama untuk bidang `pid` dan untuk bidang `statementId` . |
| `clientApplication` | string | Aplikasi yang digunakan klien untuk menghubungi seperti dilaporkan oleh klien. Klien tidak wajib memberikan informasi ini, sehingga nilainya dapat null. |
| `command` | string | Kategori umum pernyataan SQL. Nilai untuk bidang ini bergantung pada nilai `class`.<br><br>Nilai-nilai ketika `class` adalah `MAIN` meliputi:<br><br>• `CONNECT` – Ketika sesi klien terhubung.<br>• `QUERY` – Pernyataan SQL. Disertai oleh satu atau beberapa peristiwa dengan nilai `class` adalah `AUX`. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| | | • DISCONNECT – Ketika sesi klien terputus.<br>• FAILED_CONNECT – Ketika klien mencoba menghubung, tetapi tidak dapat.<br>• CHANGEUSER – Perubahan status yang merupakan bagian dari protokol jaringan MySQL, bukan dari pernyataan yang Anda terbitkan.<br><br>Nilai-nilai ketika class adalah AUX meliputi:<br>• READ – Pernyataan SELECT atau COPY apabila sumbernya sebuah relasi atau kueri.<br>• WRITE – Pernyataan INSERT, UPDATE, DELETE, TRUNCATE, atau COPY apabila tujuannya sebuah relasi.<br>• DROP – Menghapus objek.<br>• CREATE – Membuat objek.<br>• RENAME – Mengganti nama objek.<br>• ALTER – Mengubah properti-properti suatu objek. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `commandText` | string | Untuk peristiwa dengan nilai `class` adalah `MAIN`, bidang ini mewakili pernyataan SQL sebenarnya yang diberikan oleh pengguna. Bidang ini digunakan untuk semua jenis catatan kecuali untuk menghubungkan atau memutus catatan, yang dalam hal ini nilainya null.<br><br>Untuk peristiwa dengan nilai `class` adalah `AUX`, bidang ini memuat informasi tambahan tentang objek yang terlibat dalam peristiwa.<br><br>Untuk Aurora MySQL, karakter-karakter seperti tanda kutip didahului oleh garis miring kiri, yang mewakili karakter escape.<br><br>**Important**<br>Teks SQL lengkap setiap pernyataan, yang meliputi semua data sensitif, dapat dilihat di log audit. Namun, kata sandi pengguna basis data disensor jika Aurora dapat menentukannya dari konteks, seperti dalam pernyataan SQL berikut.<br>`mysql> SET PASSWORD = 'my-password ';`<br>**Note**<br>Tetapkan kata sandi selain penggugah/ prompt yang ditampilkan di sini sebagai praktik terbaik keamanan. |
| `databaseName` | string | Basis data yang terhubung dengan pengguna. |
| `dbProtocol` | string | Protokol basis data. Saat ini, nilai ini selalu `MySQL` untuk Aurora MySQL. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `dbUserName` | string | Pengguna basis data yang diautentikasi oleh klien. |
| `endTime`<br><br>(khusus catatan aktivitas basis data 1.2) | string | Waktu ketika eksekusi diakhiri untuk pernyataan SQL. Disajikan dalam format Waktu Universal Terkoordinasi (UTC).<br><br>Untuk menghitung waktu eksekusi pernyataan SQL, gunakan `endTime - startTime` . Lihat juga bidang `startTime` . |
| `errorMessage`<br><br>(khusus catatan aktivitas basis data 1.1) | string | Jika ada kesalahan, bidang ini diisi dengan pesan kesalahan yang akan dihasilkan oleh server basis data. Nilai `errorMessage` adalah null untuk pernyataan normal yang tidak mengakibatkan kesalahan.<br><br>Kesalahan didefinisikan sebagai sebarang aktivitas yang akan menghasilkan peristiwa log kesalahan MySQL yang dapat dilihat klien pada tingkat keparahan `ERROR` atau lebih tinggi. Lihat informasi yang lebih lengkap di The Error Log dalam Panduan Referensi MySQL. Misalnya, kesalahan sintaks dan pembatalan kueri menghasilkan pesan kesalahan .<br><br>Kesalahan-kesalahan internal server MySQL, seperti kesalahan proses penunjuk pemeriksaan latar belakang tidak menghasilkan pesan kesalahan. Namun, catatan untuk peristiwa seperti itu masih dipancarkan terlepas dari setelan level keparahan log. Hal ini mencegah penyerang mematikan pencatatan log untuk mencoba menghindari deteksi.<br><br>Lihat juga bidang `exitCode`. |
| `exitCode` | int | Nilai yang digunakan untuk catatan keluar sesi. Pada keluar yang mulus, ini berisi kode keluar. Kode keluar tidak selalu dapat diperoleh dalam beberapa skenario kegagalan. Dalam kasus-kasus tersebut, nilai ini mungkin nol atau mungkin kosong. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `logTime` | string | Stempel waktu seperti tercatat di jalur kode audit. Disajikan dalam format Waktu Universal Terkoordinasi (UTC). Lihat cara paling akurat menghitung durasi pernyataan di bidang-bidang `startTime` dan `endTime`. |
| `netProtocol` | string | Protokol komunikasi jaringan. Saat ini, nilai ini selalu `TCP` untuk Aurora MySQL. |
| `objectName` | string | Nama objek basis data jika pernyataan SQL beroperasi pada objek itu. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null. Untuk membangun nama objek berkualifikasi penuh, gabungkan `databaseName` dan `objectName`. Jika kueri melibatkan beberapa objek, bidang ini dapat berupa daftar nama terpisah koma. |
| `objectType` | string | Jenis objek basis data, seperti tabel, indeks, dan sebagainya. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null.<br><br>Nilai-nilai yang valid untuk Aurora MySQL meliputi:<br><br>• `INDEX`<br>• `TABLE`<br>• `UNKNOWN` |
| `paramList` | string | Bidang ini tidak digunakan untuk Aurora MySQL dan selalu null. |
| `pid` | int | ID proses untuk proses backend yang dialokasikan untuk melayani koneksi klien. Ketika server basis data dimulai ulang, `pid` berubah dan penghitung untuk bidang `statementId` dimulai ulang dari awal. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `remoteHost` | string | Alamat IP atau nama host klien yang menerbitkan pernyataan SQL. Untuk Aurora MySQL, informasi yang digunakan bergantung pada setelan parameter `skip_name_resolve` basis data. Nilai `localhost` menunjukkan aktivitas dari pengguna khusus `rdsadmin`. |
| `remotePort` | string | Nomor port klien. |
| `rowCount` | int | Jumlah baris yang dihasilkan oleh pernyataan SQL. Misalnya, jika pernyataan SELECT menghasilkan 10 baris, rowCount adalah 10. Untuk pernyataan INSERT atau UPDATE, rowCount adalah 0. |
| `serverHost` | string | Pengidentifikasi instans server basis data. |
| `serverType` | string | Jenis server basis data, misalnya `MySQL`. |
| `serverVersion` | string | Versi server basis data. Saat ini, nilai ini selalu `MySQL 5.7.12` untuk Aurora MySQL. |
| `serviceName` | string | Nama layanan. Saat ini, nilai ini selalu `Amazon Aurora MySQL` untuk Aurora MySQL. |
| `sessionId` | int | Pengidentifikasi sesi unik semu. |
| `startTime`<br><br>(khusus catatan aktivitas basis data 1.1) | string | Waktu ketika eksekusi dimulai untuk pernyataan SQL. Disajikan dalam format Waktu Universal Terkoordinasi (UTC).<br><br>Untuk menghitung waktu eksekusi pernyataan SQL, gunakan `endTime - startTime` . Lihat juga bidang `endTime`. |
| `statementId` | int | Pengidentifikasi untuk pernyataan SQL klien. Penghitung bertambah dengan setiap pernyataan SQL yang dimasukkan oleh klien. Penghitung dinolkan saat instans basis data dimulai ulang. |

| Field | Tipe data | Deskripsi |
| --- | --- | --- |
| `substatementId` | int | Pengidentifikasi untuk subpernyataan SQL. Nilai ini adalah 1 untuk peristiwa dengan kelas `MAIN` dan 2 untuk peristiwa dengan kelas `AUX`. Gunakan bidang `statementId` untuk mengidentifikasi semua peristiwa yang dihasilkan oleh pernyataan yang sama. |
| `transactionId`<br><br>(khusus catatan aktivitas basis data 1.2) | int | Pengidentifikasi untuk transaksi. |
| `type` | string | Jenis peristiwa. Nilai-nilai yang valid adalah `record` atau `heartbeat` . |

## Memproses aliran aktivitas database menggunakan AWS SDK

Anda dapat memproses aliran aktivitas secara terprogram dengan menggunakan file. AWS SDK Berikut adalah contoh Java dan Python yang berfungsi penuh tentang cara Anda dapat memproses aliran data Kinesis.

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;
```

```
import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
```

```
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
```

```
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {

        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }
```

```
        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }

        private void processSingleBlob(final ByteBuffer bytes) {
            try {
                // JSON $Activity
                final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

                // Base64.Decode
                final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
                final byte[] decodedDataKey = Base64.decode(activity.key);

                Map<String, String> context = new HashMap<>();
                context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

                // Decrypt
```

```
                final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
                final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
                final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

                // GZip Decompress
                final byte[] decompressed = decompress(decrypted);
                // JSON $ActivityRecords
                final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

                // Iterate throught $ActivityEvents
                for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                    System.out.println(GSON.toJson(event));
                }
            } catch (Exception e) {
                // Handle error.
                e.printStackTrace();
            }
        }

        private static byte[] decompress(final byte[] src) throws IOException {
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
                    break;
                } catch (ShutdownException se) {
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
```

```
                    // Backoff and re-attempt checkpoint upon transient failures
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }

    private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
        // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
        final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
```

```
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
 CREDENTIALS_PROVIDER, workerId);

 kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
 APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
    }
}
```

## Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
```

```
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)
```

```
    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters


if __name__ == '__main__':
    main()
```

## Contoh kebijakan IAM untuk aliran aktivitas database

Semua tindakan ini dimasukkan ke dalam log audit aliran. Untuk praktik kepatuhan terbaik, kami menyarankan agar Anda tidak memberikan hak istimewa ini. DBAs

Anda mengatur akses ke aliran aktivitas basis data dengan menggunakan kebijakan IAM. Lihat informasi yang lebih lengkap tentang autentikasi Aurora di Manajemen identitas dan akses untuk Aurora. Lihat informasi yang lebih lengkap tentang pembuatan kebijakan IAM di Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM.

Example Kebijakan untuk memungkinkan pengonfigurasian aliran aktivitas basis data

Untuk memberi pengguna akses terperinci untuk mengubah aliran aktivitas, gunakan kunci-kunci konteks operasi khusus layanan `rds:StartActivityStream` dan `rds:StopActivityStream` dalam kebijakan IAM. Contoh kebijakan IAM berikut memungkinkan pengguna atau peran mengonfigurasikan aliran aktivitas.

```
{
   "Version":"2012-10-17",
   "Statement":[
        {
            "Sid":"ConfigureActivityStreams",
            "Effect":"Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource":"*",
          }
     ]
}
```

Example Kebijakan untuk memungkinkan pemulaian aliran aktivitas basis data

Contoh kebijakan IAM berikut memungkinkan pengguna atau peran memulai aliran aktivitas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example Kebijakan untuk memungkinkan penghentian aliran aktivitas basis data

Contoh kebijakan IAM berikut memungkinkan pengguna atau peran menghentikan aliran aktivitas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Kebijakan untuk menolak pemulaian aliran aktivitas basis data

Contoh kebijakan IAM berikut mencegah pengguna atau peran dari memulai aliran aktivitas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Kebijakan untuk menolak penghentian aliran aktivitas basis data

Contoh kebijakan IAM berikut mencegah pengguna atau peran dari menghentikan aliran aktivitas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

# Memantau ancaman dengan Perlindungan Amazon GuardDuty RDS untuk Amazon Aurora

Amazon GuardDuty adalah layanan deteksi ancaman yang membantu melindungi akun, wadah, beban kerja, dan data di AWS lingkungan Anda. Menggunakan model machine learning (ML), serta kemampuan deteksi anomali dan ancaman, GuardDuty terus memantau berbagai sumber log dan aktivitas runtime untuk mengidentifikasi dan memprioritaskan potensi risiko keamanan dan aktivitas berbahaya di lingkungan Anda.

GuardDuty RDS Protection menganalisis dan memprofilkan peristiwa login untuk potensi ancaman akses ke database Amazon Aurora RDS Anda.  Perlindungan RDS memantau peristiwa ini dan memprofilkannya untuk potensi ancaman orang dalam atau aktor eksternal.

Untuk informasi selengkapnya tentang mengaktifkan Perlindungan GuardDuty RDS, lihat Perlindungan GuardDuty RDS di Panduan Pengguna Amazon GuardDuty .

Ketika RDS Protection mendeteksi potensi ancaman, seperti pola yang tidak biasa dalam upaya login yang berhasil atau gagal, GuardDuty menghasilkan temuan baru dengan rincian tentang database yang berpotensi dikompromikan. Anda dapat melihat detail temuan di bagian ringkasan pencarian di GuardDuty konsol Amazon. Detail temuan bervariasi berdasarkan jenis temuan. Detail utama, jenis sumber daya, dan peran sumber daya, menentukan jenis informasi yang tersedia untuk temuan apa pun. Untuk informasi selengkapnya tentang detail umum yang tersedia untuk temuan dan jenis temuan, lihat Detail Menemukan dan jenis temuan Perlindungan GuardDuty RDS masing-masing di Panduan GuardDuty Pengguna Amazon.

Anda dapat mengaktifkan atau menonaktifkan fitur Perlindungan RDS untuk fitur mana pun Akun AWS di Wilayah AWS mana pun fitur ini tersedia. Ketika Perlindungan RDS tidak diaktifkan, GuardDuty tidak mendeteksi database Aurora yang berpotensi dikompromikan atau memberikan rincian kompromi.

GuardDuty Akun yang ada dapat mengaktifkan Perlindungan RDS dengan masa uji coba 30 hari. Untuk GuardDuty akun baru, Perlindungan RDS sudah diaktifkan dan termasuk dalam periode uji coba gratis 30 hari. Untuk informasi selengkapnya, lihat Memperkirakan GuardDuty biaya di Panduan GuardDuty Pengguna Amazon.

Untuk informasi tentang Wilayah AWS tempat yang GuardDuty belum mendukung Perlindungan RDS, lihat Ketersediaan fitur khusus wilayah di Panduan Pengguna Amazon GuardDuty .

Untuk informasi tentang versi database Aurora yang didukung Perlindungan GuardDuty RDS, lihat basis data Amazon Aurora, Amazon RDS, dan Aurora Limitless yang Didukung di Panduan Pengguna Amazon. GuardDuty

# Menggunakan Amazon Aurora MySQL

Amazon Aurora MySQL adalah mesin basis data relasional yang terkelola sepenuhnya dan kompatibel dengan MySQL. Mesin basis data ini menggabungkan kecepatan dan keandalan basis data komersial kelas atas dengan kesederhanaan dan efektivitas biaya basis data sumber terbuka. Aurora MySQL adalah pengganti "drop-in" untuk MySQL serta memudahkan dan menghemat biaya untuk menyiapkan, mengoperasikan, dan menskalakan deployment MySQL baru dan yang sudah ada, sehingga Anda bebas untuk fokus pada bisnis dan aplikasi Anda. Amazon RDS menyediakan administrasi untuk Aurora dengan menangani tugas basis data rutin seperti penyediaan, patching, pencadangan, pemulihan, deteksi kegagalan, dan perbaikan. Amazon RDS juga menyediakan alat migrasi yang memerlukan hanya beberapa klik untuk mengonversi aplikasi Amazon RDS for MySQL Anda yang ada menjadi aplikasi Aurora MySQL.

Topik


- Gambaran umum Amazon Aurora MySQL
- Keamanan dengan Amazon Aurora MySQL
- Memperbarui aplikasi untuk terhubung ke klaster DB Aurora MySQL menggunakan sertifikat TLS baru
- Menggunakan otentikasi Kerberos untuk Aurora My SQL
- Migrasi data ke cluster Amazon Aurora My DB SQL
- Mengelola Amazon Aurora MySQL
- Menyesuaikan Aurora MySQL
- Kueri paralel untuk Amazon Aurora My SQL
- Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL
- Replikasi dengan Amazon Aurora MySQL
- Menggunakan penerusan tulis lokal di klaster DB Amazon Aurora MySQL
- Mengintegrasikan Amazon Aurora SQL My dengan layanan lain AWS
- Mode lab Amazon Aurora MySQL
- Praktik terbaik dengan Amazon Aurora My SQL
- Memecahkan masalah Amazon Aurora Kinerja database saya SQL
- Amazon Aurora Referensi saya SQL
- Pembaruan mesin database untuk Amazon Aurora My SQL

# Gambaran umum Amazon Aurora MySQL

Bagian berikut menyediakan gambaran umum Amazon Aurora MySQL.

Topik


- Peningkatan performa Amazon Aurora MySQL
- Amazon Aurora MySQL dan data spasial
- Aurora SQL Versi saya 3 kompatibel dengan My 8.0 SQL
- Aurora SQL Versi saya 2 kompatibel dengan My 5.7 SQL


## Peningkatan performa Amazon Aurora MySQL

Amazon Aurora menyediakan peningkatan performa untuk mendukung beragam kebutuhan basis data komersial kelas atas.

### Penyisipan cepat

Penyisipan cepat mempercepat penyisipan paralel yang diurutkan berdasarkan kunci primer dan berlaku khusus untuk pernyataan `LOAD DATA` dan `INSERT INTO ... SELECT ...`. Sisipkan cache dengan cepat ke posisi kursor dalam sebuah traversal indeks sambil mengeksekusi pernyataan. Hal ini membantu agar indeks tidak perlu di-traversing lagi.

Penyisipan cepat diaktifkan hanya untuk tabel InnoDB biasa di Aurora MySQL versi 3.03.2 dan lebih tinggi. Pengoptimalan ini tidak berfungsi untuk tabel sementara InnoDB. Ini dinonaktifkan di Aurora MySQL versi 2 untuk semua versi 2.11 dan 2.12. Pengoptimalan sisipan cepat hanya berfungsi jika optimasi Indeks Hash Adaptif dinonaktifkan.

Anda dapat memantau metrik berikut untuk menentukan efektivitas penyisipan cepat untuk klaster DB Anda:

- `aurora_fast_insert_cache_hits`: Penghitung yang bertambah saat kursor yang di-cache berhasil diambil dan diverifikasi.
- `aurora_fast_insert_cache_misses`: Penghitung yang bertambah ketika kursor yang di-cache tidak lagi valid dan Aurora melakukan traversal indeks normal.

Anda dapat mengambil nilai saat ini dari metrik penyisipan cepat dengan menggunakan perintah berikut:

```
mysql> show global status like 'Aurora_fast_insert%';
```

Anda akan mendapatkan output seperti yang berikut ini:

```
+--------------------------------+-----------+
| Variable_name                  | Value     |
+--------------------------------+-----------+
| Aurora_fast_insert_cache_hits  | 3598300   |
| Aurora_fast_insert_cache_misses | 436401336 |
+--------------------------------+-----------+
```

## Amazon Aurora MySQL dan data spasial

Daftar berikut merangkum fitur spasial Aurora MySQL utama dan menjelaskan bagaimana fitur tersebut terkait dengan fitur spasial di MySQL:

- Aurora MySQL versi 2 mendukung jenis data spasial dan fungsi relasi spasial yang sama dengan MySQL 5.7. Untuk informasi selengkapnya tentang jenis dan fungsi data ini, lihat Spatial Data Types dan Spatial Relation Functions dalam dokumentasi MySQL 5.7.
- Aurora MySQL versi 3 mendukung jenis data spasial dan fungsi relasi spasial yang sama dengan MySQL 8.0. Untuk informasi selengkapnya tentang jenis dan fungsi data ini, lihat Spatial Data Types dan Spatial Relation Functions dalam dokumentasi MySQL 8.0.
- Aurora MySQL mendukung pengindeksan spasial pada tabel InnoDB. Pengindeksan spasial meningkatkan performa kueri pada set data besar untuk kueri pada data spasial. Di MySQL, pengindeksan spasial untuk tabel InnoDB tersedia di MySQL 5.7 dan 8.0.

  Aurora MySQL menggunakan strategi pengindeksan spasial yang berbeda dari MySQL untuk performa tinggi dengan kueri spasial. Implementasi indeks spasial Aurora menggunakan kurva pengisian ruang pada B-tree, yang ditujukan untuk memberikan performa yang lebih tinggi untuk pemindaian rentang spasial daripada R-tree.

  > **Note**
  >
  > Di Aurora MySQL, transaksi pada tabel dengan indeks spasial yang ditentukan pada kolom dengan pengidentifikasi referensi spasial (SRID) tidak dapat melakukan penyisipan ke area yang dipilih untuk diperbarui oleh transaksi lain.

Pernyataan bahasa definisi data (DDL) berikut didukung untuk membuat indeks pada kolom yang menggunakan jenis data spasial.

### CREATE TABLE

Anda dapat menggunakan kata kunci `SPATIAL INDEX` dalam pernyataan `CREATE TABLE` untuk menambahkan indeks spasial ke kolom di tabel baru. Berikut adalah contohnya.

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

### ALTER TABLE

Anda dapat menggunakan kata kunci `SPATIAL INDEX` dalam pernyataan `ALTER TABLE` untuk menambahkan indeks spasial ke kolom dalam tabel yang ada. Berikut adalah contohnya.

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

### CREATE INDEX

Anda dapat menggunakan kata kunci `SPATIAL` dalam pernyataan `CREATE INDEX` untuk menambahkan indeks spasial ke kolom dalam tabel yang ada. Berikut adalah contohnya.

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

## Aurora SQL Versi saya 3 kompatibel dengan My 8.0 SQL

Anda dapat menggunakan Aurora My SQL versi 3 untuk mendapatkan fitur terbaru yang SQL kompatibel dengan Saya, peningkatan kinerja, dan perbaikan bug. Berikut ini, Anda dapat mempelajari tentang Aurora My SQL versi 3, dengan kompatibilitas My SQL 8.0. Anda dapat mempelajari cara meningkatkan cluster dan aplikasi Anda ke Aurora SQL My versi 3.

Beberapa fitur Aurora, seperti Aurora Serverless v2, membutuhkan Aurora SQL Versi saya 3.

Topik

- Fitur dari My SQL 8.0 Community Edition
- Aurora Prasyarat SQL versi 3 saya untuk Aurora My Serverless v2 SQL
- Catatan rilis untuk Aurora My versi 3 SQL

- Optimisasi kueri paralel baru
- Optimisasi untuk mengurangi waktu pengaktifan ulang basis data.
- Perilaku tabel sementara baru di Aurora MySQL versi 3
- Membandingkan Aurora SQL Versi saya 2 dan Aurora Versi saya 3 SQL
- Membandingkan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition
- Meningkatkan ke Aurora MySQL versi 3

## Fitur dari My SQL 8.0 Community Edition

Rilis awal Aurora My SQL versi 3 kompatibel dengan My SQL 8.0.23 Community Edition. SQL8.0 saya memperkenalkan beberapa fitur baru, termasuk yang berikut ini:

- Dukungan Atomic Data Definition Language (DDL). Untuk informasi selengkapnya, lihat Dukungan Atomic Data Definition Language (DDL).
- JSONfungsi. Untuk informasi penggunaan, lihat JSONFungsi di Manual SQL Referensi Saya.
- Fungsi Jendela. Untuk informasi penggunaan, lihat Fungsi Jendela di Manual SQL Referensi Saya.
- Ekspresi tabel umum (CTEs), menggunakan `WITH` klausa. Untuk informasi penggunaan, lihat WITH(Ekspresi Tabel Umum) di Manual SQL Referensi Saya.
- Klausa `ADD COLUMN` dan `RENAME COLUMN` yang dioptimalkan untuk pernyataan `ALTER TABLE`. Optimalisasi ini disebut “instanDDL.” Aurora SQL Versi saya 3 kompatibel dengan komunitas Fitur SQL instan DDL saya. DDLFitur cepat Aurora sebelumnya tidak digunakan. Untuk informasi penggunaan instanDDL, lihatDDL instan (Aurora MySQL versi 3).
- Indeks menurun, fungsional, dan tidak terlihat. Untuk informasi penggunaan, lihat Indeks Tak Terlihat, Indeks Turun, dan CREATEINDEXPernyataan di Manual Referensi Saya SQL.
- Hak istimewa berbasis peran dikendalikan melalui pernyataan. SQL Untuk informasi selengkapnya tentang perubahan pada model hak akses, lihat Model hak akses berbasis peran.
- Klausa `NOWAIT` dan `SKIP LOCKED` dengan pernyataan `SELECT ... FOR SHARE`. Klausa ini menghindari tindakan menunggu transaksi lain untuk membuka kunci baris. Untuk informasi penggunaan, lihat Mengunci Bacaan di Manual SQL Referensi Saya.
- Peningkatan pada replikasi log biner (binlog). Untuk SQL detail Aurora My, lihat. Replikasi log biner Khususnya, Anda dapat melakukan replikasi yang difilter. Untuk informasi penggunaan tentang replikasi yang difilter, lihat Cara Server Mengevaluasi Aturan Pemfilteran Replikasi di Manual Referensi Saya SQL.

- Petunjuk. Beberapa petunjuk yang SQL kompatibel dengan My 8.0 sudah di-backport ke Aurora My versi 2. SQL Untuk informasi tentang menggunakan petunjuk dengan Aurora SQL My, lihat. Aurora Petunjuk saya SQL Untuk daftar lengkap petunjuk di komunitas My SQL 8.0, lihat Petunjuk Pengoptimal di Manual Referensi Saya. SQL

Untuk daftar lengkap fitur yang ditambahkan ke My SQL 8.0 edisi komunitas, lihat posting blog Daftar lengkap fitur baru di My SQL 8.0.

Aurora SQL Versi saya 3 juga mencakup perubahan kata kunci untuk bahasa inklusif, di-backport dari komunitas My 8.0.26. SQL Untuk detail tentang perubahan tersebut, lihat Perubahan bahasa inklusif untuk Aurora Versi saya 3 SQL.

## Aurora Prasyarat SQL versi 3 saya untuk Aurora My Serverless v2 SQL

Aurora My SQL versi 3 adalah prasyarat untuk semua instans DB di cluster Aurora My Serverless v2. SQL Aurora My SQL Serverless v2 menyertakan dukungan untuk instance pembaca di cluster DB, dan fitur Aurora lainnya yang tidak tersedia untuk Aurora My Serverless v1. SQL Ini juga memiliki penskalaan yang lebih cepat dan lebih granular daripada Aurora My Serverless v1. SQL

## Catatan rilis untuk Aurora My versi 3 SQL

Untuk catatan rilis untuk semua rilis Aurora My SQL version 3, lihat Pembaruan mesin database untuk Amazon Aurora SQL My version 3 di Catatan Rilis untuk Aurora My. SQL

## Optimisasi kueri paralel baru

Optimasi kueri paralel Aurora sekarang berlaku untuk lebih banyak SQL operasi:

- Kueri pararel sekarang berlaku untuk tabel yang berisi jenis data `TEXT`, `BLOB`, `JSON`, `GEOMETRY`, serta `VARCHAR` dan `CHAR` yang lebih panjang dari 768 byte.
- Kueri paralel dapat mengoptimalkan kueri yang memerlukan tabel yang dipartisi.
- Kueri paralel dapat mengoptimalkan kueri yang memerlukan panggilan fungsi agregat dalam daftar pilih dan klausa `HAVING`.

Untuk informasi selengkapnya tentang peningkatan ini, lihat Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL. Untuk informasi umum tentang kueri paralel Aurora, lihat Kueri paralel untuk Amazon Aurora My SQL.

## Optimisasi untuk mengurangi waktu pengaktifan ulang basis data.

Cluster Aurora My SQL DB Anda harus sangat tersedia selama pemadaman yang direncanakan dan tidak direncanakan.

Administrator basis data perlu melakukan pemeliharaan basis data sesekali. Pemeliharaan ini mencakup patching basis data, tingkatkan, modifikasi parameter basis data yang memerlukan boot ulang manual, pelaksanaan failover untuk mengurangi waktu yang diperlukan dalam mengubah kelas instans, dan sebagainya. Tindakan yang direncanakan ini membutuhkan waktu henti.

Namun, waktu henti juga dapat disebabkan oleh tindakan yang tidak direncanakan, seperti failover yang tidak terduga karena kesalahan perangkat keras yang mendasarinya atau throttling sumber daya basis data. Semua tindakan yang direncanakan dan tidak direncanakan ini mengakibatkan pengaktifan ulang basis data.

Di Aurora My SQL versi 3.05 dan yang lebih tinggi, kami telah memperkenalkan pengoptimalan yang mengurangi waktu restart database. Optimisasi ini memberikan waktu henti hingga 65% lebih sedikit daripada tanpa optimisasi, dan lebih sedikit gangguan pada beban kerja basis data Anda, setelah pengaktifan ulang.

Selama pengaktifan basis data, banyak komponen memori internal yang diinisialisasi. Yang terbesar adalah kumpulan buffer InnoDB, yang di Aurora My SQL adalah 75% dari ukuran memori instance secara default. Pengujian kami telah menemukan bahwa waktu inisialisasinya sebanding dengan ukuran pool buffer InnoDB, dan oleh karena itu, akan diskalakan dengan ukuran kelas instans DB. Selama tahap inisialisasi ini, basis data tidak dapat menerima koneksi, sehingga memperpanjang waktu henti selama pengaktifan ulang. Fase pertama Aurora My SQL fast restart mengoptimalkan inisialisasi kumpulan buffer, yang mengurangi waktu untuk inisialisasi database dan dengan demikian mengurangi waktu restart secara keseluruhan.

Untuk detail selengkapnya, lihat blog Kurangi waktu henti dengan Amazon Aurora Database SQL saya memulai ulang pengoptimalan waktu.

## Perilaku tabel sementara baru di Aurora MySQL versi 3

Aurora MySQL versi 3 menangani tabel sementara secara berbeda dari Aurora MySQL versi sebelumnya. Perilaku baru ini diwarisi dari MySQL 8.0 Community Edition. Berikut dua jenis tabel sementara yang dapat dibuat dengan Aurora MySQL versi 3:

- Tabel sementara internal (atau implisit) — Dibuat oleh mesin MySQL Aurora untuk menangani operasi seperti pengurutan agregasi, tabel turunan, atau ekspresi tabel umum (). CTEs

- Tabel sementara yang dibuat pengguna (atau eksplisit) — Dibuat oleh mesin Aurora MySQL saat Anda menggunakan pernyataan `CREATE TEMPORARY TABLE`.

Terdapat pertimbangan tambahan untuk tabel sementara internal dan yang dibuat pengguna pada instans DB pembaca Aurora. Kita akan membahas perubahan ini di bagian berikut.

Topik

- Mesin penyimpanan untuk tabel sementara internal (implisit)
- Membatasi ukuran tabel sementara internal dalam memori
- Mengurangi masalah kepenuhan untuk tabel sementara internal di Aurora Replicas
- Mengoptimalkan parameter temptable_max_mmap pada instance Aurora MySQL DB
- Tabel sementara yang dibuat pengguna (eksplisit) pada instans DB pembaca
- Kesalahan dan mitigasi pembuatan tabel sementara

Mesin penyimpanan untuk tabel sementara internal (implisit)

Saat membuat set hasil perantara, Aurora MySQL awalnya mencoba menulis ke tabel sementara dalam memori. Langkah ini mungkin tidak berhasil karena tipe data yang tidak kompatibel atau batas yang dikonfigurasi. Jika demikian, tabel sementara dikonversi ke tabel sementara di disk, bukan disimpan di memori. Informasi selengkapnya tentang hal ini dapat ditemukan di Internal Temporary Table Use in MySQL dalam dokumentasi MySQL.

Di Aurora MySQL versi 3, cara kerja tabel sementara internal berbeda dengan Aurora MySQL versi sebelumnya. Alih-alih memilih antara mesin penyimpanan InnoDB dan MyISAM untuk tabel sementara seperti itu, sekarang Anda memilih antara mesin penyimpanan dan mesin penyimpanan. `TempTable` `MEMORY`

Dengan mesin penyimpanan `TempTable`, Anda dapat membuat pilihan tambahan untuk cara menangani data tertentu. Data yang terpengaruh memenuhi kumpulan memori yang menampung semua tabel sementara internal untuk instans DB.

Pilihan tersebut dapat memengaruhi performa kueri yang menghasilkan volume data sementara yang tinggi, misalnya saat melakukan agregasi seperti `GROUP BY` pada tabel besar.

> **Tip**
>
> Jika beban kerja Anda mencakup kueri yang menghasilkan tabel sementara internal, konfirmasikan performa aplikasi Anda dengan perubahan ini melalui upaya menjalankan tolok ukur dan memantau metrik terkait performa.
> Dalam kasus tertentu, jumlah data sementara sesuai dengan kumpulan memori `TempTable` atau hanya memenuhi kumpulan memori dalam jumlah kecil. Jika demikian, sebaiknya gunakan pengaturan `TempTable` untuk tabel sementara internal dan file yang dipetakan memori untuk menyimpan data overflow apa pun. Ini adalah pengaturan default.

Mesin penyimpanan `TempTable` adalah pengaturan default. `TempTable` menggunakan kumpulan memori umum untuk semua tabel sementara yang menggunakan mesin ini, bukan batas memori maksimum per tabel. Ukuran kumpulan memori ini ditentukan oleh parameter temptable_max_ram. Nilai default adalah 1 GiB pada instans DB dengan memori 16 GiB atau lebih, dan 16 MB pada instans DB dengan memori kurang dari 16 GiB. Ukuran kumpulan memori memengaruhi konsumsi memori tingkat sesi.

Dalam kasus tertentu, bila Anda menggunakan mesin penyimpanan `TempTable`, data sementara mungkin melebihi ukuran kumpulan memori. Jika demikian, Aurora MySQL akan menyimpan data overflow tersebut menggunakan mekanisme sekunder.

Anda dapat mengatur parameter temptable_max_mmap untuk memilih apakah data akan memenuhi file sementara yang dipetakan memori atau tabel sementara internal InnoDB di disk. Format data yang berbeda dan kriteria overflow dari mekanisme overflow ini dapat memengaruhi performa kueri. Hal ini dilakukan dengan memengaruhi jumlah data yang ditulis ke disk dan permintaan atas throughput penyimpanan disk.

Aurora MySQL versi 3 menyimpan data overflow dengan cara berikut:

- Pada instance DB penulis, data yang meluap ke tabel sementara internal InnoDB atau file sementara yang dipetakan memori berada di penyimpanan lokal pada instance.
- Pada instans DB pembaca, data overflow selalu berada dalam file sementara yang dipetakan memori di penyimpanan lokal.

  Instans hanya-baca tidak dapat menyimpan data apa pun pada volume cluster Aurora.

Parameter konfigurasi terkait tabel sementara internal berlaku untuk instans penulis dan pembaca dengan cara yang berbeda di klaster Anda:

- Pada instans pembaca, Aurora MySQL selalu menggunakan mesin penyimpanan `TempTable`.
- Ukuran default `temptable_max_mmap` adalah 1 GiB untuk instans penulis dan pembaca, berapa pun ukuran memori instans DB-nya. Anda dapat menyesuaikan nilai ini pada instans penulis dan pembaca.
- Mengatur `temptable_max_mmap` ke `0` akan menonaktifkan penggunaan file sementara yang dipetakan memori pada instans penulis.
- Anda tidak dapat mengatur `temptable_max_mmap` ke `0` pada instans pembaca.

> **Note**
>
> Kami tidak menyarankan penggunaan parameter temptable_use_mmap. Parameter tersebut telah dihentikan, dan dukungannya direncanakan akan dihapus dalam rilis MySQL mendatang.

Membatasi ukuran tabel sementara internal dalam memori

Sebagaimana dibahas dalam Mesin penyimpanan untuk tabel sementara internal (implisit), Anda dapat mengontrol sumber daya tabel sementara secara global dengan menggunakan pengaturan temptable_max_ram dan temptable_max_mmap.

Anda juga dapat membatasi ukuran setiap tabel sementara internal dalam memori menggunakan parameter DB tmp_table_size. Batas ini dimaksudkan untuk mencegah setiap kueri mengonsumsi sumber daya tabel sementara global dalam jumlah besar, yang dapat memengaruhi performa kueri bersamaan yang memerlukan sumber daya ini.

Parameter `tmp_table_size` menetapkan ukuran maksimum tabel sementara yang dibuat oleh mesin penyimpanan `MEMORY` di Aurora MySQL versi 3.

Di Aurora MySQL versi 3.04 dan yang lebih tinggi, `tmp_table_size` juga menetapkan ukuran maksimum tabel sementara yang dibuat oleh mesin penyimpanan `TempTable` saat parameter DB `aurora_tmptable_enable_per_table_limit` diatur ke `ON`. Perilaku ini dinonaktifkan secara default (`OFF`), yang merupakan perilaku serupa seperti di Aurora MySQL versi 3.03 dan yang lebih rendah.

- Bila `aurora_tmptable_enable_per_table_limit` diatur ke `OFF`, `tmp_table_size` tidak dipertimbangkan untuk tabel sementara internal dalam memori yang dibuat oleh mesin penyimpanan `TempTable`.

  Namun, batas sumber daya `TempTable` global masih berlaku. Bila batas sumber daya `TempTable` global tercapai, Aurora MySQL memiliki perilaku berikut:

  - Instans DB penulis – Aurora MySQL secara otomatis mengonversi tabel sementara dalam memori menjadi tabel sementara di disk InnoDB.
  - Instans DB pembaca – Kueri berakhir dengan kesalahan.

  ```
  ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
  ```

- Bila `aurora_tmptable_enable_per_table_limit` diatur ke `ON`, Aurora MySQL memiliki perilaku berikut saat batas `tmp_table_size` tercapai:

  - Instans DB penulis – Aurora MySQL secara otomatis mengonversi tabel sementara dalam memori menjadi tabel sementara di disk InnoDB.
  - Instans DB pembaca – Kueri berakhir dengan kesalahan.

  ```
  ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
  ```

  Dalam kasus ini, batas sumber daya `TempTable` global dan batas per tabel berlaku.

> **Note**
>
> Parameter `aurora_tmptable_enable_per_table_limit` tidak berpengaruh saat internal_tmp_mem_storage_engine diatur ke `MEMORY`. Dalam hal ini, ukuran maksimum tabel sementara dalam memori ditentukan oleh nilai tmp_table_size atau max_heap_table_size, mana pun yang lebih kecil.

Contoh berikut menunjukkan perilaku parameter `aurora_tmptable_enable_per_table_limit` untuk instans DB penulis dan pembaca.

Example instans DB penulis dengan **`aurora_tmptable_enable_per_table_limit`** diatur ke **`OFF`**

Tabel sementara dalam memori tidak dikonversi ke tabel sementara di disk InnoDB.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+-------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+-------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
|                   0 | 3.04.0          |                                        0 |
       1073741824 |          1073741824 |
+-------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
```

```
1 row in set (0.00 sec)
```

Example instans DB penulis dengan **aurora_tmptable_enable_per_table_limit** diatur ke **ON**

Tabel sementara dalam memori dikonversi ke tabel sementara di disk InnoDB.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+-----------------------------------------
+------------------+
|                  0 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Created_tmp_disk_tables | 0     |
+--------------------------+-------+
1 row in set (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)

mysql> show status like '%created_tmp_disk%';
```

```
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example instans DB pembaca dengan **`aurora_tmptable_enable_per_table_limit`** diatur ke **`OFF`**

Kueri selesai tanpa kesalahan karena `tmp_table_size` tidak berlaku dan batas sumber daya `TempTable` global belum tercapai.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0          |                                        0 |
       1073741824 |          1073741824 |
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example instans DB pembaca dengan **`aurora_tmptable_enable_per_table_limit`** diatur ke **`OFF`**

Kueri ini mencapai batas TempTable sumber daya global dengan `aurora_tmptable_enable_per_table_limit` disetel ke OFF. Kueri berakhir dengan kesalahan pada instans pembaca.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                      0 |
       1073741824 |           1073741824 |
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example instans DB pembaca dengan **`aurora_tmptable_enable_per_table_limit`** diatur ke **`ON`**

Kueri berakhir dengan kesalahan saat batas `tmp_table_size` tercapai.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+----------------------------------------
+------------------+
```

```
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+------------------------------------------
+------------------+
|                  1 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+------------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

Mengurangi masalah kepenuhan untuk tabel sementara internal di Aurora Replicas

Untuk mencegah masalah pembatasan ukuran pada tabel sementara, atur parameter `temptable_max_ram` dan `temptable_max_mmap` ke nilai gabungan yang dapat memenuhi persyaratan beban kerja Anda.

Berhati-hatilah saat mengatur nilai parameter `temptable_max_ram`. Mengatur nilai terlalu tinggi mengurangi memori yang tersedia pada instance database, yang dapat menyebabkan suatu out-of-memory kondisi. Pantau memori rata-rata yang dapat dikosongkan pada instans DB. Lalu, tentukan nilai yang sesuai untuk `temptable_max_ram`, sehingga Anda akan tetap memiliki memori bebas dalam jumlah wajar yang tersisa pada instans. Untuk informasi selengkapnya, lihat Masalah memori yang dapat dikosongkan di Amazon Aurora.

Penting juga bagi Anda untuk memantau ukuran penyimpanan lokal dan konsumsi ruang tabel sementara. Anda dapat memantau penyimpanan sementara yang tersedia untuk instans DB tertentu dengan CloudWatch metrik `FreeLocalStorage` Amazon, yang dijelaskan dalam CloudWatch Metrik Amazon untuk Amazon Aurora.

> **Note**
>
> Prosedur ini tidak berfungsi bila parameter `aurora_tmptable_enable_per_table_limit` diatur ke `ON`. Untuk informasi selengkapnya, lihat Membatasi ukuran tabel sementara internal dalam memori.

Example 1

Anda tahu bahwa tabel sementara Anda berkembang menjadi berukuran kumulatif 20 GiB. Anda ingin mengatur tabel sementara dalam memori ke 2 GiB dan mengembangkannya menjadi maksimum 20 GiB pada disk.

Atur `temptable_max_ram` ke **`2,147,483,648`** dan `temptable_max_mmap` ke **`21,474,836,480`**. Nilai ini dihitung dalam byte.

Pengaturan parameter ini memastikan bahwa tabel sementara Anda dapat berkembang menjadi total kumulatif 22 GiB.

Example 2

Ukuran instans Anda saat ini adalah 16xlarge atau lebih besar. Anda tidak mengetahui ukuran total tabel sementara yang mungkin Anda perlukan. Anda ingin dapat menggunakan hingga 4 GiB dalam memori dan hingga ukuran penyimpanan maksimum yang tersedia pada disk.

Atur `temptable_max_ram` ke **`4,294,967,296`** dan `temptable_max_mmap` ke **`1,099,511,627,776`**. Nilai ini dihitung dalam byte.

Di sini, Anda mengatur `temptable_max_mmap` ke 1 TiB, yang lebih kecil dari penyimpanan lokal maksimum sebesar 1,2 TiB pada instans DB Aurora 16xlarge.

Pada ukuran instans yang lebih kecil, sesuaikan nilai `temptable_max_mmap` agar tidak mengisi penyimpanan lokal yang tersedia. Misalnya, instans 2xlarge hanya memiliki penyimpanan lokal yang tersedia sebesar 160 GiB. Karena itu, sebaiknya atur nilainya menjadi kurang dari 160 GiB. Untuk informasi selengkapnya tentang penyimpanan lokal yang tersedia untuk ukuran instans DB, lihat Batas penyimpanan sementara untuk Aurora MySQL.

Mengoptimalkan parameter temptable_max_mmap pada instance Aurora MySQL DB

`temptable_max_mmap`Parameter di Aurora MySQL mengontrol jumlah maksimum ruang disk lokal yang dapat digunakan oleh file yang dipetakan memori sebelum meluap ke tabel sementara InnoDB on-disk (pada instance DB penulis) atau menyebabkan kesalahan (pada instance DB pembaca). Menyetel parameter instans DB ini dengan benar dapat membantu mengoptimalkan kinerja instans DB Anda.

Prasyarat

1. Pastikan bahwa Skema Kinerja diaktifkan. Anda dapat memverifikasi ini dengan menjalankan perintah SQL berikut:

```
SELECT @@performance_schema;
```

Nilai output 1 menunjukkan bahwa itu diaktifkan.

2. Konfirmasikan bahwa instrumentasi memori tabel sementara diaktifkan. Anda dapat memverifikasi ini dengan menjalankan perintah SQL berikut:

```
SELECT name, enabled FROM performance_schema.setup_instruments WHERE name LIKE
 '%memory%temptable%';
```

`enabled`Kolom menunjukkan `YES` entri instrumentasi memori tabel sementara yang relevan.

Memantau penggunaan tabel sementara

Saat menyetel nilai awal`temptable_max_mmap`, kami sarankan Anda memulai dengan 80% ukuran penyimpanan lokal untuk kelas instans DB yang Anda gunakan. Ini memastikan bahwa tabel sementara memiliki ruang disk yang cukup untuk beroperasi secara efisien, sambil menyisakan ruang untuk penggunaan disk lain pada instance.

Untuk menemukan ukuran penyimpanan lokal untuk kelas instans DB Anda, lihatBatas penyimpanan sementara untuk Aurora MySQL.

Misalnya, jika Anda menggunakan kelas instans db.r5.large DB, ukuran penyimpanan lokal adalah 32 GiB. Dalam hal ini, Anda awalnya akan mengatur `temptable_max_mmap` parameter ke 80% dari 32 GiB, yaitu 25,6 GiB.

Setelah mengatur `temptable_max_mmap` nilai awal, jalankan beban kerja puncak Anda pada instance Aurora MySQL. Pantau penggunaan disk tabel sementara saat ini dan tinggi menggunakan kueri SQL berikut:

```
SELECT event_name, current_count, current_alloc, current_avg_alloc, high_count,
 high_alloc, high_avg_alloc
FROM sys.memory_global_by_current_bytes WHERE event_name LIKE 'memory/temptable/%';
```

Kueri ini mengambil informasi berikut:

- `event_name`— Nama memori tabel sementara atau acara penggunaan disk.
- `current_count`— Jumlah memori tabel sementara atau blok disk yang dialokasikan saat ini.
- `current_alloc`— Jumlah memori atau disk saat ini yang dialokasikan untuk tabel sementara.
- `current_avg_alloc`— Ukuran rata-rata saat ini dari memori tabel sementara atau blok disk.

- `high_count`— Jumlah tertinggi memori tabel sementara atau blok disk yang dialokasikan.
- `high_alloc`— Jumlah memori atau disk tertinggi yang dialokasikan untuk tabel sementara.
- `high_avg_alloc`— Ukuran rata-rata tertinggi memori tabel sementara atau blok disk.

Jika kueri Anda gagal dengan kesalahan penuh Tabel menggunakan pengaturan ini, ini menunjukkan bahwa beban kerja Anda memerlukan lebih banyak ruang disk untuk operasi tabel sementara. Dalam hal ini, pertimbangkan untuk meningkatkan ukuran instans DB Anda menjadi satu dengan lebih banyak ruang penyimpanan lokal.

Mengatur **`temptable_max_mmap`** nilai optimal

Gunakan prosedur berikut untuk memantau dan mengatur ukuran yang tepat untuk `temptable_max_mmap` parameter.

1. Tinjau output dari kueri sebelumnya, dan identifikasi penggunaan disk tabel sementara puncak, seperti yang ditunjukkan oleh `high_alloc` kolom.
2. Berdasarkan penggunaan disk tabel sementara puncak, sesuaikan `temptable_max_mmap` parameter dalam grup parameter DB untuk instance Aurora MySQL DB Anda.

   Tetapkan nilainya menjadi sedikit lebih tinggi dari penggunaan disk tabel sementara puncak untuk mengakomodasi pertumbuhan masa depan.
3. Terapkan perubahan grup parameter ke instans DB Anda.
4. Pantau penggunaan disk tabel sementara lagi selama beban kerja puncak Anda untuk memastikan bahwa `temptable_max_mmap` nilai baru sesuai.
5. Ulangi langkah sebelumnya sesuai kebutuhan untuk menyempurnakan `temptable_max_mmap` parameter.

Tabel sementara yang dibuat pengguna (eksplisit) pada instans DB pembaca

Anda dapat membuat tabel sementara eksplisit menggunakan kata kunci `TEMPORARY` dalam pernyataan `CREATE TABLE`. Tabel sementara eksplisit didukung pada instans DB penulis di klaster DB Aurora. Anda juga dapat menggunakan tabel sementara eksplisit pada instans DB pembaca, tetapi tabel tidak dapat menerapkan penggunaan mesin penyimpanan InnoDB.

Untuk mencegah timbulnya kesalahan saat membuat tabel sementara eksplisit pada instans DB pembaca Aurora MySQL, pastikan Anda menjalankan semua pernyataan `CREATE TEMPORARY TABLE` dengan salah satu atau kedua cara berikut:

- Jangan tentukan klausa `ENGINE=InnoDB`.

- Jangan atur mode SQL ke NO_ENGINE_SUBSTITUTION.

Kesalahan dan mitigasi pembuatan tabel sementara

Kesalahan yang Anda terima berbeda tergantung pada apakah Anda menggunakan pernyataan CREATE TEMPORARY TABLE biasa atau variasi CREATE TEMPORARY TABLE AS SELECT. Contoh berikut menunjukkan berbagai jenis kesalahan.

Perilaku tabel sementara ini hanya berlaku untuk instans hanya-baca. Contoh pertama ini mengonfirmasi jenis instans yang terhubung dengan sesi.

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

Untuk pernyataan CREATE TEMPORARY TABLE biasa, pernyataan akan gagal saat mode SQL NO_ENGINE_SUBSTITUTION diaktifkan. Ketika NO_ENGINE_SUBSTITUTION dinonaktifkan (default), substitusi mesin yang sesuai dibuat, dan pembuatan tabel sementara berhasil.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Untuk pernyataan CREATE TEMPORARY TABLE AS SELECT, pernyataan akan gagal saat mode SQL NO_ENGINE_SUBSTITUTION diaktifkan. Ketika NO_ENGINE_SUBSTITUTION dinonaktifkan (default), substitusi mesin yang sesuai dibuat, dan pembuatan tabel sementara berhasil.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
+-------+-----------------------------------------------------------------+
| Table | Create Table                                                    |
+-------+-----------------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Untuk informasi selengkapnya tentang aspek penyimpanan dan implikasi kinerja tabel sementara di Aurora MySQL versi 3, lihat posting blog TempTable Gunakan mesin penyimpanan di Amazon RDS for MySQL dan Amazon Aurora MySQL.

## Membandingkan Aurora SQL Versi saya 2 dan Aurora Versi saya 3 SQL

Gunakan yang berikut ini untuk mempelajari tentang perubahan yang harus diperhatikan saat Anda memutakhirkan klaster Aurora My SQL version 2 Anda ke versi 3.

Topik

- Dukungan Atomic Data Definition Language (DDL)
- Perbedaan fitur antara Aurora My SQL versi 2 dan 3
- Dukungan kelas instans
- Perubahan parameter untuk Aurora Versi saya 3 SQL
- Variabel status
- Perubahan bahasa inklusif untuk Aurora Versi saya 3 SQL
- AUTO_ INCREMENT nilai
- Replikasi log biner

Dukungan Atomic Data Definition Language (DDL)

Salah satu perubahan terbesar dari My SQL 5.7 ke 8.0 adalah pengenalan Atomic Data Dictionary. Sebelum My SQL 8.0, kamus SQL data Saya menggunakan pendekatan berbasis file untuk menyimpan metadata seperti definisi tabel (.frm), pemicu (.trg), dan berfungsi secara terpisah dari metadata mesin penyimpanan (seperti InnoDB's). Ini memiliki beberapa masalah, termasuk risiko tabel menjadi "yatim piatu" jika sesuatu yang tidak terduga terjadi selama DDL operasi, menyebabkan metadata berbasis file dan mesin penyimpanan tidak sinkron.

Untuk memperbaikinya, My SQL 8.0 memperkenalkan Atomic Data Dictionary, yang menyimpan semua metadata dalam satu set tabel InnoDB internal dalam skema. `mysql` Arsitektur baru ini menyediakan cara transaksional ACIDyang sesuai untuk mengelola metadata database, memecahkan masalah “atomDDL” dari pendekatan berbasis file lama. Untuk informasi selengkapnya tentang Kamus Data Atom, lihat Penghapusan penyimpanan metadata berbasis file dan dukungan pernyataan definisi data atom di Manual Referensi Saya. SQL

Karena perubahan arsitektur ini, Anda harus mempertimbangkan hal berikut saat memutakhirkan dari Aurora SQL My versi 2 ke versi 3:

- Metadata berbasis file dari versi 2 harus dimigrasikan ke tabel kamus data baru selama proses upgrade ke versi 3. Bergantung pada berapa banyak objek database yang dimigrasikan, ini bisa memakan waktu.
- Perubahan juga telah memperkenalkan beberapa ketidakcocokan baru yang mungkin perlu ditangani sebelum Anda dapat meningkatkan dari My SQL 5.7 ke 8.0. Misalnya, 8.0 memiliki beberapa kata kunci cadangan baru yang dapat bertentangan dengan nama objek database yang ada.

Untuk membantu Anda mengidentifikasi ketidakcocokan ini sebelum memutakhirkan mesin, Aurora My SQL menjalankan serangkaian pemeriksaan kompatibilitas pemutakhiran (prechecks) untuk menentukan apakah ada objek yang tidak kompatibel dalam kamus database Anda, sebelum melakukan pemutakhiran kamus data. Untuk informasi lebih lanjut tentang prechecks, lihat Pra-cek pemutakhiran versi utama untuk Aurora My SQL.

Perbedaan fitur antara Aurora My SQL versi 2 dan 3

SQLFitur Amazon Aurora My berikut didukung di Aurora My SQL for My SQL 5.7, tetapi fitur ini tidak didukung di Aurora My untuk My 8.0: SQL SQL

- Anda tidak dapat menggunakan Aurora My SQL versi 3 untuk Aurora Serverless kluster v1. Aurora SQL Versi saya 3 berfungsi dengan Aurora Serverless v2.
- Mode lab tidak berlaku untuk Aurora My SQL versi 3. Tidak ada fitur mode lab di Aurora My SQL versi 3. Instan DDL menggantikan DDL fitur online cepat yang sebelumnya tersedia dalam mode lab. Sebagai contoh, lihat DDL instan (Aurora MySQL versi 3).
- Cache kueri dihapus dari komunitas My SQL 8.0 dan juga dari Aurora SQL My versi 3.
- Aurora SQL Versi saya 3 kompatibel dengan komunitas Fitur bergabung dengan SQL hash saya. Implementasi khusus Aurora dari gabungan hash di Aurora My version 2 tidak digunakan. SQL Untuk informasi tentang menggunakan hash join dengan kueri paralel Aurora, lihat Mengaktifkan hash join untuk klaster kueri paralel dan Aurora Petunjuk saya SQL. Untuk informasi penggunaan umum tentang gabungan hash, lihat Pengoptimalan Gabung Hash di Manual Referensi Saya SQL.
- Prosedur `mysql.lambda_async` tersimpan yang tidak digunakan lagi di Aurora My SQL version 2 dihapus di versi 3. Untuk versi 3, gunakan fungsi asinkron `lambda_async` sebagai gantinya.
- Karakter default yang ditetapkan di Aurora My SQL versi 3 adalah. `utf8mb4` Di Aurora My SQL versi 2, set karakter default adalah. `latin1` Untuk informasi tentang set karakter ini, lihat Set Karakter utf8mb4 (4-Byte UTF -8 Unicode Encoding) di Manual Referensi Saya. SQL

Beberapa SQL fitur Aurora My tersedia untuk kombinasi tertentu dari versi mesin AWS Region dan DB. Untuk detailnya, lihat Fitur yang didukung di Amazon Aurora oleh dan mesin Wilayah AWS Aurora DB.

Dukungan kelas instans

Aurora SQL Versi saya 3 mendukung serangkaian kelas instance yang berbeda dari Aurora My versi 2: SQL

- Untuk instans yang lebih besar, Anda dapat menggunakan kelas instans modern seperti `db.r5`, `db.r6g`, dan `db.x2g`.
- Untuk instans yang lebih kecil, Anda dapat menggunakan kelas instans modern seperti `db.t3` dan `db.t4g`.

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Kelas instance berikut dari Aurora My SQL version 2 tidak tersedia untuk Aurora My versi 3: SQL

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

Periksa skrip administrasi Anda untuk setiap CLI pernyataan yang membuat instance Aurora SQL My DB. Nama kelas instance hardcode yang tidak tersedia untuk Aurora SQL My versi 3. Jika perlu, ubah nama kelas instance menjadi nama yang didukung Aurora My SQL version 3.

> **Tip**
>
> Untuk memeriksa kelas instance yang dapat Anda gunakan untuk kombinasi spesifik Aurora SQL versi dan AWS Wilayah Saya, gunakan perintah. `describe-orderable-db-instance-options` AWS CLI

Untuk detail selengkapnya tentang kelas instans Aurora, lihat Kelas instans Amazon Aurora DB.

Perubahan parameter untuk Aurora Versi saya 3 SQL

Aurora My SQL version 3 mencakup parameter konfigurasi level cluster dan instance-level baru. Aurora SQL Versi saya 3 juga menghapus beberapa parameter yang ada di Aurora My versi 2. SQL Beberapa nama parameter diubah sebagai hasil dari inisiatif untuk bahasa inklusif. Untuk kompatibilitas mundur, Anda masih dapat mengambil nilai parameter menggunakan nama lama atau nama baru. Namun, Anda harus menggunakan nama baru untuk menentukan nilai parameter dalam grup parameter kustom.

Di Aurora My SQL versi 3, nilai `lower_case_table_names` parameter diatur secara permanen pada saat cluster dibuat. Jika Anda menggunakan nilai nondefault untuk opsi ini, siapkan grup parameter kustom Aurora SQL My version 3 Anda sebelum memutakhirkan. Kemudian, tentukan grup parameter selama operasi pembuatan klaster atau pemulihan snapshot.

> **Note**
>
> Dengan database global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan di tempat dari Aurora SQL My versi 2 ke versi 3 jika parameter dihidupkan. `lower_case_table_names` Gunakan metode pemulihan snapshot sebagai gantinya.

Di Aurora My SQL versi 3, `read_only` parameter `init_connect` dan tidak berlaku untuk pengguna yang memiliki hak istimewa. `CONNECTION_ADMIN` Ini termasuk pengguna master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran.

Untuk daftar lengkap parameter Aurora My SQL cluster, lihat. Parameter tingkat klaster Tabel ini mencakup semua parameter dari Aurora My SQL versi 2 dan 3. Tabel mencakup catatan yang menunjukkan parameter mana yang baru di Aurora My SQL versi 3 atau telah dihapus dari Aurora My versi 3. SQL

Untuk daftar lengkap parameter Aurora My SQL instance, lihat. Parameter tingkat instans Tabel ini mencakup semua parameter dari Aurora My SQL versi 2 dan 3. Tabel mencakup catatan yang menunjukkan parameter mana yang baru di Aurora My SQL versi 3 dan parameter mana yang dihapus dari Aurora My versi 3. SQL Ini juga mencakup catatan yang menunjukkan parameter mana yang dapat dimodifikasi di versi sebelumnya tetapi tidak Aurora My versi 3. SQL

Untuk informasi tentang nama parameter yang berubah, lihat Perubahan bahasa inklusif untuk Aurora Versi saya 3 SQL.

Variabel status

Untuk informasi tentang variabel status yang tidak berlaku untuk Aurora MySQL, lihat. Variabel SQL status saya yang tidak berlaku untuk Aurora My SQL

Perubahan bahasa inklusif untuk Aurora Versi saya 3 SQL

Aurora My SQL version 3 kompatibel dengan versi 8.0.23 dari edisi My community. SQL Aurora SQL Versi saya 3 juga mencakup perubahan dari My SQL 8.0.26 terkait dengan kata kunci dan skema sistem untuk bahasa inklusif. Misalnya, perintah `SHOW REPLICA STATUS` sekarang lebih disarankan daripada `SHOW SLAVE STATUS`.

CloudWatch Metrik Amazon berikut memiliki nama baru di Aurora SQL My versi 3.

Di Aurora My SQL versi 3, hanya nama metrik baru yang tersedia. Pastikan untuk memperbarui alarm atau otomatisasi lain yang bergantung pada nama metrik saat Anda meningkatkan ke Aurora My versi 3. SQL

| Nama lama | Nama baru |
|---|---|
| `ForwardingMasterDMLLatency` | `ForwardingWriterDMLLatency` |
| `ForwardingMasterOpenSessions` | `ForwardingWriterOpenSessions` |
| `AuroraDMLRejectedMasterFull` | `AuroraDMLRejectedWriterFull` |
| `ForwardingMasterDMLThroughput` | `ForwardingWriterDMLThroughput` |

Variabel status berikut memiliki nama baru di Aurora My SQL versi 3.

Untuk kompatibilitas, Anda dapat menggunakan salah satu nama dalam rilis awal Aurora My SQL versi 3. Nama variabel status lama akan dihapus dalam rilis mendatang.

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan |
|---|---|
| `Aurora_fwd_master_dml_stmt_duration` | `Aurora_fwd_writer_dml_stmt_duration` |
| `Aurora_fwd_master_dml_stmt_count` | `Aurora_fwd_writer_dml_stmt_count` |
| `Aurora_fwd_master_select_stmt_duration` | `Aurora_fwd_writer_select_stmt_duration` |
| `Aurora_fwd_master_select_stmt_count` | `Aurora_fwd_writer_select_stmt_count` |

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan | |
|---|---|---|
| `Aurora_fwd_master_errors_session_timeout` | `Aurora_fwd_writer_errors_session_timeout` | |
| `Aurora_fwd_master_open_sessions` | `Aurora_fwd_writer_open_sessions` | |
| `Aurora_fwd_master_errors_session_limit` | `Aurora_fwd_writer_errors_session_limit` | |
| `Aurora_fwd_master_errors_rpc_timeout` | `Aurora_fwd_writer_errors_rpc_timeout` | |

Parameter konfigurasi berikut memiliki nama baru di Aurora My SQL versi 3.

Untuk kompatibilitas, Anda dapat memeriksa nilai parameter di `mysql` klien dengan menggunakan salah satu nama dalam rilis Aurora My SQL versi 3 awal. Anda hanya dapat menggunakan nama baru saat memodifikasi nilai dalam grup parameter kustom. Nama parameter lama akan dihapus dalam rilis mendatang.

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan | |
|---|---|---|
| `aurora_fwd_master_idle_timeout` | `aurora_fwd_writer_idle_timeout` | |
| `aurora_fwd_master_max_connections_pct` | `aurora_fwd_writer_max_connections_pct` | |
| `master_verify_checksum` | `source_verify_checksum` | |
| `sync_master_info` | `sync_source_info` | |

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan | |
|---|---|---|
| init_slave | init_replica | |
| rpl_stop_slave_timeout | rpl_stop_replica_timeout | |
| log_slow_slave_statements | log_slow_replica_statements | |
| slave_max_allowed_packet | replica_max_allowed_packet | |
| slave_compressed_protocol | replica_compressed_protocol | |
| slave_exec_mode | replica_exec_mode | |
| slave_type_conversions | replica_type_conversions | |
| slave_sql_verify_checksum | replica_sql_verify_checksum | |
| slave_parallel_type | replica_parallel_type | |
| slave_preserve_commit_order | replica_preserve_commit_order | |
| log_slave_updates | log_replica_updates | |
| slave_allow_batching | replica_allow_batching | |
| slave_load_tmpdir | replica_load_tmpdir | |
| slave_net_timeout | replica_net_timeout | |

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan | |
|---|---|---|
| sql_slave_skip_counter | sql_replica_skip_counter | |
| slave_skip_errors | replica_skip_errors | |
| slave_checkpoint_period | replica_checkpoint_period | |
| slave_checkpoint_group | replica_checkpoint_group | |
| slave_transaction_retries | replica_transaction_retries | |
| slave_parallel_workers | replica_parallel_workers | |
| slave_pending_jobs_size_max | replica_pending_jobs_size_max | |
| pseudo_slave_mode | pseudo_replica_mode | |

Prosedur tersimpan berikut memiliki nama baru di Aurora My SQL versi 3.

Untuk kompatibilitas, Anda dapat menggunakan salah satu nama dalam rilis awal Aurora My SQL versi 3. Nama prosedur lama akan dihapus dalam rilis mendatang.

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan | |
|---|---|---|
| mysql.rds_set_master_auto_position | mysql.rds_set_source_auto_position | |
| mysql.rds_set_external_master | mysql.rds_set_external_source | |

| Nama yang akan dihapus | Nama yang baru atau direkomendasikan | |
|---|---|---|
| `mysql.rds_set_external_master_with_auto_position` | `mysql.rds_set_external_source_with_auto_position` | |
| `mysql.rds_reset_external_master` | `mysql.rds_reset_external_source` | |
| `mysql.rds_next_master_log` | `mysql.rds_next_source_log` | |

AUTO_ INCREMENT nilai

Di Aurora My SQL versi 3, Aurora mempertahankan `AUTO_INCREMENT` nilai untuk setiap tabel saat me-restart setiap instans DB. Di Aurora My SQL versi 2, `AUTO_INCREMENT` nilainya tidak dipertahankan setelah restart.

`AUTO_INCREMENT`Nilai tidak dipertahankan saat Anda menyiapkan klaster baru dengan memulihkan dari snapshot, melakukan point-in-time pemulihan, dan mengkloning cluster. Dalam kasus ini, nilai `AUTO_INCREMENT` diinisialisasi ke nilai berdasarkan nilai kolom terbesar dalam tabel pada saat snapshot dibuat. Perilaku ini berbeda dari pada RDS My SQL 8.0, di mana `AUTO_INCREMENT` nilainya dipertahankan selama operasi ini.

Replikasi log biner

Dalam edisi komunitas My SQL 8.0, replikasi log biner diaktifkan secara default. Di Aurora My SQL versi 3, replikasi log biner dimatikan secara default.

> **Tip**
>
> Jika persyaratan ketersediaan tinggi Anda dipenuhi oleh fitur replikasi default Aurora, Anda dapat menonaktifkan replikasi log biner. Dengan begitu, Anda dapat menghindari overhead performa replikasi log biner. Anda juga dapat menghindari pemantauan dan pemecahan masalah terkait yang diperlukan untuk mengelola replikasi log biner.

Aurora mendukung replikasi log biner dari sumber My SQL 5.7 yang kompatibel dengan Aurora My versi 3. SQL Sistem sumber dapat berupa cluster Aurora My SQL DB, instans RDS for My SQL DB, atau instans Saya lokal. SQL

Seperti halnya komunitas MySQL, Aurora My SQL mendukung replikasi dari sumber yang menjalankan versi tertentu ke target yang menjalankan versi utama yang sama atau satu versi utama yang lebih tinggi. Misalnya, replikasi dari SQL sistem My 5.6 yang kompatibel ke Aurora My SQL versi 3 tidak didukung. Mereplikasi dari Aurora SQL My versi 3 ke sistem My 5.7—kompatibel atau SQL SQL My 5.6—kompatibel tidak didukung. Untuk detail tentang menggunakan replikasi log biner, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Aurora SQL Versi saya 3 mencakup peningkatan replikasi log biner di komunitas My SQL 8.0, seperti replikasi yang difilter. Untuk detail tentang peningkatan komunitas My SQL 8.0, lihat Bagaimana Server Mengevaluasi Aturan Pemfilteran Replikasi di Manual Referensi Saya SQL.

Kompresi transaksi untuk replikasi log biner

Untuk informasi penggunaan tentang kompresi log biner, lihat Kompresi Transaksi Log Biner di Manual SQL Referensi Saya.

Batasan berikut berlaku untuk kompresi log biner di Aurora My SQL versi 3:

- Transaksi yang data log binernya lebih besar dari ukuran paket maksimum yang diizinkan tidak akan dikompresi. Ini benar terlepas dari apakah pengaturan kompresi log SQL biner Aurora My diaktifkan. Transaksi semacam itu direplikasi tanpa dikompresi.
- Jika Anda menggunakan konektor untuk change data capture (CDC) yang belum mendukung My SQL 8.0, Anda tidak dapat menggunakan fitur ini. Kami menyarankan Anda menguji konektor pihak ketiga secara menyeluruh dengan kompresi log biner. Selain itu, kami menyarankan Anda melakukannya sebelum mengaktifkan kompresi binlog pada sistem yang menggunakan replikasi binlog untuk. CDC

## Membandingkan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition

Anda dapat menggunakan informasi berikut untuk mempelajari tentang perubahan yang harus diperhatikan ketika Anda mengonversi dari sistem yang kompatibel dengan MySQL 8.0 yang berbeda ke Aurora MySQL versi 3.

Secara umum, Aurora MySQL versi 3 mendukung set fitur MySQL 8.0.23 komunitas. Beberapa fitur baru dari MySQL 8.0 edisi komunitas tidak berlaku untuk Aurora MySQL. Beberapa fitur tersebut tidak kompatibel dengan beberapa aspek Aurora, seperti arsitektur penyimpanan Aurora. Fitur lain tidak diperlukan karena layanan manajemen Amazon RDS menyediakan fungsionalitas yang setara. Fitur berikut di MySQL 8.0 komunitas tidak didukung atau berfungsi secara berbeda di Aurora MySQL versi 3.

Untuk catatan rilis untuk semua rilis Aurora MySQL versi 3, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 dalam Catatan Rilis untuk Aurora MySQL.

Topik


- Fitur MySQL 8.0 tidak tersedia di Aurora MySQL versi 3
- Model hak akses berbasis peran
- Menemukan ID server database
- Autentikasi


## Fitur MySQL 8.0 tidak tersedia di Aurora MySQL versi 3

Fitur berikut dari MySQL 8.0 komunitas tidak tersedia atau berfungsi secara berbeda di Aurora MySQL versi 3.

- Grup sumber daya dan pernyataan SQL terkait tidak didukung di Aurora MySQL.
- Aurora MySQL tidak mendukung ruang tabel undo yang ditentukan pengguna dan pernyataan SQL terkait, seperti, dan. `CREATE UNDO TABLESPACE` `ALTER UNDO TABLESPACE ... SET INACTIVE` `DROP UNDO TABLESPACE`
- Aurora MySQL tidak mendukung pemotongan ruang tabel batal untuk versi Aurora MySQL yang lebih rendah dari 3,06. Di Aurora MySQL versi 3.06 dan yang lebih tinggi, pemotongan tablespace undo otomatis didukung.
- Anda tidak dapat memodifikasi pengaturan plugin MySQL apa pun.
- Plugin X tidak didukung.
- Replikasi multisumber tidak didukung.

## Model hak akses berbasis peran

Dengan Aurora MySQL versi 3, Anda tidak dapat memodifikasi tabel dalam basis data `mysql` secara langsung. Secara khusus, Anda tidak dapat mengatur pengguna dengan melakukan penyisipan

ke dalam tabel `mysql.user`. Sebagai gantinya, Anda menggunakan pernyataan SQL untuk memberikan hak akses berbasis peran. Anda juga tidak dapat membuat jenis objek lain seperti prosedur tersimpan dalam basis data `mysql`. Anda masih dapat mengueri tabel `mysql`. Jika Anda menggunakan replikasi log biner, perubahan yang dilakukan langsung ke tabel `mysql` di klaster sumber tidak direplikasi ke klaster target.

Dalam beberapa kasus, aplikasi Anda mungkin menggunakan pintasan untuk membuat pengguna atau objek lain dengan melakukan penyisipan ke dalam tabel `mysql`. Jika demikian, ubah kode aplikasi Anda untuk menggunakan pernyataan yang sesuai seperti `CREATE USER`. Jika aplikasi Anda membuat prosedur tersimpan atau objek lain dalam basis data `mysql`, gunakan basis data yang berbeda sebagai gantinya.

Untuk mengekspor metadata bagi pengguna database selama migrasi dari database MySQL eksternal, Anda dapat menggunakan perintah MySQL Shell sebagai gantinya. `mysqldump` Untuk informasi selengkapnya, lihat Instance Dump Utility, Schema Dump Utility, dan Table Dump Utility.

Untuk menyederhanakan pengelolaan izin bagi banyak pengguna atau aplikasi, Anda dapat menggunakan pernyataan `CREATE ROLE` untuk membuat peran yang memiliki serangkaian izin. Kemudian, Anda dapat menggunakan pernyataan `GRANT` dan `SET ROLE` serta fungsi `current_role` untuk menetapkan peran ke pengguna atau aplikasi, mengganti peran saat ini, dan memeriksa peran mana yang berlaku. Untuk informasi selengkapnya tentang sistem izin berbasis peran di MySQL 8.0, lihat Using Roles dalam Panduan Referensi MySQL.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses paling rendah yang diperlukan untuk aplikasi Anda.

Topik

- rds_superuser_role
- Hak istimewa memeriksa pengguna untuk replikasi log biner
- Peran untuk mengakses layanan lain AWS

rds_superuser_role

Aurora MySQL versi 3 mencakup peran khusus yang memiliki semua hak akses berikut. Peran ini bernama `rds_superuser_role`. Pengguna administratif primer untuk setiap klaster sudah diberi peran ini. Peran `rds_superuser_role` mencakup hak akses berikut untuk semua objek basis data:

- `ALTER`
- `APPLICATION_PASSWORD_ADMIN`
- `ALTER ROUTINE`
- `CONNECTION_ADMIN`
- `CREATE`
- `CREATE ROLE`
- `CREATE ROUTINE`
- `CREATE TEMPORARY TABLES`
- `CREATE USER`
- `CREATE VIEW`
- `DELETE`
- `DROP`
- `DROP ROLE`
- `EVENT`
- `EXECUTE`
- `FLUSH_OPTIMIZER_COSTS`(Aurora MySQL versi 3.09 dan lebih tinggi)
- `FLUSH_STATUS`(Aurora MySQL versi 3.09 dan lebih tinggi)
- `FLUSH_TABLES`(Aurora MySQL versi 3.09 dan lebih tinggi)
- `FLUSH_USER_RESOURCES`(Aurora MySQL versi 3.09 dan lebih tinggi)
- `INDEX`
- `INSERT`
- `LOCK TABLES`
- `PROCESS`
- `REFERENCES`
- `RELOAD`
- `REPLICATION CLIENT`

- `REPLICATION SLAVE`
- `ROLE_ADMIN`
- `SET_USER_ID`
- `SELECT`
- `SHOW DATABASES`
- `SHOW_ROUTINE` (Aurora MySQL versi 3.04 dan lebih tinggi)
- `SHOW VIEW`
- `TRIGGER`
- `UPDATE`
- `XA_RECOVER_ADMIN`

Definisi peran juga mencakup `WITH GRANT OPTION` sehingga pengguna administratif dapat memberikan peran tersebut kepada pengguna lain. Secara khusus, administrator harus memberikan hak akses yang diperlukan untuk melakukan replikasi log biner dengan klaster Aurora MySQL sebagai target.

> **Tip**
>
> Untuk melihat detail lengkap izin, masukkan pernyataan berikut.
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

Hak istimewa memeriksa pengguna untuk replikasi log biner

Aurora MySQL versi 3 mencakup hak istimewa memeriksa pengguna untuk replikasi log biner (binlog),. `rdsrepladmin_priv_checks_user` Selain hak istimewa`rds_superuser_role`, pengguna ini memiliki `replication_applier` hak istimewa.

Ketika Anda mengaktifkan replikasi binlog dengan memanggil prosedur `mysql.rds_start_replication` tersimpan, `rdsrepladmin_priv_checks_user` dibuat.

`rdsrepladmin_priv_checks_user@localhost`Pengguna adalah pengguna yang dicadangkan. Jangan memodifikasinya.

Peran untuk mengakses layanan lain AWS

Aurora MySQL versi 3 mencakup peran yang dapat Anda gunakan untuk mengakses layanan lain. AWS Anda dapat menetapkan banyak peran ini sebagai alternatif untuk memberikan hak istimewa. Misalnya, Anda menentukan `GRANT AWS_LAMBDA_ACCESS TO user`, bukan `GRANT INVOKE LAMBDA ON *.* TO user`. Untuk prosedur untuk mengakses AWS layanan lain, lihatMengintegrasikan Amazon Aurora SQL My dengan layanan lain AWS. Aurora MySQL versi 3 mencakup peran berikut yang terkait dengan mengakses layanan lain: AWS

- `AWS_LAMBDA_ACCESS`— Sebuah alternatif untuk hak `INVOKE LAMBDA` istimewa. Untuk informasi penggunaan, lihat Memanggil fungsi Lambda dari cluster Amazon Aurora My DB SQL.
- `AWS_LOAD_S3_ACCESS`— Sebuah alternatif untuk hak `LOAD FROM S3` istimewa. Untuk informasi penggunaan, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3.
- `AWS_SELECT_S3_ACCESS`— Sebuah alternatif untuk hak `SELECT INTO S3` istimewa. Untuk informasi penggunaan, lihat Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3.
- `AWS_COMPREHEND_ACCESS`— Sebuah alternatif untuk hak `INVOKE COMPREHEND` istimewa. Untuk informasi penggunaan, lihat Memberikan akses pengguna basis data ke machine learning Aurora.
- `AWS_SAGEMAKER_ACCESS`— Sebuah alternatif untuk hak `INVOKE SAGEMAKER` istimewa. Untuk informasi penggunaan, lihat Memberikan akses pengguna basis data ke machine learning Aurora.
- `AWS_BEDROCK_ACCESS`— Tidak ada `INVOKE` hak istimewa analog untuk Amazon Bedrock. Untuk informasi penggunaan, lihat Memberikan akses pengguna basis data ke machine learning Aurora.

Ketika Anda memberikan akses dengan menggunakan peran di Aurora MySQL versi 3, Anda juga perlu mengaktifkan peran ini dengan menggunakan pernyataan `SET ROLE role_name` atau `SET ROLE ALL`. Contoh berikut menunjukkan cara melakukannya. Ganti nama peran yang sesuai untuk `AWS_SELECT_S3_ACCESS`.

```
# Grant role to user.

mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
```

```
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-----------------------------------------------------+
| CURRENT_ROLE()                                      |
+-----------------------------------------------------+
| `AWS_SELECT_S3_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-----------------------------------------------------+
```

Menemukan ID server database

ID server database (`server_id`) diperlukan untuk replikasi biner logging (binlog). Cara menemukan ID server berbeda di Aurora MySQL dari MySQL Komunitas.

Di MySQL Komunitas, ID server adalah nomor, yang Anda peroleh dengan menggunakan sintaks berikut saat masuk ke server:

```
mysql> select @@server_id;

+-------------+
| @@server_id |
+-------------+
| 2           |
+-------------+
1 row in set (0.00 sec)
```

Di Aurora MySQL, ID server adalah ID instans DB, yang Anda dapatkan dengan menggunakan sintaks berikut saat masuk ke instance DB:

```
mysql> select @@aurora_server_id;
```

```
+------------------------+
| @@aurora_server_id     |
+------------------------+
| mydbcluster-instance-2 |
+------------------------+
1 row in set (0.00 sec)
```

Untuk informasi lebih lanjut tentang replikasi binlog, lihat. Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner)

Autentikasi

Di MySQL 8.0 komunitas, plugin autentikasi default adalah `caching_sha2_password`. Aurora MySQL versi 3 masih menggunakan plugin `mysql_native_password`. Anda tidak dapat mengubah pengaturan `default_authentication_plugin`. Namun, Anda dapat membuat pengguna baru dan mengubah pengguna saat ini, dan kata sandi masing-masing menggunakan plugin otentikasi baru. Berikut adalah contohnya.

```
mysql> CREATE USER 'testnewsha'@'%' IDENTIFIED WITH caching_sha2_password BY
 'aNewShaPassword';
Query OK, 0 rows affected (0.74 sec)
```

### Meningkatkan ke Aurora MySQL versi 3

Untuk informasi tentang memutakhirkan database Anda dari Aurora MySQL versi 2 ke versi 3, lihat. Meningkatkan versi mayor klaster DB Amazon Aurora MySQL

## Aurora SQL Versi saya 2 kompatibel dengan My 5.7 SQL

Topik ini menjelaskan perbedaan antara Aurora My SQL version 2 dan My SQL 5.7 Community Edition.

> **Important**
>
> Aurora My SQL versi 2 mencapai akhir dukungan standar pada 31 Oktober 2024. Untuk informasi selengkapnya, lihat Mempersiapkan Amazon Aurora My SQL -Compatible Edition versi 2 akhir dukungan standar.

## Fitur tidak didukung di Aurora Versi saya 2 SQL

Fitur berikut didukung di My SQL 5.7, tetapi saat ini tidak didukung di Aurora SQL My versi 2:

- `CREATE TABLESPACE`SQLpernyataan
- Plugin replikasi kelompok
- Peningkatan ukuran halaman
- Pemuatan pool buffer InnoDB saat startup
- Plugin pengurai teks lengkap InnoDB
- Replikasi multisumber
- Perubahan ukuran pool buffer online
- Plugin validasi kata sandi – Anda dapat menginstal plugin ini, tetapi tidak didukung. Anda tidak dapat menyesuaikan plugin.
- Plugin tulis ulang kueri
- Penyaringan replikasi
- Protokol X

Untuk informasi selengkapnya tentang fitur-fitur ini, lihat dokumentasi SQL 5.7 Saya.

## Perilaku tablespace sementara di Aurora Versi saya 2 SQL

Di My SQL 5.7, tablespace sementara adalah autoextending dan bertambah besar ukurannya seperlunya untuk mengakomodasi tabel sementara on-disk. Ketika tabel sementara dihapus, ruang kosong dapat digunakan kembali untuk tabel sementara baru, tetapi ruang tabel sementara tetap pada ukuran yang diperluas dan tidak menyusut. Ruang tabel sementara dihapus dan dibuat ulang saat mesin diaktifkan ulang.

Di Aurora My SQL versi 2, perilaku berikut berlaku:

- Untuk cluster Aurora My SQL DB baru yang dibuat dengan versi 2.10 dan yang lebih tinggi, tablespace sementara dihapus dan dibuat ulang saat Anda me-restart database. Hal ini memungkinkan fitur pengubahan ukuran dinamis mengklaim kembali ruang penyimpanan.
- Untuk klaster Aurora My SQL DB yang ada ditingkatkan menjadi:
  - Versi 2.10 atau lebih tinggi – Ruang tabel sementara dihapus dan dibuat ulang saat Anda mengaktifkan ulang basis data. Hal ini memungkinkan fitur pengubahan ukuran dinamis mengklaim kembali ruang penyimpanan.

- Versi 2.09 – Ruang tabel sementara tidak dihapus saat Anda mengaktifkan ulang basis data.

Anda dapat memeriksa ukuran tablespace sementara pada klaster Aurora SQL My version 2 DB Anda dengan menggunakan kueri berikut:

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

Untuk informasi selengkapnya, lihat Ruang Meja Sementara di SQL Dokumentasi Saya.

## Mesin penyimpanan untuk tabel sementara di disk

Aurora SQL Versi saya 2 menggunakan mesin penyimpanan yang berbeda untuk tabel sementara internal on-disk tergantung pada peran instance.

- Pada instans penulis, tabel sementara di disk menggunakan mesin penyimpanan InnoDB secara default. Tabel tersebut disimpan di ruang tabel sementara di volume klaster Aurora.

  Anda dapat mengubah perilaku ini pada instans penulis dengan memodifikasi nilai untuk parameter DB `internal_tmp_disk_storage_engine`. Untuk informasi selengkapnya, lihat Parameter tingkat instans.
- Pada instance pembaca, tabel sementara pada disk menggunakan mesin ISAM penyimpanan Saya, yang menggunakan penyimpanan lokal. Itu karena instans hanya baca tidak dapat menyimpan data apa pun pada volume klaster Aurora.

# Keamanan dengan Amazon Aurora MySQL

Keamanan untuk Amazon Aurora MySQL dikelola di tiga tingkat:

- Untuk mengontrol siapa yang dapat melakukan tindakan pengelolaan Amazon RDS pada cluster DB MySQL Aurora dan instans DB, Anda menggunakan (IAM). AWS Identity and Access Management Saat Anda terhubung AWS menggunakan kredensyal IAM, AWS akun Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk melakukan operasi pengelolaan Amazon RDS. Untuk informasi selengkapnya, silakan lihat Manajemen identitas dan akses untuk Aurora

  Jika Anda menggunakan IAM untuk mengakses konsol Amazon RDS, pastikan untuk masuk terlebih dahulu AWS Management Console dengan kredensyal pengguna IAM Anda. Lalu pergi ke konsol Amazon RDS di https://console.aws.amazon.com/rds/.
- Pastikan untuk membuat klaster DB Aurora MySQL di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Untuk mengontrol perangkat dan EC2 instans Amazon mana yang dapat membuka koneksi ke titik akhir dan port instans DB untuk cluster DB MySQL Aurora di VPC, gunakan grup keamanan VPC. Anda dapat membuat titik akhir dan koneksi port ini menggunakan Keamanan Lapisan Pengangkutan (TLS). Selain itu, aturan firewall di perusahaan Anda dapat mengendalikan perangkat yang berjalan di perusahaan Anda untuk dapat membuka koneksi ke instans DB. Untuk informasi lebih lanjut tentang VPCs, lihatAmazon VPC dan Aurora.

  Penghunian VPC yang didukung tergantung pada kelas instans yang digunakan oleh klaster DB Aurora MySQL Anda. Dengan penghunian VPC `default`, VPC berjalan di perangkat keras bersama. Dengan penghunian VPC `dedicated`, VPC berjalan pada instans perangkat keras khusus. Kelas instans DB performa yang dapat melonjak hanya mendukung penghunian VPC default. Kelas instans DB performa yang dapat melonjak mencakup kelas instans db.t2, db.t3, dan db.t4g DB. Semua kelas instans Aurora MySQL DB lainnya mendukung penghunian VPC default dan khusus.

  > **Note**
  >
  > Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Untuk informasi selengkapnya tentang kelas instans, lihat Kelas instans Amazon Aurora DB. Untuk informasi selengkapnya tentang penghunian VPC `default` dan `dedicated`, lihat Instans khusus dalam Panduan Pengguna Amazon Elastic Compute Cloud.

- Untuk mengautentikasi login dan izin untuk klaster DB Amazon Aurora MySQL, Anda dapat mengambil salah satu dari pendekatan berikut, atau kombinasi dari pendekatan tersebut:
  - Anda dapat menggunakan pendekatan yang sama dengan instans MySQL mandiri.

    Perintah seperti `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE`, dan `SET PASSWORD` beroperasi dalam basis data on-premise seperti halnya memodifikasi langsung tabel skema basis data. Untuk informasi selengkapnya, lihat Access control and account management dalam dokumentasi MySQL.
  - Anda juga dapat menggunakan autentikasi basis data IAM.

    Dengan autentikasi basis data IAM, Anda mengautentikasi klaster DB Anda dengan menggunakan pengguna IAM atau peran IAM dan token autentikasi. Token autentikasi adalah nilai unik yang dihasilkan dengan menggunakan proses penandatanganan Signature Versi 4. Dengan menggunakan autentikasi basis data IAM, Anda dapat menggunakan kredensyal yang sama untuk mengontrol akses ke AWS sumber daya dan basis data Anda. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

> **Note**
>
> Untuk informasi selengkapnya, lihat Keamanan dalam Amazon Aurora.

Di bagian berikut, lihat informasi tentang izin pengguna untuk koneksi Aurora MySQL dan TLS dengan cluster DB MySQL Aurora.

Topik

- Hak akses pengguna master Amazon Aurora MySQL
- Koneksi TLS ke cluster DB MySQL Aurora

## Hak akses pengguna master Amazon Aurora MySQL

Saat Anda membuat instans DB Amazon Aurora MySQL, pengguna master memiliki hak default yang tercantum di Hak akses akun pengguna master.

Untuk menyediakan layanan manajemen untuk setiap klaster DB, pengguna `admin` dan `rdsadmin` dibuat saat klaster DB dibuat. Mencoba menghapus, mengubah nama, mengubah kata sandi, atau mengubah hak akses untuk akun `rdsadmin` akan mengakibatkan kesalahan.

Dalam klaster DB Aurora MySQL versi 2, pengguna `admin` dan `rdsadmin` dibuat ketika klaster DB dibuat. Dalam klaster DB Aurora MySQL versi 3, pengguna `admin`, `rdsadmin`, dan `rds_superuser_role` dibuat.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses minimal yang diperlukan untuk aplikasi Anda.

Untuk manajemen klaster DB Aurora MySQL, perintah `kill` dan `kill_query` standar telah dibatasi. Sebagai gantinya, gunakan perintah Amazon RDS `rds_kill` dan `rds_kill_query` untuk menghentikan sesi pengguna atau kueri di instans DB Aurora MySQL.

> **Note**
>
> Enkripsi instans basis data dan snapshot tidak didukung untuk wilayah Tiongkok (Ningxia).

## Koneksi TLS ke cluster DB MySQL Aurora

Klaster DB Amazon Aurora MySQL mendukung koneksi Keamanan Lapisan Pengangkutan (TLS) dari aplikasi yang menggunakan proses dan kunci publik yang sama seperti instans DB RDS for MySQL.

Amazon RDS membuat sertifikat TLS dan menginstal sertifikat tersebut pada instans DB saat Amazon RDS menyediakan instans. Sertifikat ini ditandatangani oleh otoritas sertifikat. Sertifikat TLS mencakup titik akhir instans DB sebagai Nama Umum (CN) untuk sertifikat TLS agar melindungi dari serangan spoofing. Oleh karena itu, Anda hanya dapat menggunakan titik akhir klaster DB untuk

terhubung ke klaster DB menggunakan TLS jika klien Anda mendukung Nama Alternatif Subjek (SAN). Jika tidak, Anda harus menggunakan titik akhir instans milik instans penulis.

Untuk informasi tentang mengunduh sertifikat, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

Kami merekomendasikan Driver AWS JDBC sebagai klien yang mendukung SAN dengan TLS. Untuk informasi selengkapnya tentang Driver AWS JDBC dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) JDBC Driver. GitHub

Topik

- Mewajibkan koneksi TLS ke klaster DB Aurora MySQL
- Versi TLS untuk Aurora MySQL
- Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL
- Mengenkripsi koneksi ke klaster DB Aurora MySQL

## Mewajibkan koneksi TLS ke klaster DB Aurora MySQL

Anda dapat mewajibkan agar semua koneksi pengguna ke klaster DB Aurora MySQL Anda menggunakan TLS dengan parameter klaster DB `require_secure_transport`. Secara default, parameter `require_secure_transport` diatur ke `OFF`. Anda dapat mengatur parameter `require_secure_transport` ke `ON` untuk mewajibkan TLS untuk koneksi ke klaster DB Anda.

Anda dapat mengatur nilai parameter `require_secure_transport` dengan memperbarui grup parameter klaster DB untuk klaster DB Anda. Anda tidak perlu mem-boot ulang klaster DB Anda agar perubahan dapat diterapkan. Untuk informasi selengkapnya tentang grup parameter, lihat .

> **Note**
>
> Parameter `require_secure_transport` tersedia untuk Aurora MySQL versi 2 dan 3. Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom. Parameter ini tidak tersedia dalam grup parameter instans DB.

Saat parameter `require_secure_transport` diatur ke `ON` untuk klaster DB, klien basis data dapat terhubung dengannya jika klien basis data ini dapat membuat koneksi terenkripsi. Jika tidak, pesan kesalahan seperti yang berikut ini ditampilkan kepada klien:

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

## Versi TLS untuk Aurora MySQL

Aurora MySQL mendukung Keamanan Lapisan Pengangkutan (TLS) versi 1.0, 1.1, 1.2, 1.2, dan 1.3. Mulai dari Aurora MySQL versi 3.04.0 dan lebih tinggi, Anda dapat menggunakan protokol TLS 1.3 untuk mengamankan koneksi Anda. Tabel berikut menunjukkan dukungan TLS untuk versi Aurora MySQL.

| Versi Aurora MySQL | TLS 1.0 | TLS 1.1 | TLS 1.2 | TLS 1.3 | Default |
|---|---|---|---|---|---|
| Aurora MySQL versi 2 | Didukung | Didukung | Didukung | Tidak Support | Semua versi TLS yang didukung |
| Aurora MySQL versi 3 (lebih rendah dari 3.04.0) | Didukung | Didukung | Didukung | Tidak Support | Semua versi TLS yang didukung |
| Aurora MySQL versi 3 (3.04.0 dan lebih tinggi) | Tidak didukung | Tidak didukung | Didukung | Didukung | Semua versi TLS yang didukung |

**Important**

Jika Anda menggunakan grup parameter khusus untuk cluster Aurora MySQL Anda dengan versi 2 dan versi yang lebih rendah dari 3.04.0, sebaiknya gunakan TLS 1.2 karena TLS 1.0 dan 1.1 kurang aman. Edisi komunitas MySQL 8.0.26 dan Aurora MySQL 3.03 dan versi minornya menghentikan dukungan untuk TLS versi 1.1 dan 1.0.
Edisi komunitas MySQL 8.0.28 dan Aurora MySQL versi 3.04.0 dan lebih tinggi yang kompatibel tidak mendukung TLS 1.1 dan TLS 1.0. Jika Anda menggunakan Aurora MySQL

versi 3.04.0 atau lebih tinggi, jangan atur protokol TLS ke 1.0 dan 1.1 di grup parameter kustom Anda.
Untuk Aurora MySQL versi 3.04.0 dan yang lebih tinggi, pengaturan default-nya adalah TLS 1.3 dan TLS 1.2.

Anda dapat menggunakan parameter klaster DB `tls_version` untuk menunjukkan versi protokol yang diizinkan. Parameter klien serupa ada untuk sebagian besar alat klien atau driver basis data. Beberapa klien lama mungkin tidak mendukung versi TLS yang lebih baru. Secara default, klaster DB mencoba menggunakan versi protokol TLS tertinggi yang diizinkan oleh konfigurasi server dan klien.

Atur parameter klaster DB `tls_version` ke salah satu nilai berikut:

- `TLSv1.3`
- `TLSv1.2`
- `TLSv1.1`
- `TLSv1`

Anda juga dapat mengatur parameter `tls_version` sebagai string daftar dipisahkan koma. Jika Anda ingin menggunakan protokol TLS 1.2 dan TLS 1.0, parameter `tls_version` harus menyertakan semua protokol dari protokol terendah hingga protokol tertinggi. Dalam hal ini, `tls_version` ditetapkan sebagai:

```
tls_version=TLSv1,TLSv1.1,TLSv1.2
```

Untuk informasi tentang memodifikasi parameter dalam grup parameter klaster DB, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora. Jika Anda menggunakan AWS CLI untuk memodifikasi parameter cluster `tls_version` DB, `ApplyMethod` harus diatur kepending-reboot. Jika metode aplikasinya adalah `pending-reboot`, perubahan parameter diterapkan setelah Anda menghentikan dan mengaktifkan ulang klaster DB yang terkait dengan grup parameter.

## Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data Anda. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS klien ke basis data Anda. Dengan cipher suite yang

dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Melakukan hal ini akan mencegah penggunaan cipher yang tidak aman atau usang.

Cipher suite yang dapat dikonfigurasi didukung di Aurora MySQL versi 3 dan Aurora MySQL versi 2. Untuk menentukan daftar cipher TLS 1.2, TLS 1.1, atau TLS 1.0 yang diizinkan untuk mengenkripsi koneksi, modifikasi parameter klaster `ssl_cipher`. Atur parameter `ssl_cipher` dalam grup parameter klaster menggunakan AWS Management Console, AWS CLI, atau API RDS.

Atur parameter `ssl_cipher` ke string nilai cipher yang dipisahkan koma untuk versi TLS Anda. Untuk aplikasi klien, Anda dapat menentukan cipher yang akan digunakan untuk koneksi terenkripsi dengan menggunakan opsi `--ssl-cipher` saat menghubungkan ke basis data. Untuk informasi selengkapnya tentang melakukan koneksi ke basis data Anda, lihat Menghubungkan ke klaster DB Amazon Aurora MySQL.

Mulai dari Aurora MySQL versi 3.04.0 dan lebih tinggi, Anda dapat menentukan cipher suite TLS 1.3. Untuk menentukan cipher suite TLS 1.3 yang diizinkan, ubah parameter `tls_ciphersuites` dalam grup parameter Anda. TLS 1.3 telah mengurangi jumlah cipher suite yang tersedia karena perubahan dalam konvensi penamaan yang menghapus mekanisme pertukaran kunci dan sertifikat yang digunakan. Atur `tls_ciphersuites` ke string nilai cipher dipisahkan koma untuk TLS 1.3.

Tabel berikut menunjukkan cipher yang didukung bersama dengan protokol enkripsi TLS dan versi mesin Aurora MySQL yang valid untuk setiap cipher.

| Cipher | Protokol enkripsi | Versi Aurora MySQL yang didukung |
|---|---|---|
| `ECDHE-RSA-AES128-SHA` | TLS 1.0 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `ECDHE-RSA-AES128-SHA256` | TLS 1.2 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `ECDHE-RSA-AES128-GCM-SHA256` | TLS 1.2 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `ECDHE-RSA-AES256-SHA` | TLS 1.0 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |

| Cipher | Protokol enkripsi | Versi Aurora MySQL yang didukung |
|---|---|---|
| `ECDHE-RSA-AES256-GCM-SHA384` | TLS 1.2 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `ECDHE-ECDSA-AES128-GCM-SHA256` | TLS 1.2 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `ECDHE-ECDSA-AES256-GCM-SHA384` | TLS 1.2 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `ECDHE-ECDSA-CHACHA20-POLY1305` | TLS 1.2 | 3.04.0 dan lebih tinggi, 2.11.0 dan lebih tinggi |
| `TLS_AES_128_GCM_SHA256` | TLS 1.3 | 3.04.0 dan lebih tinggi |
| `TLS_AES_256_GCM_SHA384` | TLS 1.3 | 3.04.0 dan lebih tinggi |
| `TLS_CHACHA20_POLY1305_SHA256` | TLS 1.3 | 3.04.0 dan lebih tinggi |

**Important**

`DHE-RSA`cipher tidak didukung oleh versi Aurora MySQL mulai dari 2.11.0. di Aurora MySQL 2, dan mulai dari 3.04.0 di Aurora MySQL 3.

Untuk informasi tentang memodifikasi parameter dalam grup parameter klaster DB, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora. Jika Anda menggunakan CLI untuk memodifikasi parameter klaster DB `ssl_cipher`, pastikan untuk mengatur `ApplyMethod` ke `pending-reboot`. Jika metode aplikasinya adalah `pending-reboot`, perubahan parameter diterapkan setelah Anda menghentikan dan mengaktifkan ulang klaster DB yang terkait dengan grup parameter.

Anda juga dapat menggunakan perintah describe-engine-default-cluster-parameter CLI untuk menentukan rangkaian sandi mana yang saat ini didukung untuk keluarga grup parameter tertentu. Contoh berikut menunjukkan cara mendapatkan nilai yang diizinkan untuk parameter klaster `ssl_cipher` untuk Aurora MySQL versi 2.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
  "ParameterName": "ssl_cipher",
  "ParameterValue": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-
AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "Description": "The list of permissible ciphers for connection encryption.",
  "Source": "system",
  "ApplyType": "static",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-
RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "IsModifiable": true,
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
       ...some output truncated...
```

Untuk informasi selengkapnya tentang cipher, lihat variabel ssl_cipher dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang format cipher suite, lihat openssl-ciphers list format dan dokumentasi openssl-ciphers string format di situs web OpenSSL.

## Mengenkripsi koneksi ke klaster DB Aurora MySQL

Untuk mengenkripsi koneksi menggunakan klien `mysql` default, luncurkan klien mysql menggunakan parameter `--ssl-ca` untuk merujuk ke kunci publik, misalnya:

Untuk MySQL 5.7 dan 8.0:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
```

```
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

Untuk MySQL 5.6:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

Ganti *full_path_to_CA_certificate* dengan jalur lengkap ke sertifikat Certificate Authority (CA) Anda. Untuk informasi tentang mengunduh sertifikat, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

Anda dapat mewajibkan koneksi TLS untuk akun pengguna tertentu. Misalnya, Anda dapat menggunakan salah satu pernyataan berikut, bergantung pada versi MySQL Anda, untuk mewajibkan koneksi TLS pada `encrypted_user` akun pengguna.

Untuk MySQL 5.7 dan 8.0:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

Untuk MySQL 5.6:

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

Saat Anda menggunakan Proksi RDS, Anda terhubung ke titik akhir proksi, bukan titik akhir klaster biasa. Anda dapat menjadikan SSL/TLS wajib atau opsional untuk koneksi ke proksi, dengan cara yang sama seperti untuk koneksi langsung ke klaster DB Aurora. Untuk informasi tentang menggunakan Proksi RDS, lihat Proksi Amazon RDS untuk Aurora.

> **Note**
>
> Untuk informasi selengkapnya tentang koneksi TLS dengan MySQL, lihat dokumentasi MySQL.

# Memperbarui aplikasi untuk terhubung ke klaster DB Aurora MySQL menggunakan sertifikat TLS baru

Sejak 13 Januari 2023, Amazon RDS telah menerbitkan serifikat Otoritas Sertifikat (CA) baru untuk terhubung ke klaster DB Aurora menggunakan Keamanan Lapisan Pengangkutan (TLS). Setelah itu, Anda dapat menemukan informasi tentang pembaruan aplikasi untuk menggunakan sertifikat baru.

Topik ini dapat membantu Anda menentukan apakah aplikasi klien menggunakan TLS untuk terhubung ke klaster DB Anda. Jika demikian, Anda dapat memeriksa lebih lanjut apakah aplikasi tersebut memerlukan verifikasi sertifikat untuk terhubung.

> **Note**
>
> Beberapa aplikasi dikonfigurasi untuk terhubung ke klaster DB Aurora MySQL hanya jika aplikasi tersebut berhasil memverifikasi sertifikat pada server.
> Untuk aplikasi tersebut, Anda harus memperbarui penyimpanan kepercayaan aplikasi klien untuk menyertakan sertifikat CA baru.

Setelah memperbarui sertifikat CA di penyimpanan kepercayaan aplikasi klien, Anda dapat merotasi sertifikat di klaster DB Anda. Sebaiknya Anda menguji prosedur ini di lingkungan pengembangan dan penahapan sebelum menerapkannya di lingkungan produksi Anda.

Untuk informasi selengkapnya tentang rotasi sertifikat, lihat Merotasi sertifikat SSL/TLS. Untuk informasi selengkapnya tentang cara mengunduh sertifikat, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke . Untuk informasi tetang cara menggunakan TLS dengan klaster DB Aurora MySQL, lihat Koneksi TLS ke cluster DB MySQL Aurora.

Topik

- Menentukan apakah ada aplikasi yang tersambung ke klaster DB Aurora MySQL menggunakan TLS
- Menentukan apakah klien memerlukan verifikasi sertifikat untuk terhubung
- Memperbarui penyimpanan kepercayaan aplikasi Anda
- Contoh kode Java untuk membangun koneksi TLS

## Menentukan apakah ada aplikasi yang tersambung ke klaster DB Aurora MySQL menggunakan TLS

Jika Anda menggunakan Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7) dan Skema Performa diaktifkan, jalankan kueri berikut untuk memeriksa apakah koneksi menggunakan TLS. Untuk informasi tentang cara mengaktifkan Skema Performa, lihat Memulai cepat Skema Performa dalam dokumentasi MySQL.

```
mysql> SELECT id, user, host, connection_type
       FROM performance_schema.threads pst
       INNER JOIN information_schema.processlist isp
       ON pst.processlist_id = isp.id;
```

Dalam output contoh ini, Anda dapat melihat sesi Anda sendiri (`admin`) dan aplikasi yang masuk sebagai `webapp1` menggunakan TLS.

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

## Menentukan apakah klien memerlukan verifikasi sertifikat untuk terhubung

Anda dapat memeriksa apakah klien JDBC dan klien MySQL memerlukan verifikasi sertifikat untuk terhubung.

### JDBC

Contoh MySQL Connector/J 8.0 berikut menunjukkan satu cara untuk memeriksa properti koneksi JDBC aplikasi untuk menentukan apakah koneksi yang berhasil memerlukan sertifikat yang valid. Untuk informasi selengkapnya tentang semua opsi koneksi JDBC untuk MySQL, lihat Properti konfigurasi dalam dokumentasi MySQL.

Saat menggunakan MySQL Connector/J 8.0, koneksi TLS memerlukan verifikasi terhadap sertifikat CA server jika properti koneksi Anda memiliki `sslMode` yang diatur ke `VERIFY_CA` atau `VERIFY_IDENTITY`, seperti pada contoh berikut.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> Jika Anda menggunakan MySQL Java Connector v5.1.38 atau yang lebih baru, atau MySQL Java Connector v8.0.9 atau yang lebih baru untuk terhubung ke basis data Anda, meski Anda belum mengonfigurasi aplikasi secara ekplisit untuk menggunakan TLS saat terhubung ke basis data Anda, driver klien ini akan menggunakan TLS secara default. Selain itu, saat menggunakan TLS, aplikasi akan melakukan verifikasi sertifikat parsial dan gagal terhubung jika sertifikat server basis data kedaluwarsa.

### MySQL

Contoh Klien MySQL berikut menunjukkan dua cara untuk memeriksa koneksi MySQL skrip untuk menentukan apakah koneksi yang berhasil memerlukan sertifikat yang valid. Untuk informasi selengkapnya tentang semua opsi koneksi dengan Klien MySQL, lihat Konfigurasi sisi klien untuk koneksi terenkripsi dalam dokumentasi MySQL.

Saat menggunakan Klien MySQL 5.7 atau MySQL 8.0, koneksi TLS memerlukan verifikasi terhadap sertifikat CA server jika, untuk opsi `--ssl-mode`, Anda menentukan `VERIFY_CA` atau `VERIFY_IDENTITY`, seperti pada contoh berikut.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

Saat menggunakan Klien MySQL 5.6, koneksi SSL memerlukan verifikasi terhadap sertifikat CA server jika Anda menentukan opsi `--ssl-verify-server-cert`, seperti pada contoh berikut.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

## Memperbarui penyimpanan kepercayaan aplikasi Anda

Untuk informasi tentang cara memperbarui penyimpanan kepercayaan untuk aplikasi MySQL, lihat Menginstal sertifikat SSL dalam dokumentasi MySQL.

> **Note**
>
> Saat memperbarui penyimpanan kepercayaan, Anda dapat mempertahankan sertifikat lama selain menambahkan sertifikat baru.

### Memperbarui penyimpanan kepercayaan aplikasi Anda untuk JDBC

Anda dapat memperbarui penyimpanan kepercayaan untuk aplikasi yang menggunakan JDBC untuk koneksi TLS.

Untuk informasi tentang cara mengunduh sertifikat root, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

Untuk contoh skrip yang mengimpor sertifikat, lihat Contoh skrip untuk mengimpor sertifikat ke trust store Anda.

Jika Anda menggunakan driver JDBC mysql dalam aplikasi, atur properti berikut dalam aplikasi.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> Tentukan kata sandi selain perintah yang ditampilkan di sini sebagai praktik keamanan terbaik.

Saat Anda memulai aplikasi, atur properti berikut.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

## Contoh kode Java untuk membangun koneksi TLS

Contoh kode berikut menunjukkan cara menyiapkan koneksi SSL yang memvalidasi sertifikat server menggunakan JDBC.

```
public class MySQLSSLTest {
```

```
        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
}
```

**Important**

Setelah Anda menentukan bahwa koneksi database Anda menggunakan TLS dan telah memperbarui toko kepercayaan aplikasi Anda, Anda dapat memperbarui database Anda untuk menggunakan sertifikat rds-ca-rsa 2048-g1. Untuk petunjuk, lihat langkah 3 dalam Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB Anda.

# Menggunakan otentikasi Kerberos untuk Aurora My SQL

Anda dapat menggunakan otentikasi Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke cluster Aurora My DB Anda. SQL Untuk melakukannya, konfigurasikan cluster DB Anda untuk digunakan AWS Directory Service for Microsoft Active Directory untuk otentikasi Kerberos. AWS Directory Service for Microsoft Active Directory disebut juga AWS Managed Microsoft AD. Ini adalah fitur yang tersedia dengan AWS Directory Service. Untuk mempelajari lebih lanjut, lihat Apa itu AWS Directory Service? dalam Panduan AWS Directory Service Administrasi.

Untuk memulai, buat AWS Managed Microsoft AD direktori untuk menyimpan kredensyal pengguna. Kemudian, berikan domain Active Directory dan informasi lainnya ke cluster Aurora My SQL DB Anda. Ketika pengguna mengautentikasi dengan cluster Aurora SQL My DB, permintaan otentikasi diteruskan ke direktori. AWS Managed Microsoft AD

Dengan menyimpan semua kredensial Anda di direktori yang sama, Anda dapat menghemat waktu dan tenaga. Dengan pendekatan ini, Anda memiliki sebuah lokasi terpusat untuk menyimpan dan mengelola kredensial bagi beberapa klaster DB. Menggunakan direktori juga dapat meningkatkan profil keamanan keseluruhan Anda.

Selain itu, Anda dapat mengakses kredensial dari Microsoft Active Directory on-premise Anda sendiri. Untuk melakukannya, buat hubungan domain tepercaya sehingga direktori AWS Managed Microsoft AD mempercayai Microsoft Active Directory on-premise Anda. Dengan cara ini, pengguna Anda dapat mengakses klaster Aurora My SQL DB Anda dengan pengalaman masuk tunggal (SSO) Windows yang sama seperti ketika mereka mengakses beban kerja di jaringan lokal Anda.

Database dapat menggunakan Kerberos, AWS Identity and Access Management (IAM), atau Kerberos dan otentikasi. IAM Namun, karena Kerberos dan IAM otentikasi menyediakan metode otentikasi yang berbeda, pengguna tertentu dapat masuk ke database hanya menggunakan satu atau metode otentikasi lainnya, tetapi tidak keduanya. Untuk informasi selengkapnya tentang IAM otentikasi, lihatAutentikasi basis data IAM.

Daftar Isi


- Ikhtisar otentikasi Kerberos untuk cluster Aurora My DB SQL
- Batasan otentikasi Kerberos untuk Aurora My SQL
- Menyiapkan autentikasi Kerberos untuk klaster DB Aurora MySQL
  - Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD
  - Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise

- Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora
- Langkah 4: Buat dan konfigurasikan pengguna
- Langkah 5: Buat atau modifikasi klaster DB Aurora MySQL
- Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos
  - Memodifikasi login Aurora MySQL yang ada
- Langkah 7: Konfigurasi klien MySQL
- Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil
- Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos
  - Menggunakan login Kerberos Aurora MySQL untuk terhubung ke klaster DB
  - Autentikasi Kerberos dengan basis data global Aurora
  - Migrasi dari RDS for MySQL ke Aurora MySQL
  - Mencegah caching tiket
  - Pencatatan log untuk autentikasi Kerberos
- Mengelola klaster DB di dalam domain
  - Memahami keanggotaan domain


## Ikhtisar otentikasi Kerberos untuk cluster Aurora My DB SQL

Untuk mengatur otentikasi Kerberos untuk cluster Aurora My SQL DB, selesaikan langkah-langkah umum berikut. Langkah ini dijelaskan secara lebih mendetail nanti.

1. Gunakan AWS Managed Microsoft AD untuk membuat AWS Managed Microsoft AD direktori. Anda dapat menggunakan AWS Management Console, yang AWS CLI, atau AWS Directory Service untuk membuat direktori. Untuk petunjuk terperinci, lihat Membuat AWS Managed Microsoft AD direktori Anda di Panduan AWS Directory Service Administrasi.
2. Buat peran AWS Identity and Access Management (IAM) yang menggunakan IAM kebijakan terkelola`AmazonRDSDirectoryServiceAccess`. Peran ini memungkinkan Amazon Aurora untuk melakukan panggilan ke direktori Anda.

   Agar peran mengizinkan akses, titik akhir AWS Security Token Service (AWS STS) harus diaktifkan di Wilayah AWS untuk AWS akun Anda. AWS STS endpoint aktif secara default di semua Wilayah AWS, dan Anda dapat menggunakannya tanpa tindakan lebih lanjut. Untuk

informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan AWS STSWilayah AWS dalam Panduan Pengguna. IAM

3. Buat dan konfigurasikan pengguna di AWS Managed Microsoft AD direktori menggunakan alat Microsoft Active Directory. Untuk informasi selengkapnya tentang membuat pengguna di Active Directory, lihat Mengelola pengguna dan grup di Microsoft AD AWS terkelola di Panduan AWS Directory Service Administrasi.
4. Buat atau modifikasi cluster Aurora My SQL DB. Jika Anda menggunakan salah satu CLI atau RDS API dalam permintaan buat, tentukan pengenal domain dengan `Domain` parameter. Gunakan `d-*` pengenal yang dihasilkan saat Anda membuat direktori dan nama IAM peran yang Anda buat.

   Jika Anda memodifikasi cluster Aurora My SQL DB yang ada untuk menggunakan otentikasi Kerberos, tetapkan parameter domain dan IAM peran untuk cluster DB. Temukan cluster DB VPC sama dengan direktori domain.
5. Gunakan kredensyal pengguna RDS utama Amazon untuk terhubung ke klaster Aurora My DB. SQL Buat pengguna database di Aurora My SQL dengan menggunakan instruksi di. Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos

   Pengguna yang Anda buat dengan cara ini dapat masuk ke cluster Aurora My SQL DB menggunakan otentikasi Kerberos. Untuk informasi selengkapnya, lihat Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos.

Untuk menggunakan autentikasi Kerberos dengan Microsoft Active Directory on-premise atau yang di-host mandiri, buat sebuah trust forest. Trust forest adalah hubungan kepercayaan antara dua kelompok domain. Kepercayaan bisa satu arah atau dua arah. Untuk informasi selengkapnya tentang penggunaan trust hutan AWS Directory Service, lihat Kapan membuat hubungan kepercayaan dalam Panduan AWS Directory Service Administrasi.

## Batasan otentikasi Kerberos untuk Aurora My SQL

Batasan berikut berlaku untuk otentikasi Kerberos untuk Aurora My: SQL

- Otentikasi Kerberos didukung untuk Aurora My SQL versi 3.03 dan yang lebih tinggi.

  Untuk informasi tentang Wilayah AWS dukungan, lihatAutentikasi Kerberos dengan Aurora MySQL.
- Untuk menggunakan otentikasi Kerberos dengan Aurora MySQL, SQL klien atau konektor Saya harus menggunakan versi 8.0.26 atau lebih tinggi pada platform Unix, 8.0.27 atau lebih tinggi di

Windows. Jika tidak, plugin `authentication_kerberos_client` sisi klien tidak tersedia dan Anda tidak dapat mengautentikasi.

- Hanya AWS Managed Microsoft AD didukung di Aurora My. SQL Namun, Anda dapat bergabung dengan klaster Aurora My SQL DB ke domain Microsoft AD Terkelola bersama yang dimiliki oleh akun yang berbeda secara bersamaan. Wilayah AWS

  Anda juga dapat menggunakan Active Directory on-premise Anda sendiri. Untuk informasi selengkapnya, lihat Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise.
- Saat menggunakan Kerberos untuk mengautentikasi pengguna yang terhubung ke Aurora My SQL cluster dari SQL Klien saya atau dari driver di sistem operasi Windows, secara default kasus karakter nama pengguna database harus cocok dengan kasus pengguna di Direktori Aktif. Misalnya, jika pengguna di Active Directory muncul sebagai `Admin`, nama pengguna basis data harus `Admin`.

  Namun, Anda sekarang dapat menggunakan perbandingan nama pengguna yang tidak peka huruf besar/kecil dengan plugin `authentication_kerberos`. Untuk informasi selengkapnya, lihat Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil.
- Anda harus melakukan boot ulang instans DB pembaca setelah mengaktifkan fitur untuk menginstal plugin `authentication_kerberos`.
- Replikasi ke instans DB yang tidak mendukung plugin `authentication_kerberos` dapat menyebabkan kegagalan replikasi.
- Agar basis data global Aurora dapat menggunakan autentikasi Kerberos, Anda harus mengonfigurasinya untuk setiap klaster DB di dalam basis data global.
- Nama domain harus kurang dari 62 karakter.
- Jangan memodifikasi port klaster DB setelah mengaktifkan autentikasi Kerberos. Jika Anda memodifikasi port, autentikasi Kerberos tidak akan berfungsi lagi.

## Menyiapkan autentikasi Kerberos untuk klaster DB Aurora MySQL

Gunakan AWS Managed Microsoft AD untuk mengatur otentikasi Kerberos untuk cluster DB MySQL Aurora. Untuk menyiapkan autentikasi Kerberos, lakukan langkah berikut.

Topik

- Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD

- Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise
- Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora
- Langkah 4: Buat dan konfigurasikan pengguna
- Langkah 5: Buat atau modifikasi klaster DB Aurora MySQL
- Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos
- Langkah 7: Konfigurasi klien MySQL
- Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil

## Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD

AWS Directory Service membuat Direktori Aktif yang dikelola sepenuhnya di AWS Cloud. Saat Anda membuat AWS Managed Microsoft AD direktori, AWS Directory Service buat dua pengontrol domain dan server Domain Name System (DNS) atas nama Anda. Server-server direktori dibuat di subnet yang berbeda di VPC. Redundansi ini membantu memastikan bahwa direktori Anda tetap dapat diakses meskipun terjadi kegagalan.

Saat Anda membuat AWS Managed Microsoft AD direktori, AWS Directory Service lakukan tugas-tugas berikut atas nama Anda:

- Menyiapkan Active Directory dalam VPC.
- Membuat akun administrator direktori dengan nama pengguna `Admin` dan kata sandi yang ditentukan. Anda menggunakan akun ini untuk mengelola direktori Anda.

> **Note**
>
> Pastikan untuk menyimpan kata sandi ini. AWS Directory Service tidak menyimpannya. Anda dapat mengaturnya ulang, tetapi tidak dapat mengambilnya.

- Membuat grup keamanan untuk pengontrol direktori.

Saat Anda meluncurkan AWS Managed Microsoft AD, AWS buat Unit Organisasi (OU) yang berisi semua objek direktori Anda. OU ini memiliki nama NetBIOS yang Anda masukkan saat membuat direktori Anda. Itu terletak di root domain, yang dimiliki dan dikelola oleh AWS.

`Admin`Akun yang dibuat dengan AWS Managed Microsoft AD direktori Anda memiliki izin untuk kegiatan administratif yang paling umum untuk OU Anda, termasuk:

- Membuat, memperbarui, atau menghapus pengguna
- Tambahkan sumber daya ke domain Anda, seperti server file atau server cetak, kemudian tetapkan izin untuk sumber daya itu kepada pengguna di OU Anda
- Buat tambahan OUs dan wadah
- Melimpahkan kewenangan
- Memulihkan objek-objek yang dihapus dari Keranjang Sampah Active Directory
- Jalankan PowerShell modul AD dan DNS Windows pada Layanan Web Direktori Aktif

Akun `Admin` juga memiliki hak untuk melakukan aktivitas di seluruh domain berikut:

- Mengelola konfigurasi DNS (menambahkan, menghapus, atau memperbarui catatan, zona, dan penerus)
- Melihat log peristiwa DNS
- Melihat log peristiwa keamanan

Untuk membuat direktori dengan AWS Managed Microsoft AD

1. Masuk ke AWS Management Console dan buka AWS Directory Service konsol di https://console.aws.amazon.com/directoryservicev2/.
2. Di panel navigasi, pilih Direktori, lalu pilih Siapkan direktori.
3. Pilih AWS Managed Microsoft AD. AWS Managed Microsoft AD adalah satu-satunya pilihan yang saat ini dapat Anda gunakan dengan Amazon RDS.
4. Masukkan informasi berikut:

   Nama DNS Direktori

   Nama berkualifikasi penuh untuk direktori, seperti **`corp.example.com`**.

   Nama NetBIOS direktori

   Nama singkat untuk direktori, seperti **`CORP`**.

   Deskripsi direktori

   (Opsional) Deskripsi untuk direktori.

Kata sandi admin

Kata sandi untuk administrator direktori. Proses pembuatan direktori menciptakan akun administrator dengan nama pengguna Admin dan kata sandi ini.

Kata sandi administrator direktori dan tidak boleh menyertakan kata “admin”. Kata sandi bersifat peka kapital dan harus terdiri atas 8–64 karakter. Kata sandi juga harus berisi setidaknya satu karakter dari tiga di antara empat kategori berikut:

- Huruf kecil (a-z)
- Huruf besar (A-Z)
- Angka (0–9)
- Karakter non-alfanumerik (~!@#$%^&*_-+=`|\(){}[]:;'"<>,.?/)

Konfirmasikan kata sandi

Kata sandi administrator dimasukkan kembali.

5. Pilih Selanjutnya.
6. Masukkan informasi berikut di bagian Jaringan, lalu pilih Berikutnya:

   VPC

   VPC untuk direktori. Buat klaster DB Aurora MySQL di VPC yang sama ini.

   Subnet

   Subnet untuk server direktori. Kedua subnet harus berada di Zona Ketersediaan yang berbeda.
7. Tinjau informasi direktori dan buat perubahan yang diperlukan. Jika informasi sudah benar, pilih Buat direktori.

Dibutuhkan beberapa menit untuk membuat direktori. Setelah direktori berhasil dibuat, nilai Status berubah menjadi Aktif.

Untuk melihat informasi tentang direktori Anda, pilih nama direktori di daftar direktori. Catat nilai ID Direktori karena Anda memerlukan nilai ini saat membuat atau memodifikasi klaster DB Aurora MySQL Anda.

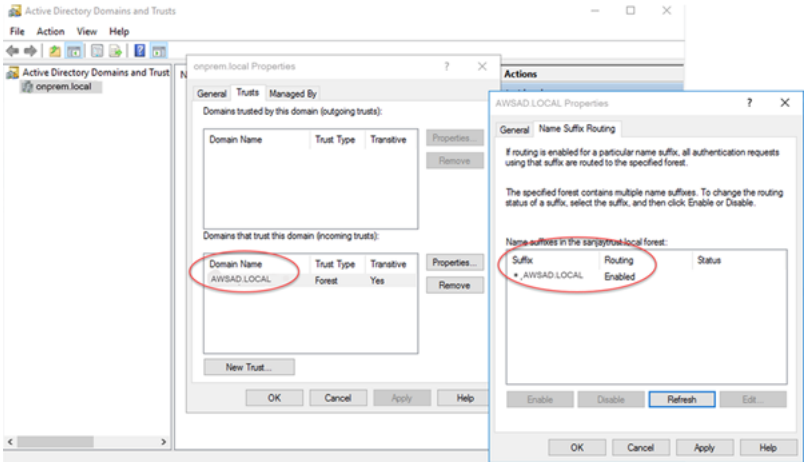
## Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise

Jika Anda tidak berencana menggunakan Microsoft Active Directory on-premise Anda sendiri, langsung lompat ke Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora.

Untuk menggunakan autentikasi Kerberos dengan Active Directory lokal, Anda perlu membuat hubungan domain yang dapat dipercaya menggunakan trust hutan antara Microsoft Active Directory lokal dan direktori (dibuat di AWS Managed Microsoft AD ). Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD Kepercayaan dapat bersifat satu arah, di mana direktori AWS Managed Microsoft AD mempercayai Microsoft Active Directory on-premise. Kepercayaan juga dapat bersifat dua arah, di mana kedua Active Directory saling mempercayai. Untuk informasi selengkapnya tentang menyiapkan trust menggunakan AWS Directory Service, lihat Kapan membuat hubungan kepercayaan di Panduan AWS Directory Service Administrasi.

> **Note**
>
> Jika Anda menggunakan Microsoft Active Directory on-premise:
>
> - Klien Windows tidak dapat terhubung menggunakan titik akhir kustom Aurora. Untuk mempelajari selengkapnya, lihat Koneksi titik akhir Amazon Aurora.
> - Untuk basis data global:
>   - Klien Windows dapat terhubung menggunakan titik akhir instans atau klaster di Wilayah AWS utama dari basis data global saja.
>   - Klien Windows tidak dapat terhubung menggunakan titik akhir cluster di sekunder Wilayah AWS.

Pastikan bahwa nama domain Microsoft Active Directory on-premise Anda mencakup perutean akhiran DNS yang sesuai dengan hubungan kepercayaan yang baru dibuat. Tangkapan layar berikut menunjukkan sebuah contoh.




## Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora

Agar Amazon Aurora memanggil AWS Directory Service Anda, Anda memerlukan peran AWS Identity and Access Management (IAM) yang menggunakan kebijakan IAM terkelola. `AmazonRDSDirectoryServiceAccess` Peran ini memungkinkan Aurora untuk melakukan panggilan ke AWS Directory Service.

Saat Anda membuat cluster DB menggunakan AWS Management Console, dan Anda memiliki `iam:CreateRole` izin, konsol akan membuat peran ini secara otomatis. Dalam hal ini, nama perannya adalah `rds-directoryservice-kerberos-access-role`. Jika tidak, Anda harus membuat peran IAM secara manual. Saat Anda membuat peran IAM ini, pilih`Directory Service`, dan lampirkan kebijakan AWS terkelola `AmazonRDSDirectoryServiceAccess` padanya.

Untuk informasi selengkapnya tentang membuat peran IAM untuk layanan, lihat Membuat peran untuk mendelegasikan izin ke AWS layanan di Panduan Pengguna IAM.

Anda memiliki opsi untuk membuat kebijakan dengan izin yang diperlukan alih-alih menggunakan kebijakan IAM yang dikelola `AmazonRDSDirectoryServiceAccess`. Dalam hal ini, peran IAM harus memiliki kebijakan kepercayaan IAM berikut.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Peran ini juga harus memiliki kebijakan peran IAM berikut.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

## Langkah 4: Buat dan konfigurasikan pengguna

Anda dapat membuat pengguna dengan alat Pengguna Active Directory dan Komputer. Alat ini bagian dari alat-alat Active Directory Domain Services dan Active Directory Lightweight Directory Services. Pengguna mewakili orang atau entitas individual yang memiliki akses ke direktori Anda.

Untuk membuat pengguna di AWS Directory Service direktori, Anda menggunakan EC2 instans lokal atau Amazon berdasarkan Microsoft Windows yang digabungkan ke AWS Directory Service direktori Anda. Anda harus login ke instans sebagai pengguna yang memiliki hak istimewa untuk membuat

pengguna. Untuk informasi selengkapnya, lihat Mengelola pengguna dan grup di AWS Managed Microsoft AD di Panduan Administrasi Layanan Direktori AWS .

### Langkah 5: Buat atau modifikasi klaster DB Aurora MySQL

Buat atau modifikasi klaster DB Aurora MySQL untuk penggunaan dengan direktori Anda. Anda dapat menggunakan konsol, AWS CLI, atau RDS API untuk mengaitkan cluster DB dengan direktori. Anda dapat melakukan tugas ini dengan salah satu cara berikut:

- Buat cluster Aurora MySQL DB baru menggunakan konsol, perintah CLI, atau operasi Create create-db-clusterRDS API. DBCluster

  Untuk petunjuk, lihat Membuat klaster DB Amazon Aurora.
- Ubah cluster Aurora MySQL DB yang ada menggunakan konsol, perintah CLI, atau operasi Modify modify-db-clusterRDS API. DBCluster

  Untuk petunjuk, lihat Memodifikasi klaster DB Amazon Aurora.
- Kembalikan cluster DB MySQL Aurora dari snapshot DB menggunakan konsol, perintah restore-db-cluster-fromCLI -snapshot, atau operasi Restore RDS API. DBCluster FromSnapshot

  Untuk petunjuk, lihat Memulihkan dari snapshot klaster DB.
- Kembalikan cluster Aurora MySQL DB ke point-in-time menggunakan konsol, perintah - restore-db-cluster-topoint-in-timeCLI, atau operasi Restore RDS API. DBCluster ToPointInTime

  Untuk petunjuk, lihat Memulihkan klaster DB ke waktu tertentu.

Autentikasi Kerberos hanya didukung untuk klaster DB Aurora MySQL dalam VPC. Klaster DB boleh berada dalam VPC yang sama dengan direktori, atau dalam VPC yang berbeda. VPC klaster DB harus memiliki grup keamanan VPC yang memungkinkan komunikasi keluar ke direktori Anda.

Konsol

Saat Anda menggunakan konsol untuk membuat, memodifikasi, atau memulihkan klaster DB, pilih Autentikasi Kerberos di bagian Autentikasi basis data. Pilih Jelajah Direktori lalu pilih direktori, atau pilih Buat direktori baru.

AWS CLI

Saat Anda menggunakan AWS CLI atau RDS API, kaitkan cluster DB dengan direktori. Parameter berikut diperlukan agar klaster DB dapat menggunakan direktori domain yang Anda buat:

- Untuk parameter `--domain`, gunakan pengidentifikasi domain (pengidentifikasi "d-*") yang dihasilkan saat Anda membuat direktori.
- Untuk parameter `--domain-iam-role-name`, gunakan peran yang Anda buat dengan menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`.

Misalnya, perintah CLI berikut memodifikasi klaster DB untuk menggunakan direktori.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --domain d-ID \
    --domain-iam-role-name role-name
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> Jika Anda memodifikasi klaster DB untuk mengaktifkan autentikasi Kerberos, lakukan boot ulang terhadap instans DB pembaca setelah melakukan perubahan.

## Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos

Cluster DB bergabung ke AWS Managed Microsoft AD domain. Jadi, Anda dapat membuat pengguna Aurora MySQL dari pengguna Active Directory di domain Anda. Izin basis data dikelola melalui izin Aurora MySQL standar yang diberikan kepada dan dicabut dari pengguna ini.

Anda dapat mengizinkan pengguna Active Directory untuk mengautentikasi dengan Aurora MySQL. Untuk melakukannya, pertama-tama gunakan kredensial pengguna primer Amazon RDS untuk terhubung ke klaster DB Aurora MySQL seperti klaster DB lainnya. Setelah Anda login, buat

pengguna yang diautentikasi secara eksternal dengan autentikasi Kerberos di Aurora MySQL seperti yang ditunjukkan di sini:

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- Ganti *user_name* dengan nama pengguna. Pengguna (baik manusia maupun aplikasi) dari domain Anda sekarang dapat terhubung ke klaster DB dari mesin klien yang digabungkan ke domain menggunakan autentikasi Kerberos.
- Ganti *host_name* dengan nama host. Anda dapat menggunakan `%` sebagai wildcard. Anda juga dapat menggunakan alamat IP tertentu untuk nama host.
- Ganti *realm_name* dengan nama ranah direktori domain. Nama realm biasanya sama dengan nama domain DNS dalam huruf besar, seperti `CORP.EXAMPLE.COM`. Realm adalah sekelompok sistem yang menggunakan Pusat Distribusi Kunci Kerberos yang sama.

Contoh berikut ini menciptakan pengguna basis data dengan nama `Admin` yang mengautentikasi terhadap Active Directory dengan nama realm `MYSQL.LOCAL`.

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

Memodifikasi login Aurora MySQL yang ada

Anda juga dapat memodifikasi login Aurora MySQL yang ada untuk menggunakan autentikasi Kerberos dengan menggunakan sintaks berikut:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

## Langkah 7: Konfigurasi klien MySQL

Untuk mengonfigurasi klien MySQL, lakukan langkah-langkah berikut:

1. Buat file `krb5.conf` (atau yang setara) untuk menunjuk ke domain.
2. Verifikasi bahwa lalu lintas dapat mengalir antara host klien dan AWS Directory Service. Gunakan utilitas jaringan seperti Netcat, untuk tugas-tugas berikut:
   - Memeriksa lalu lintas melalui DNS untuk port 53.
   - Verifikasi lalu lintas TCP/UDP untuk port 53 dan untuk Kerberos, yang mencakup port 88 dan 464 untuk. AWS Directory Service

3. Periksa bahwa lalu lintas dapat mengalir antara host klien dan instans basis data melalui port basis data. Misalnya, gunakan `mysql` untuk menghubungkan dan mengakses basis data.

Berikut ini adalah contoh `krb5.conf` konten untuk AWS Managed Microsoft AD.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

Berikut ini adalah contoh konten `krb5.conf` untuk Microsoft Active Directory on-premise.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

### Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil

Secara default, huruf besar/kecil nama pengguna basis data MySQL harus sesuai dengan login Active Directory. Namun, Anda sekarang dapat menggunakan perbandingan nama pengguna yang tidak peka huruf besar/kecil dengan plugin `authentication_kerberos`. Untuk melakukannya, Anda mengatur parameter klaster DB `authentication_kerberos_caseins_cmp` ke `true`.

Menggunakan perbandingan nama pengguna yang tidak peka huruf besar/kecil

1. Buat grup parameter klaster DB kustom. Ikuti prosedur di Membuat grup parameter cluster DB di Amazon Aurora.
2. Edit grup parameter baru untuk mengatur nilai `authentication_kerberos_caseins_cmp` ke `true`. Ikuti prosedur di Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.
3. Kaitkan grup parameter klaster DB dengan klaster DB Aurora MySQL Anda. Ikuti prosedur di Mengaitkan grup parameter cluster DB dengan cluster DB di Amazon Aurora.
4. Boot ulang klaster DB.

## Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos

Untuk menghindari kesalahan, gunakan klien MySQL dengan versi 8.0.26 atau lebih tinggi pada platform Unix, 8.0.27 atau lebih tinggi di Windows.

### Menggunakan login Kerberos Aurora MySQL untuk terhubung ke klaster DB

Untuk membuat koneksi ke Aurora MySQL dengan autentikasi Kerberos, Anda masuk sebagai pengguna basis data yang Anda buat menggunakan instruksi di Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos.

Pada prompt perintah, buat koneksi ke salah satu titik akhir yang terkait dengan klaster DB Aurora MySQL Anda. Saat Anda diminta memasukkan kata sandi, masukkan kata sandi Kerberos yang terkait dengan nama penggunanya.

Saat Anda mengautentikasi dengan Kerberos, sebuah tiket pemberian tiket (TGT) dibuat jika belum ada. Plugin `authentication_kerberos` menggunakan TGT untuk mendapatkan tiket layanan, yang kemudian disajikan ke server basis data Aurora MySQL.

Anda dapat menggunakan klien MySQL untuk terhubung ke Aurora MySQL dengan autentikasi Kerberos menggunakan Windows atau Unix.

Unix

Anda dapat terhubung dengan menggunakan salah satu metode berikut:

- Dapatkan TGT secara manual. Dalam hal ini, Anda tidak perlu memberikan kata sandi ke klien MySQL.
- Berikan kata sandi untuk login Active Directory langsung ke klien MySQL.

Plugin sisi klien didukung pada platform Unix untuk klien MySQL versi 8.0.26 dan yang lebih tinggi.

Terhubung dengan mendapatkan TGT secara manual

1. Pada antarmuka baris perintah, gunakan perintah berikut untuk mendapatkan TGT.

```
kinit user_name
```

2. Gunakan perintah `mysql` berikut ini untuk masuk ke titik akhir instans DB klaster DB Anda.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

> **Note**
>
> Autentikasi dapat gagal jika keytab dirotasi pada instans DB. Dalam hal ini, dapatkan TGT baru dengan menjalankan ulang `kinit`.

Terhubung secara langsung

1. Pada antarmuka baris perintah, gunakan perintah `mysql` berikut ini untuk masuk ke titik akhir instans DB klaster DB Anda.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

2. Masukkan kata sandi untuk pengguna Direktori Aktif.

Windows

Pada Windows, autentikasi biasanya dilakukan pada waktu login, jadi Anda tidak perlu mendapatkan TGT secara manual untuk terhubung ke klaster DB Aurora MySQL. Huruf besar/kecil pada nama pengguna basis data harus sesuai dengan karakter pengguna di Active Directory. Misalnya, jika pengguna di Active Directory muncul sebagai `Admin`, nama pengguna basis data harus `Admin`.

Plugin sisi klien didukung pada Windows untuk klien MySQL versi 8.0.27 dan yang lebih tinggi.

Terhubung secara langsung

- Pada antarmuka baris perintah, gunakan perintah `mysql` berikut ini untuk masuk ke titik akhir instans DB klaster DB Anda.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name
```

## Autentikasi Kerberos dengan basis data global Aurora

Autentikasi Kerberos untuk Aurora MySQL didukung untuk basis data global Aurora. Untuk mengautentikasi pengguna pada klaster DB sekunder menggunakan Active Directory klaster DB primer, lakukan replikasi Active Directory ke Wilayah AWS sekunder. Anda mengaktifkan autentikasi Kerberos pada klaster sekunder menggunakan ID domain yang sama seperti untuk klaster primer. Replikasi AWS Managed Microsoft AD hanya didukung dengan versi Active Directory Enterprise. Untuk informasi selengkapnya, lihat Replikasi Multi-Wilayah di Panduan Administrasi AWS Directory Service.

## Migrasi dari RDS for MySQL ke Aurora MySQL

Setelah Anda bermigrasi dari RDS for MySQL dengan autentikasi Kerberos diaktifkan ke Aurora MySQL, ubah pengguna yang dibuat dengan plugin `auth_pam` untuk menggunakan plugin `authentication_kerberos`. Sebagai contoh:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

## Mencegah caching tiket

Jika TGT yang valid tidak ada saat aplikasi klien MySQL dimulai, aplikasi dapat memperoleh dan meng-cache TGT. Jika Anda ingin mencegah caching TGT, atur parameter konfigurasi dalam file `/etc/krb5.conf`.

> **Note**
>
> Konfigurasi ini hanya berlaku untuk host klien yang menjalankan Unix, bukan Windows.

Mencegah caching TGT

- Tambahkan bagian `[appdefaults]` ke `/etc/krb5.conf` seperti berikut:

```
[appdefaults]
  mysql = {
    destroy_tickets = true
  }
```

## Pencatatan log untuk autentikasi Kerberos

Variabel lingkungan `AUTHENTICATION_KERBEROS_CLIENT_LOG` menetapkan tingkat pencatatan log untuk autentikasi Kerberos. Anda dapat menggunakan log untuk debugging sisi klien.

Nilai yang diizinkan adalah 1-5. Pesan log ditulis ke output kesalahan standar. Tabel berikut ini menjelaskan setiap tingkat pencatatan log.

| Tingkat pencatatan log | Deskripsi |
|---|---|
| 1 atau tidak diatur | Tidak ada pencatatan log |
| 2 | Pesan kesalahan |
| 3 | Pesan kesalahan dan peringatan |
| 4 | Pesan kesalahan, peringatan, dan informasi |
| 5 | Pesan kesalahan, peringatan, informasi, dan debug |

# Mengelola klaster DB di dalam domain

Anda dapat menggunakan AWS CLI atau API RDS untuk mengelola klaster DB dan hubungannya dengan Active Directory terkelola Anda. Misalnya, Anda dapat mengaitkan Active Directory untuk autentikasi Kerberos dan melepaskan Active Direktory untuk menonaktifkan autentikasi Kerberos. Anda juga dapat memindahkan klaster DB untuk diautentikasi secara eksternal oleh satu Active Directory ke yang lain.

Misalnya, menggunakan API Amazon RDS, Anda dapat melakukan hal berikut:

- Untuk mencoba ulang pengaktifan autentikasi Kerberos untuk keanggotaan yang gagal, gunakan operasi API `ModifyDBInstance` dan tentukan ID direktori keanggotaan saat ini.
- Untuk memperbarui nama peran IAM untuk keanggotaan, gunakan operasi API `ModifyDBInstance` dan tentukan ID direktori keanggotaan saat ini dan peran IAM baru.
- Untuk menonaktifkan autentikasi Kerberos pada klaster DB, gunakan operasi API `ModifyDBInstance` dan tentukan `none` sebagai parameter domain.
- Untuk memindahkan klaster DB dari satu domain ke domain lain, gunakan operasi API `ModifyDBInstance` dan tentukan pengidentifikasi domain baru sebagai parameter domain.
- Untuk memerinci keanggotaan bagi setiap klaster DB, gunakan operasi API `DescribeDBInstances`.

## Memahami keanggotaan domain

Setelah Anda membuat atau mengubah klaster DB, klaster tersebut akan menjadi anggota domain. Anda dapat melihat status keanggotaan domain untuk klaster DB dengan menjalankan perintah CLI describe-db-clusters. Status klaster DB dapat berupa salah satu hal berikut ini:

- `kerberos-enabled` – Klaster DB mengaktifkan autentikasi Kerberos.
- `enabling-kerberos` – AWS sedang dalam proses mengaktifkan autentikasi Kerberos pada klaster DB ini.
- `pending-enable-kerberos` – Mengaktifkan autentikasi Kerberos tertunda pada klaster DB ini.
- `pending-maintenance-enable-kerberos` – AWS akan mencoba mengaktifkan autentikasi Kerberos pada klaster DB selama periode pemeliharaan terjadwal berikutnya.
- `pending-disable-kerberos` – Menonaktifkan autentikasi Kerberos tertunda pada klaster DB ini.

- `pending-maintenance-disable-kerberos` – AWS akan mencoba menonaktifkan autentikasi Kerberos pada klaster DB selama periode pemeliharaan terjadwal berikutnya.
- `enable-kerberos-failed` – Masalah konfigurasi telah mencegah AWS dari mengaktifkan autentikasi Kerberos pada klaster DB. Periksa dan perbaiki konfigurasi Anda sebelum menerbitkan ulang perintah modifikasi klaster DB.
- `disabling-kerberos` – AWS sedang dalam proses menonaktifkan autentikasi Kerberos pada klaster DB ini.

Permintaan untuk mengaktifkan autentikasi Kerberos dapat gagal karena masalah konektivitas jaringan atau peran IAM yang salah. Misalnya, anggaplah Anda membuat klaster DB atau memodifikasi klaster DB yang sudah ada dan upaya untuk mengaktifkan autentikasi Kerberos tersebut gagal. Dalam hal ini, terbitkan ulang perintah modifikasi atau modifikasi klaster DB yang baru dibuat untuk bergabung ke domain.

# Migrasi data ke cluster Amazon Aurora My DB SQL

Anda memiliki beberapa opsi untuk memigrasikan data dari database yang ada ke kluster Amazon Aurora SQL My DB. Opsi migrasi Anda juga bergantung pada database yang Anda migrasikan dan ukuran data yang Anda migrasikan.

Ada dua jenis migrasi yang berbeda: fisik dan logis. Migrasi fisik berarti salinan fisik file basis data digunakan untuk memigrasikan basis data. Migrasi logis berarti bahwa migrasi dilakukan dengan menerapkan perubahan basis data logis, seperti penyisipan, pembaruan, dan penghapusan.

Migrasi fisik memiliki keuntungan sebagai berikut:

- Migrasi fisik lebih cepat daripada migrasi logis, terutama untuk basis data besar.
- Performa basis data tidak terpengaruh saat cadangan diambil untuk migrasi fisik.
- Migrasi fisik dapat memigrasikan semua hal dalam basis data sumber, termasuk komponen basis data yang kompleks.

Migrasi fisik memiliki batasan sebagai berikut:

- Parameter `innodb_page_size` harus diatur ke nilai default-nya (`16KB`).
- Parameter `innodb_data_file_path` harus dikonfigurasi dengan hanya satu file data yang menggunakan nama file data default `"ibdata1:12M:autoextend"`. Basis data yang berisi dua file data, atau memiliki file data dengan nama yang berbeda, tidak dapat dimigrasi menggunakan metode ini.

  Berikut ini adalah contoh nama file yang tidak diizinkan: `"innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend"` dan `"innodb_data_file_path=ibdata01:50M:autoextend"`.
- Parameter `innodb_log_files_in_group` harus diatur ke nilai default-nya (`2`).

Migrasi logis memiliki keuntungan sebagai berikut:

- Anda dapat memigrasikan subset basis data, seperti tabel atau bagian tertentu dari tabel.
- Data dapat dimigrasikan terlepas dari struktur penyimpanan fisik.

Migrasi logis memiliki batasan sebagai berikut:

- Migrasi logis biasanya lebih lambat daripada migrasi fisik.
- Komponen basis data yang kompleks dapat memperlambat proses migrasi logis. Dalam beberapa kasus, komponen basis data kompleks bahkan dapat memblokir migrasi logis.

Tabel berikut menjelaskan opsi dan jenis migrasi untuk setiap opsi.

| Migrasi dari | Jenis migrasi | Solusi |
| --- | --- | --- |
| Sebuah RDS untuk instance SQL DB Saya | Fisik | Anda dapat bermigrasi dari instans RDS for My SQL DB dengan terlebih dahulu membuat replika baca Aurora SQL My dari instance My DB. SQL Ketika jeda replika antara instans My SQL DB dan Aurora SQL My read replica adalah 0, Anda dapat mengarahkan aplikasi klien Anda untuk membaca dari replika baca Aurora dan kemudian menghenti kan replikasi untuk menjadikan replika baca Aurora My sebagai cluster Aurora SQL My DB mandiri untuk membaca dan menulis. SQL Untuk detailnya, lihat Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora. |
| Sebuah RDS untuk snapshot SQL DB Saya | Fisik | Anda dapat memigrasikan data langsung dari snapshot RDS for My SQL DB ke cluster Amazon Aurora SQL My DB. Untuk detailnya, lihat Memigrasi kan snapshot RDS for MySQL ke Aurora. |
| SQLDatabase saya di luar Amazon RDS | Logis | Anda dapat membuat dump data menggunakan `mysqldump` utilitas, lalu mengimpor data tersebut ke cluster Amazon Aurora SQL My DB yang ada. Untuk detailnya, lihat Migrasi logis dari My SQL ke Amazon Aurora My SQL dengan menggunakan mysqldump. |

| Migrasi dari | Jenis migrasi | Solusi |
| --- | --- | --- |
| | | Untuk mengekspor metadata bagi pengguna database selama migrasi dari SQL database Saya eksternal , Anda juga dapat menggunakan perintah My SQL Shell sebagai gantinya. `mysqldump` Untuk informasi selengkapnya, lihat Instance Dump Utility, Schema Dump Utility, dan Table Dump Utility.<br><br>**Note**<br>Utilitas mysqlpump tidak digunakan lagi pada My 8.0.34. SQL |
| SQLDatabase saya di luar Amazon RDS | Fisik | Anda dapat menyalin file cadangan dari database ke bucket Amazon Simple Storage Service (Amazon S3), lalu memulihkan cluster Amazon Aurora SQL My DB dari file-file tersebut. Opsi ini dapat jauh lebih cepat dibanding kan memigrasikan data menggunak an `mysqldump` . Untuk detailnya, lihat Migrasi fisik dari MySQL dengan menggunakan XtraBackup Percona dan Amazon S3. |

| Migrasi dari | Jenis migrasi | Solusi |
| --- | --- | --- |
| SQLDatabase saya di luar Amazon RDS | Logis | Anda dapat menyimpan data dari basis data Anda sebagai file teks dan menyalin file tersebut ke bucket Amazon S3. Anda kemudian dapat memuat data tersebut ke cluster Aurora My SQL DB yang ada menggunakan perintah `LOAD DATA FROM S3` SayaSQL. Untuk informasi selengkapnya, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3. |
| Database yang tidak kompatibel dengan Saya SQL | Logis | Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan data dari database yang tidak kompatibel dengan SayaSQL. Untuk informasi selengkap nya AWS DMS, lihat Apa itu layanan migrasi AWS database? |

> **Note**
>
> Jika Anda memigrasikan SQL database Saya di luar AmazonRDS, opsi migrasi yang dijelaskan dalam tabel hanya didukung jika database Anda mendukung InnoDB atau ISAM ruang tabel Saya.
>
> Jika SQL database Saya yang Anda migrasi ke Aurora SQL My `memcached` menggunakan, `memcached` hapus sebelum memigrasinya.
>
> Anda tidak dapat bermigrasi ke Aurora SQL My versi 3.05 dan yang lebih tinggi dari beberapa versi SQL My 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan Anda meningkatkan ke SQL versi Saya 8.0.28 sebelum bermigrasi.

# Memigrasikan data dari basis data MySQL eksternal ke klaster DB Amazon Aurora MySQL

Jika basis data Anda mendukung ruang tabel InnoDB atau MyISAM, Anda memiliki opsi ini untuk memigrasikan data Anda ke klaster DB Amazon Aurora MySQL:

- Anda dapat membuat dump data menggunakan utilitas `mysqldump`, lalu mengimpor data tersebut ke klaster DB Amazon Aurora MySQL yang ada. Untuk informasi selengkapnya, lihat Migrasi logis dari My SQL ke Amazon Aurora My SQL dengan menggunakan mysqldump.
- Anda dapat menyalin file cadangan penuh dan cadangan inkremental dari basis data Anda ke bucket Amazon S3, lalu memulihkan klaster DB Amazon Aurora MySQL dari file tersebut. Opsi ini dapat jauh lebih cepat dibandingkan memigrasikan data menggunakan `mysqldump`. Untuk informasi selengkapnya, lihat Migrasi fisik dari MySQL dengan menggunakan XtraBackup Percona dan Amazon S3.

Topik


- Migrasi fisik dari MySQL dengan menggunakan XtraBackup Percona dan Amazon S3
- Migrasi logis dari My SQL ke Amazon Aurora My SQL dengan menggunakan mysqldump


## Migrasi fisik dari MySQL dengan menggunakan XtraBackup Percona dan Amazon S3

Anda dapat menyalin file cadangan penuh dan inkremental dari basis data sumber MySQL versi 5.7 atau 8.0 ke bucket Amazon S3. Kemudian, Anda dapat memulihkan ke klaster DB Amazon Aurora MySQL dengan versi mesin DB mayor yang sama dari file-file tersebut.

Opsi ini bisa jadi jauh lebih cepat dibandingkan memigrasi data menggunakan `mysqldump` karena penggunaan `mysqldump` akan mengulangi semua perintah untuk membuat kembali skema dan data dari basis data sumber di klaster DB Aurora MySQL baru Anda. Dengan menyalin file data MySQL sumber Anda, Aurora MySQL dapat segera menggunakan file tersebut sebagai data untuk klaster DB Aurora MySQL.

Anda juga dapat meminimalkan waktu henti dengan menggunakan replikasi log biner selama proses migrasi. Jika Anda menggunakan replikasi log biner, basis data MySQL eksternal tetap terbuka untuk transaksi saat data sedang dimigrasi ke klaster DB Aurora MySQL. Setelah klaster DB Aurora MySQL dibuat, gunakan replikasi log biner untuk menyinkronkan klaster DB Aurora MySQL dengan transaksi yang terjadi setelah pencadangan. Saat klaster DB Aurora MySQL sinkron dengan basis data MySQL, selesaikan migrasi dengan sepenuhnya beralih ke klaster DB Aurora MySQL untuk

transaksi baru. Untuk informasi selengkapnya, lihat Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi.

Daftar Isi


- Pertimbangan dan batasan
- Sebelum Anda memulai
  - Instalasi Percona XtraBackup
  - Izin yang diperlukan
  - Membuat peran layanan IAM
- Mencadangkan file untuk dipulihkan sebagai klaster DB Amazon Aurora MySQL
  - Membuat backup lengkap dengan Percona XtraBackup
  - Menggunakan backup bertahap dengan Percona XtraBackup
  - Pertimbangan cadangan
- Memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3
- Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi
  - Mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi
  - Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal
- Mengurangi waktu migrasi fisik ke Amazon Aurora MySQL
  - Jenis tabel yang tidak didukung
  - Akun pengguna dengan hak akses yang tidak didukung
  - Hak akses dinamis di Aurora MySQL versi 3
  - Objek tersimpan dengan 'rdsadmin'@'localhost' sebagai pendefinisi


## Pertimbangan dan batasan

Batasan dan pertimbangan berikut berlaku untuk memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3:

- Anda dapat memigrasikan data Anda hanya ke klaster DB baru, bukan klaster DB yang sudah ada.
- Anda harus menggunakan Percona XtraBackup untuk mencadangkan data Anda ke S3. Untuk informasi selengkapnya, lihat Instalasi Percona XtraBackup.

- Bucket Amazon S3 dan cluster DB MySQL Aurora harus berada di Wilayah yang sama. AWS
- Anda tidak dapat memulihkan dari hal berikut:
  - Ekspor snapshot klaster DB ke Amazon S3. Anda juga tidak dapat memigrasikan data dari ekspor snapshot klaster DB ke bucket S3 Anda.
  - Basis data sumber terenkripsi, tetapi Anda dapat mengenkripsi data yang sedang dimigrasikan. Anda juga dapat membiarkan data tidak terenkripsi selama proses migrasi.
  - Basis data MySQL 5.5 atau 5.6
- Percona Server untuk MySQL tidak didukung sebagai database sumber, karena dapat berisi tabel dalam skema. `compression_dictionary*` `mysql`
- Anda tidak dapat memulihkan ke klaster DB Aurora Serverless.
- Migrasi mundur tidak didukung untuk versi mayor atau versi minor. Misalnya, Anda tidak dapat bermigrasi dari MySQL versi 8.0 ke Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7), dan Anda tidak dapat bermigrasi dari MySQL versi 8.0.32 ke Aurora MySQL versi 3.03, yang kompatibel dengan MySQL komunitas versi 8.0.26.
- Anda tidak dapat bermigrasi ke MySQL versi 3.05 dan yang lebih tinggi dari beberapa versi MySQL 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan agar Anda meningkatkan ke MySQL 8.0.28 sebelum migrasi.
- Pengimporan dari Amazon S3 tidak didukung di kelas instans DB db.t2.micro. Namun, Anda dapat memulihkan ke kelas instans DB yang berbeda, dan mengubah kelas instans DB di lain waktu. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB.
- Amazon S3 membatasi ukuran file yang diunggah ke bucket S3 hingga 5 TB. Jika sebuah file cadangan melebihi 5 TB, maka Anda harus membagi file cadangan ke dalam beberapa file yang lebih kecil.
- Amazon RDS membatasi jumlah file yang dapat diunggah ke sebuah bucket S3 hingga 1 juta. Jika data cadangan untuk basis data Anda, termasuk semua cadangan penuh dan inkremental, melebihi 1 juta file, gunakan sebuah file Gzip (.gz), tar (.tar.gz), atau Percona xbstream (.xbstream) untuk menyimpan file cadangan penuh dan inkremental dalam bucket Amazon S3. Percona XtraBackup 8.0 hanya mendukung Percona xbstream untuk kompresi.
- Untuk menyediakan layanan manajemen untuk setiap klaster DB, pengguna `rdsadmin` dibuat saat klaster DB dibuat. Karena ini adalah pengguna yang dicadangkan di RDS, batasan berikut berlaku:
  - Fungsi, prosedur, tampilan, peristiwa, dan pemicu dengan pendefinisi `'rdsadmin'@'localhost'` tidak diimpor. Untuk informasi selengkapnya, lihat Objek tersimpan dengan 'rdsadmin'@'localhost' sebagai pendefinisi dan Hak akses pengguna master Amazon Aurora MySQL.

  - Ketika klaster DB Aurora MySQL dibuat, pengguna master akan dibuat dengan hak akses paling tinggi yang didukung. Saat memulihkan dari cadangan, hak akses yang tidak didukung yang diberikan kepada pengguna yang diimpor akan dihapus secara otomatis selama impor.

    Untuk mengidentifikasi pengguna yang mungkin terpengaruh oleh hal ini, lihat Akun pengguna dengan hak akses yang tidak didukung. Untuk informasi selengkapnya tentang hak akses yang didukung di Aurora MySQL, lihat Model hak akses berbasis peran.

- Untuk Aurora MySQL versi 3, hak akses dinamis tidak diimpor. Hak akses dinamis yang didukung Aurora dapat diimpor setelah migrasi. Untuk informasi selengkapnya, lihat Hak akses dinamis di Aurora MySQL versi 3.
- Tabel yang dibuat pengguna dalam skema `mysql` tidak dimigrasikan.
- Parameter `innodb_data_file_path` harus dikonfigurasi dengan hanya satu file data yang menggunakan nama file data default `ibdata1:12M:autoextend`. Basis data dengan dua file data, atau dengan file data yang memiliki nama yang berbeda, tidak dapat dimigrasikan menggunakan metode ini.

  Berikut ini adalah contoh nama file yang tidak diizinkan: `innodb_data_file_path=ibdata1:50M`, `ibdata2:50M:autoextend` dan `innodb_data_file_path=ibdata01:50M:autoextend`.
- Anda tidak dapat bermigrasi dari basis data sumber yang memiliki tabel di luar direktori data MySQL default.
- Ukuran maksimum yang didukung untuk cadangan yang tidak terkompresi menggunakan metode ini saat ini terbatas pada 64 TiB. Untuk cadangan terkompresi, batas ini lebih rendah untuk memperhitungkan persyaratan ruang yang tidak terkompresi. Dalam kasus seperti itu, ukuran cadangan maksimum yang didukung adalah (`64 TiB – compressed backup size`).
- Aurora MySQL tidak mendukung pengimporan komponen dan plugin MySQL serta komponen dan plugin eksternal lainnya.
- Aurora MySQL tidak memulihkan semuanya dari basis data Anda. Kami menyarankan agar Anda menyimpan skema dan nilai basis data untuk item berikut dari basis data MySQL sumber Anda, lalu menambahkannya ke klaster DB Aurora MySQL yang dipulihkan setelah dibuat:
  - Akun pengguna
  - Fungsi
  - Prosedur tersimpan

- Informasi zona waktu. Informasi zona waktu dimuat dari sistem operasi lokal klaster DB Aurora MySQL Anda. Untuk informasi selengkapnya, lihat Zona waktu lokal untuk klaster DB Amazon Aurora.

### Sebelum Anda memulai

Sebelum Anda dapat menyalin data ke bucket Amazon S3 dan memulihkan klaster DB dari file tersebut, Anda harus melakukan hal berikut:

- Instal Percona XtraBackup di server lokal Anda.
- Izinkan Aurora MySQL untuk mengakses bucket Amazon S3 atas nama Anda.

### Instalasi Percona XtraBackup

Amazon Aurora dapat memulihkan cluster DB dari file yang dibuat menggunakan Percona. XtraBackup Anda dapat menginstal Percona XtraBackup dari Unduhan Perangkat Lunak - Percona.

Untuk migrasi MySQL 5.7, gunakan Percona 2.4. XtraBackup

Untuk migrasi MySQL 8.0, gunakan Percona 8.0. XtraBackup Pastikan bahwa versi Percona kompatibel dengan XtraBackup versi mesin database sumber Anda.

### Izin yang diperlukan

Untuk memigrasikan data MySQL Anda ke klaster DB Amazon Aurora MySQL, beberapa izin diperlukan:

- Pengguna yang meminta Aurora membuat cluster baru dari bucket Amazon S3 harus memiliki izin untuk membuat daftar bucket untuk akun Anda. AWS Anda memberikan izin ini kepada pengguna menggunakan kebijakan AWS Identity and Access Management (IAM).
- Aurora memerlukan izin untuk bertindak atas nama Anda untuk mengakses Amazon S3 bucket tempat Anda menyimpan file yang digunakan untuk membuat klaster DB Amazon Aurora MySQL Anda. Anda memberikan izin yang diperlukan Aurora menggunakan peran layanan IAM.
- Pengguna yang mengajukan permintaan juga harus memiliki izin untuk menampilkan daftar peran IAM untuk akun AWS Anda.
- Jika pengguna yang mengajukan permintaan tersebut bertujuan untuk membuat peran layanan IAM atau meminta Aurora membuat peran layanan IAM (dengan menggunakan konsol), pengguna tersebut harus memiliki izin untuk membuat peran IAM bagi akun AWS Anda.

- Jika Anda berencana untuk mengenkripsi data selama proses migrasi, perbarui kebijakan IAM pengguna yang akan melakukan migrasi untuk memberikan akses RDS ke yang AWS KMS keys digunakan untuk mengenkripsi cadangan. Untuk petunjuk, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS.

Misalnya, kebijakan IAM berikut memberikan izin minimum yang diperlukan pengguna untuk menggunakan konsol untuk menampilkan daftar peran IAM, membuat peran IAM, menampilkan daftar bucket Amazon S3 untuk akun Anda, dan menampilkan daftar kunci KMS.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

Selain itu, bagi pengguna yang ingin mengaitkan peran IAM dengan bucket Amazon S3, pengguna IAM harus memiliki izin `iam:PassRole` untuk peran IAM tersebut. Izin ini memungkinkan administrator membatasi peran IAM mana yang dapat dikaitkan dengan bucket Amazon S3.

Misalnya, kebijakan IAM berikut mengizinkan pengguna mengaitkan peran bernama `S3Access` dengan bucket Amazon S3.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

Untuk informasi selengkapnya tentang izin pengguna IAM, lihat Mengelola akses menggunakan kebijakan.

Membuat peran layanan IAM

Anda dapat AWS Management Console membuat peran untuk Anda dengan memilih opsi Buat Peran Baru (ditampilkan nanti dalam topik ini). Jika Anda memilih opsi ini dan menentukan nama untuk peran baru, Aurora akan membuat peran layanan IAM yang diperlukan Aurora untuk mengakses bucket Amazon S3 Anda dengan nama yang Anda berikan.

Sebagai alternatif, Anda dapat membuat peran secara manual menggunakan prosedur berikut.

Untuk membuat peran IAM bagi Aurora untuk mengakses Amazon S3

1. Selesaikan langkah-langkah dalam Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3.
2. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Selesaikan langkah-langkah dalam Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

Mencadangkan file untuk dipulihkan sebagai klaster DB Amazon Aurora MySQL

Anda dapat membuat cadangan lengkap file database MySQL Anda menggunakan XtraBackup Percona dan mengunggah file cadangan ke bucket Amazon S3. Atau, jika Anda sudah menggunakan Percona XtraBackup untuk mencadangkan file database MySQL Anda, Anda dapat mengunggah direktori dan file cadangan lengkap dan tambahan yang ada ke bucket Amazon S3.

Topik


- Membuat backup lengkap dengan Percona XtraBackup
- Menggunakan backup bertahap dengan Percona XtraBackup
- Pertimbangan cadangan


Membuat backup lengkap dengan Percona XtraBackup

Untuk membuat cadangan lengkap file database MySQL Anda yang dapat dipulihkan dari Amazon S3 untuk membuat cluster DB MySQL Aurora, gunakan utilitas Percona () untuk mencadangkan database Anda. XtraBackup `xtrabackup`

Misalnya, perintah berikut akan membuat cadangan dari basis data MySQL dan menyimpan file-nya dalam folder `/on-premises/s3-restore/backup`.

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/
s3-restore/backup>
```

Jika Anda ingin mengompresi cadangan Anda ke dalam satu file (yang dapat dibagi, jika perlu), Anda dapat menggunakan opsi `--stream` untuk menyimpan cadangan Anda dalam salah satu format berikut:

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

Perintah berikut membuat cadangan basis data MySQL Anda yang dibagi menjadi beberapa file Gzip.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

Perintah berikut membuat cadangan basis data MySQL Anda yang dipecah menjadi beberapa file tar.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

Perintah berikut membuat cadangan basis data MySQL Anda yang dipecah menjadi beberapa file xbstream.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> Jika Anda melihat kesalahan berikut, ini mungkin disebabkan perbedaan format file dalam perintah Anda:
>
> ```
> ERROR:/bin/tar: This does not look like a tar archive
> ```

Setelah Anda mencadangkan database MySQL Anda menggunakan utilitas XtraBackup Percona, Anda dapat menyalin direktori dan file cadangan Anda ke bucket Amazon S3.

Untuk informasi tentang membuat dan mengunggah file ke bucket Amazon S3, lihat Memulai Amazon Simple Storage Service dalam Panduan Memulai Amazon S3.

Menggunakan backup bertahap dengan Percona XtraBackup

Amazon Aurora MySQL mendukung cadangan penuh dan tambahan yang dibuat menggunakan Percona. XtraBackup Jika Anda sudah menggunakan Percona XtraBackup untuk melakukan pencadangan penuh dan bertahap dari file database MySQL Anda, Anda tidak perlu membuat cadangan lengkap dan mengunggah file cadangan ke Amazon S3. Sebagai gantinya, Anda dapat menghemat cukup banyak waktu dengan menyalin direktori dan file cadangan yang ada untuk cadangan penuh dan inkremental ke bucket Amazon S3. Untuk informasi selengkapnya, lihat Buat cadangan inkremental di situs web Percona.

Saat menyalin file cadangan penuh dan inkremental ke bucket Amazon S3, Anda harus menyalin konten direktori dasar secara berulang. Konten tersebut termasuk cadangan penuh dan juga semua

direktori dan file cadangan inkremental. Salinan ini harus mempertahankan struktur direktori di bucket Amazon S3. Aurora melakukan iterasi ke semua file dan direktori. Aurora menggunakan file `xtrabackup-checkpoints` yang disertakan dengan setiap cadangan inkremental untuk mengidentifikasi direktori dasar dan untuk mengurutkan cadangan inkremental berdasarkan rentang nomor urutan log (LSN).

Untuk informasi tentang membuat dan mengunggah file ke bucket Amazon S3, lihat Memulai Amazon Simple Storage Service dalam Panduan Memulai Amazon S3.

Pertimbangan cadangan

Aurora tidak mendukung cadangan sebagian yang dibuat menggunakan Percona. XtraBackup Anda tidak dapat menggunakan opsi berikut untuk membuat cadangan parsial saat mencadangkan file sumber untuk basis data Anda: `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude`, atau `--databases-file`.

Untuk informasi selengkapnya tentang mencadangkan database Anda dengan Percona XtraBackup, lihat Percona XtraBackup - Dokumentasi dan Bekerja dengan log biner di situs web Percona.

Aurora mendukung backup tambahan yang dibuat menggunakan Percona. XtraBackup Untuk informasi selengkapnya, lihat Buat cadangan inkremental di situs web Percona.

Aurora menggunakan file cadangan Anda berdasarkan nama file. Pastikan untuk menamai file cadangan Anda dengan ekstensi file yang sesuai berdasarkan format file—misalnya, `.xbstream` untuk file yang disimpan menggunakan format Percona xbstream.

Aurora menggunakan file cadangan Anda dalam urutan abjad dan juga dalam urutan angka alami. Selalu gunakan opsi `split` saat Anda mengeluarkan perintah `xtrabackup` untuk memastikan bahwa file cadangan Anda ditulis dan diberi nama sesuai urutan yang benar.

Amazon S3 membatasi ukuran file yang diunggah ke bucket Amazon S3 hingga 5 TB. Jika data cadangan untuk basis data Anda melebihi 5 TB, gunakan perintah `split` untuk membagi file cadangan menjadi beberapa file dengan ukuran masing-masing kurang dari 5 TB.

Aurora membatasi jumlah file sumber yang diunggah ke bucket Amazon S3 hingga 1 juta file. Dalam beberapa kasus, data cadangan untuk basis data Anda, termasuk semua cadangan penuh dan inkremental, dapat menghasilkan jumlah file yang besar. Dalam hal ini, gunakan file tarball (.tar.gz) untuk menyimpan file cadangan penuh dan inkremental di bucket Amazon S3.

Saat Anda mengunggah file ke bucket Amazon S3, Anda dapat menggunakan enkripsi di sisi server untuk mengenkripsi data. Anda kemudian dapat memulihkan klaster DB Amazon Aurora MySQL dari

file terenkripsi tersebut. Amazon Aurora MySQL dapat memulihkan klaster DB dengan file terenkripsi menggunakan jenis enkripsi di sisi server berikut:

- Enkripsi di sisi server dengan kunci yang dikelola Amazon S3 (SSE-S3) – Setiap objek dienkripsi dengan kunci unik yang menggunakan enkripsi multifaktor yang kuat.
- Enkripsi sisi server dengan kunci yang AWS KMS dikelola (SSE-KMS) — Mirip dengan SSE-S3, tetapi Anda memiliki opsi untuk membuat dan mengelola kunci enkripsi sendiri, dan juga perbedaan lainnya.

Untuk informasi tentang menggunakan enkripsi di sisi server saat mengunggah file ke bucket Amazon S3, lihat Melindungi data menggunakan enkripsi di sisi server dalam Panduan Developer Amazon S3.

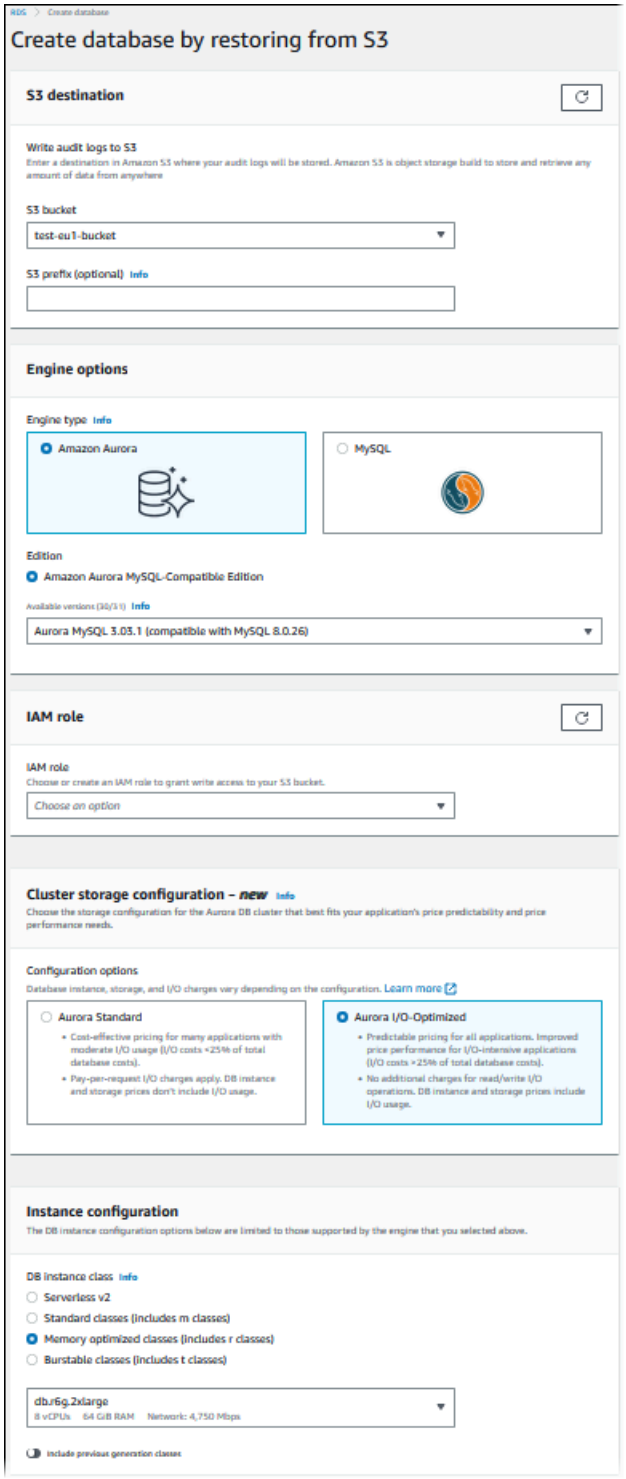
Memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3

Anda dapat memulihkan file cadangan dari bucket Amazon S3 untuk membuat klaster DB Amazon Aurora MySQL baru dengan menggunakan konsol Amazon RDS.

Untuk memulihkan klaster DB Amazon Aurora MySQL dari file di bucket Amazon S3

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas konsol Amazon RDS, pilih AWS Wilayah tempat membuat cluster DB Anda. Pilih AWS Wilayah yang sama dengan bucket Amazon S3 yang berisi cadangan database Anda.
3. Di panel navigasi, pilih Basis data, lalu pilih Pulihkan dari S3.
4. Pilih Pulihkan dari S3.

   Halaman Buat basis data dengan memulihkan dari S3 akan muncul.




5. Di bagian Tujuan S3:

   a. Pilih Bucket S3 yang berisi file cadangan.

   b. (Opsional) Untuk Awalan jalur folder S3, masukkan awalan jalur file untuk file yang disimpan di bucket Amazon S3 Anda.

Jika Anda tidak menentukan prefiks, RDS akan membuat instans DB Anda menggunakan semua file dan folder di folder root bucket S3. Jika Anda menentukan prefiks, RDS akan membuat instans DB Anda menggunakan file dan folder dalam bucket S3 yang jalur file-nya dimulai dengan prefiks yang ditentukan.

Misalnya, Anda menyimpan file cadangan di S3 dalam subfolder bernama “cadangan”, dan Anda memiliki beberapa set file cadangan, masing-masing memiliki direktori tersendiri (gzip_backup1, gzip_backup2, dan sebagainya). Dalam kasus ini, Anda menentukan prefiks backup/gzip_backup1 untuk memulihkan dari file di folder gzip_backup1.

6. Di bagian Opsi mesin:

   a. Untuk Tipe mesin, pilih Amazon Aurora.

   b. Untuk Versi, pilih versi mesin Aurora MySQL untuk instans DB Anda yang dipulihkan.

7. Untuk Peran IAM, Anda dapat memilih peran IAM yang sudah ada.

8. (Opsional) Anda juga dapat dibuatkan peran IAM dengan memilih Buat peran baru. Jika demikian:

   a. Masukkan Nama peran IAM.

   b. Pilih opsi untuk Izinkan akses ke kunci KMS:

      - Jika Anda tidak mengenkripsi file cadangan, pilih Tidak.
      - Jika Anda mengenkripsi file cadangan dengan AES-256 (SSE-S3) saat mengunggahnya ke Amazon S3, pilih Tidak. Dalam kasus ini, data didekripsi secara otomatis.
      - Jika Anda mengenkripsi file cadangan dengan enkripsi sisi server AWS KMS (SSE-KMS) saat Anda mengunggahnya ke Amazon S3, pilih Ya. Selanjutnya, pilih kunci KMS yang benar untuk AWS KMS key.

        AWS Management Console Membuat kebijakan IAM yang memungkinkan Aurora untuk mendekripsi data.

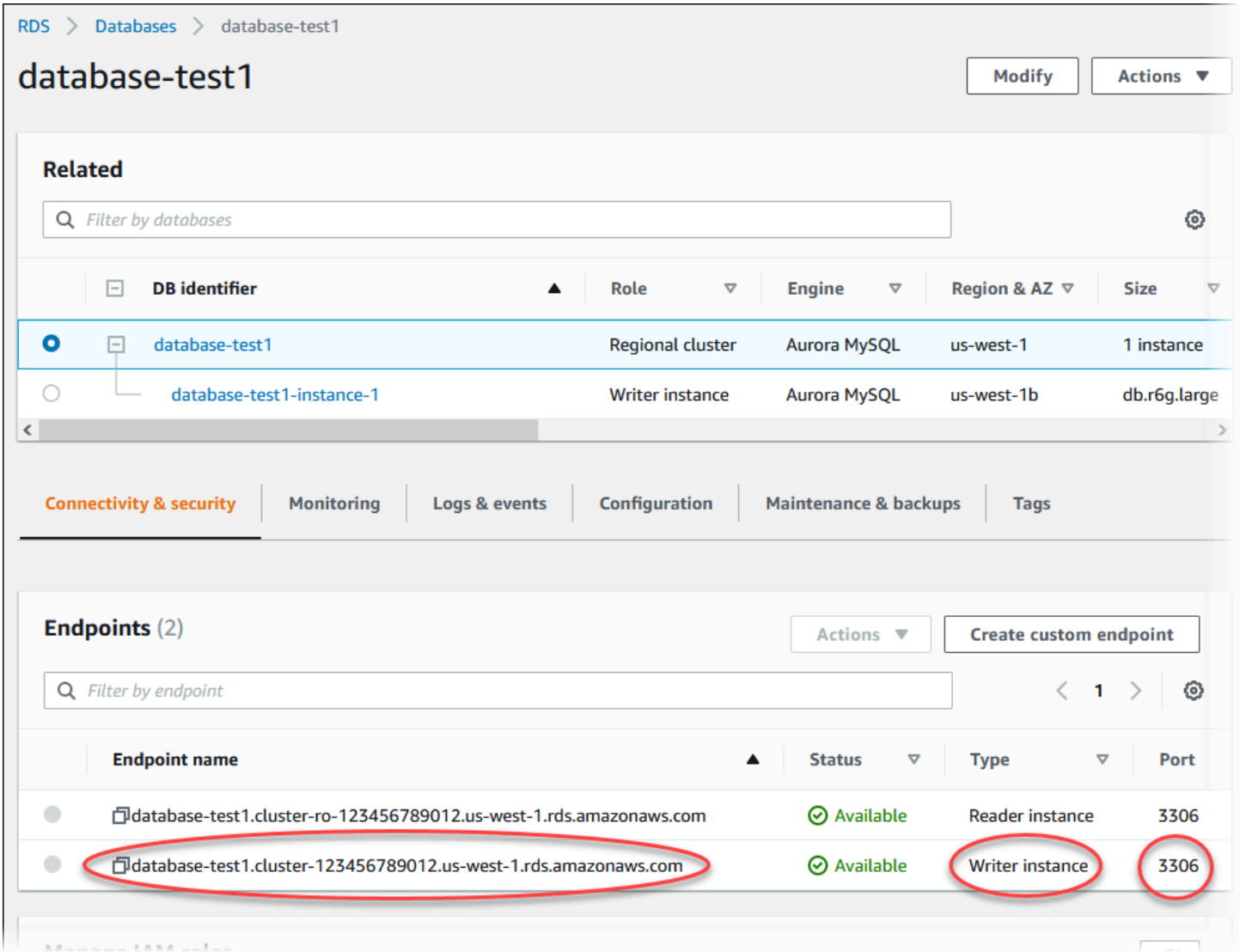
   Untuk informasi selengkapnya, lihat Melindungi data menggunakan enkripsi di sisi server dalam Panduan Developer Amazon S3.

9. Pilih pengaturan untuk klaster DB Anda, seperti konfigurasi penyimpanan klaster DB, kelas instans DB, pengidentifikasi klaster DB, dan kredensial login. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

10. Sesuaikan pengaturan tambahan untuk klaster DB Aurora MySQL Anda berdasarkan kebutuhan.
11. Pilih Buat basis data untuk meluncurkan instans DB Aurora Anda.

Pada konsol Amazon RDS, instans DB baru muncul dalam daftar instans DB. Instans DB memiliki status membuat hingga instans DB dibuat dan siap digunakan. Saat statusnya berubah menjadi tersedia, Anda dapat terhubung ke instans primer untuk klaster DB Anda. Bergantung pada kelas instans DB dan penyimpanan yang dialokasikan, perlu waktu beberapa menit agar instans baru tersedia.

Untuk melihat klaster yang baru dibuat, pilih tampilan Basis data di konsol Amazon RDS lalu pilih klaster DB. Untuk informasi selengkapnya, lihat Melihat klaster DB Amazon Aurora.



Catat port dan titik akhir penulis dari klaster DB. Gunakan titik akhir dan port klaster DB tersebut dalam string koneksi JDBC dan ODBC Anda untuk aplikasi apa pun yang menjalankan operasi tulis atau baca.

## Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi

Untuk mempersingkat atau meniadakan waktu henti selama migrasi, Anda dapat mereplikasi transaksi yang dilakukan pada basis data MySQL Anda ke klaster DB Aurora MySQL Anda. Replikasi memungkinkan klaster DB disinkronkan dengan transaksi di basis data MySQL yang terjadi selama migrasi. Ketika klaster DB telah sepenuhnya sinkron, Anda dapat menghentikan replikasi dan menyelesaikan migrasi ke Aurora MySQL.

Topik


- Mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi
- Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal


### Mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi

Untuk mereplikasi data dengan aman, Anda dapat menggunakan replikasi terenkripsi.

> **Note**
>
> Jika Anda tidak perlu menggunakan replikasi terenkripsi, Anda dapat melewatkan langkah-langkah ini dan melanjutkan ke petunjuk dalam Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal.

Berikut ini adalah prasyarat untuk menggunakan replikasi terenkripsi:

- Lapisan Soket Aman (SSL) harus diaktifkan di basis data primer MySQL eksternal.
- Kunci klien dan sertifikat klien harus disiapkan untuk klaster DB Aurora MySQL.

Selama replikasi terenkripsi, klaster DB Aurora MySQL berperan sebagai klien untuk server basis data MySQL. Sertifikat dan kunci untuk klien Aurora MySQL ada di dalam file dengan format .pem.

Untuk mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi

1. Pastikan bahwa Anda siap untuk replikasi terenkripsi:

   - Jika Anda tidak mengaktifkan SSL di basis data primer MySQL eksternal dan tidak menyiapkan kunci klien dan sertifikat klien, aktifkan SSL di server basis data MySQL dan buat kunci klien dan sertifikat klien yang diperlukan.
   - Jika SSL diaktifkan pada primer eksternal, berikan kunci klien dan sertifikat untuk klaster DB Aurora MySQL. Jika Anda tidak memilikinya, buat kunci dan sertifikat baru untuk klaster DB Aurora MySQL. Untuk menandatangani sertifikat klien, Anda harus memiliki kunci otoritas sertifikat yang Anda gunakan untuk mengonfigurasi SSL di basis data primer MySQL eksternal.

   Untuk informasi selengkapnya, lihat Creating SSL certificates and keys using openssl dalam dokumentasi MySQL.

   Anda memerlukan sertifikat otoritas sertifikat, kunci klien, dan sertifikat klien.
2. Hubungkan ke klaster DB Aurora MySQL sebagai pengguna primer menggunakan SSL.

   Untuk informasi tentang menghubungkan ke klaster DB Aurora MySQL dengan SSL, lihat Koneksi TLS ke cluster DB MySQL Aurora.
3. Jalankan prosedur tersimpan mysql.rds_import_binlog_ssl_material untuk mengimpor informasi SSL ke dalam klaster DB Aurora MySQL.

   Untuk parameter `ssl_material_value`, masukkan informasi dari file format .pem untuk klaster DB Aurora MySQL di payload JSON yang benar.

   Contoh berikut mengimpor informasi SSL ke dalam klaster DB Aurora MySQL. Dalam file berformat .pem, kode konten biasanya lebih panjang dari kode konten yang ditunjukkan dalam contoh.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
```

```
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

Untuk informasi selengkapnya, lihat mysql.rds_import_binlog_ssl_material dan Koneksi TLS ke cluster DB MySQL Aurora.

> **Note**
>
> Setelah menjalankan prosedur ini, rahasia disimpan dalam file. Untuk menghapus file nanti, Anda dapat menjalankan prosedur tersimpan mysql.rds_remove_binlog_ssl_material.

Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal

Anda dapat menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi.

Untuk menyinkronkan klaster DB Aurora MySQL Anda dengan basis data MySQL menggunakan replikasi

1. Pastikan file /etc/my.cnf untuk basis data MySQL eksternal memiliki entri yang relevan.

   Jika replikasi terenkripsi tidak diperlukan, pastikan basis data MySQL eksternal dimulai dengan log biner (binlog) diaktifkan dan SSL dinonaktifkan. Berikut ini adalah entri yang relevan dalam file /etc/my.cnf untuk data yang tidak terenkripsi.

   ```
   log-bin=mysql-bin
   server-id=2133421
   ```

```
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

Jika replikasi terenkripsi diperlukan, pastikan basis data MySQL eksternal dimulai dengan SSL dan binlog diaktifkan. Entri dalam file /etc/my.cnf mencakup lokasi file .pem untuk server basis data MySQL.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

Anda dapat memverifikasi bahwa SSL diaktifkan dengan perintah berikut.

```
mysql> show variables like 'have_ssl';
```

Output Anda harus serupa dengan berikut ini.

```
+~-~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~---+
| Variable_name | Value |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~---+
| have_ssl      | YES   |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~---+
1 row in set (0.00 sec)
```

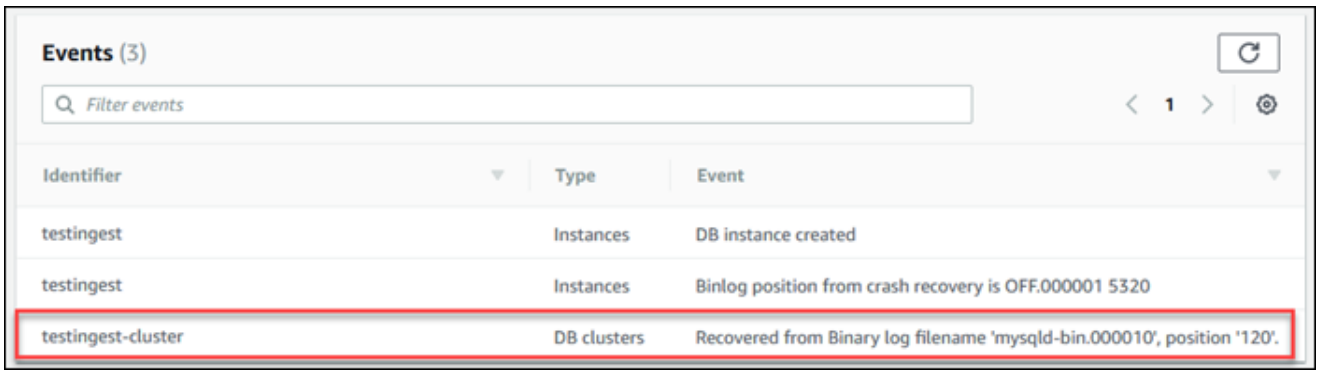
2. Tentukan posisi log biner awal untuk replikasi. Anda menentukan posisi untuk memulai replikasi di langkah selanjutnya.

   Menggunakan AWS Management Console

   a. Masuk ke AWS Management Console dan buka konsol Amazon RDS di [https://console.aws.amazon.com/rds/](https://console.aws.amazon.com/rds/).

b. Di panel navigasi, pilih Peristiwa.

c. Di daftar Peristiwa, catat posisi di peristiwa Dipulihkan dari nama file log biner.




Menggunakan AWS CLI

Anda juga bisa mendapatkan nama dan posisi file binlog dengan menggunakan perintah AWS CLI describe-events. Hal berikut menunjukkan contoh perintah `describe-events`.

```
PROMPT> aws rds describe-events
```

Pada output, identifikasi peristiwa yang menunjukkan posisi binlog.

3. Saat terhubung dengan basis data MySQL eksternal, buat pengguna yang akan digunakan untuk replikasi. Akun ini digunakan hanya untuk replikasi dan harus dibatasi pada domain Anda untuk meningkatkan keamanan. Berikut adalah contohnya.

```
mysql> CREATE USER '<user_name>'@'<domain_name>' IDENTIFIED BY '<password>';
```

Pengguna memerlukan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE`. Berikan hak akses ini kepada pengguna.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

Jika Anda perlu menggunakan replikasi terenkripsi, wajibkan koneksi SSL untuk pengguna replikasi. Misalnya, Anda dapat menggunakan pernyataan berikut untuk mewajibkan koneksi SSL pada akun pengguna *<user_name>*.

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> Jika `REQUIRE SSL` tidak disertakan, koneksi replikasi dapat kembali ke koneksi yang tidak terenkripsi tanpa peringatan.

4. Di konsol Amazon RDS, tambahkan alamat IP server yang meng-host basis data MySQL eksternal ke grup keamanan VPC untuk klaster DB Aurora MySQL. Untuk informasi selengkapnya tentang memodifikasi grup keamanan VPC, lihat Grup keamanan untuk VPC Anda dalam Panduan Pengguna Amazon Virtual Private Cloud.

   Anda mungkin juga perlu mengonfigurasi jaringan lokal Anda untuk mengizinkan koneksi dari alamat IP klaster DB Aurora MySQL Anda agar klaster DB ini dapat berkomunikasi dengan instans MySQL eksternal Anda. Untuk menemukan alamat IP klaster DB Aurora MySQL, gunakan perintah `host`.

   ```
   host <db_cluster_endpoint>
   ```

   Nama host adalah nama DNS dari titik akhir klaster DB Aurora MySQL.

5. Aktifkan replikasi log biner dengan menjalankan prosedur tersimpan mysql.rds_reset_external_master ) RDS SQL SQL atau mysql.rds_reset_external_source ) SQL SQL. Prosedur tersimpan ini memiliki sintaksis berikut.

   ```
   CALL mysql.rds_set_external_master (
     host_name
     , host_port
     , replication_user_name
     , replication_user_password
     , mysql_binary_log_file_name
     , mysql_binary_log_file_location
     , ssl_encryption
   );

   CALL mysql.rds_set_external_source (
     host_name
   ```

```
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Untuk informasi tentang parameter, lihat mysql.rds_reset_external_master ) RDS SQL SQL dan mysql.rds_reset_external_source ) SQL SQL.

Untuk `mysql_binary_log_file_name` dan `mysql_binary_log_file_location`, gunakan posisi dalam peristiwa Dipulihkan dari nama file log biner yang Anda catat sebelumnya.

Jika data dalam klaster DB Aurora MySQL tidak dienkripsi, parameter `ssl_encryption` harus diatur ke `0`. Jika data dienkripsi, parameter `ssl_encryption` harus diatur ke `1`.

Contoh berikut menjalankan prosedur untuk klaster DB Aurora MySQL yang memiliki data terenkripsi.

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);
```

Prosedur tersimpan ini menetapkan parameter yang digunakan klaster DB Aurora MySQL untuk menghubungkan ke basis data MySQL eksternal dan membaca log binernya. Jika data

dienkripsi, prosedur ini juga akan mengunduh sertifikat otoritas sertifikat SSL, sertifikat klien, dan kunci klien ke disk lokal.

6. Mulai replikasi log biner dengan menjalankan prosedur tersimpan mysql.rds_start_replication.

```
CALL mysql.rds_start_replication;
```

7. Pantau seberapa jauh klaster DB Aurora MySQL tertinggal dari basis data primer replikasi MySQL. Untuk melakukannya, hubungkan ke klaster DB Aurora MySQL dan jalankan perintah berikut.

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

   Dalam output perintah, bidang `Seconds Behind Master` menunjukkan seberapa jauh klaster DB Aurora MySQL tertinggal dari basis data primer MySQL. Ketika nilai ini `0` (nol), klaster DB Aurora MySQL telah sinkron dengan basis data primer, dan Anda dapat melanjutkan ke langkah berikutnya untuk menghentikan replikasi.

8. Hubungkan ke basis data primer replikasi MySQL dan hentikan replikasi. Untuk melakukannya, jalankan prosedur tersimpan mysql.rds_stop_replication.

```
CALL mysql.rds_stop_replication;
```

### Mengurangi waktu migrasi fisik ke Amazon Aurora MySQL

Anda dapat membuat modifikasi basis data berikut untuk mempercepat proses migrasi basis data ke Amazon Aurora MySQL.

> **Important**
>
> Pastikan untuk melakukan pembaruan ini pada salinan basis data produksi, bukan pada basis data produksi. Kemudian, Anda dapat mencadangkan salinan ini dan memulihkannya

ke klaster DB Aurora MySQL untuk menghindari gangguan layanan apa pun pada basis data produksi Anda.

Jenis tabel yang tidak didukung

Aurora MySQL hanya mendukung mesin InnoDB untuk tabel basis data. Jika Anda memiliki tabel MyISAM di basis data Anda, tabel tersebut harus dikonversi sebelum bermigrasi ke Aurora MySQL. Proses konversi memerlukan ruang tambahan untuk konversi MyISAM ke InnoDB selama prosedur migrasi.

Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, konversikan semua tabel MyISAM Anda ke tabel InnoDB sebelum memigrasikannya. Ukuran tabel InnoDB yang dihasilkan akan setara dengan ukuran yang diperlukan oleh Aurora MySQL untuk tabel tersebut. Untuk mengonversi tabel MyISAM ke InnoDB, jalankan perintah berikut:

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL tidak mendukung tabel atau halaman terkompresi, yaitu tabel yang dibuat dengan `ROW_FORMAT=COMPRESSED` atau `COMPRESSION = {"zlib"|"lz4"}`.

Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, perluas tabel terkompresi Anda dengan mengatur `ROW_FORMAT` ke `DEFAULT`, `COMPACT`, `DYNAMIC`, atau `REDUNDANT`. Untuk halaman terkompresi, atur `COMPRESSION="none"`.

Untuk informasi selengkapnya, lihat InnoDB row formats dan InnoDB table and page compression dalam dokumentasi MySQL.

Anda dapat menggunakan skrip SQL berikut pada instans DB MySQL yang ada untuk menampilkan daftar tabel dalam basis data Anda yang merupakan tabel MyISAM atau tabel terkompresi.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
```

```
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

Akun pengguna dengan hak akses yang tidak didukung

Akun pengguna dengan hak akses yang tidak didukung oleh Aurora MySQL akan diimpor tanpa hak akses yang tidak didukung. Untuk daftar hak akses yang didukung, lihat Model hak akses berbasis peran.

Anda dapat menjalankan kueri SQL berikut pada basis data sumber Anda untuk menampilkan daftar akun pengguna yang memiliki hak akses yang tidak didukung.

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Hak akses dinamis di Aurora MySQL versi 3

Hak akses dinamis tidak akan diimpor. Aurora MySQL versi 3 mendukung hak akses dinamis berikut.

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

Contoh skrip berikut memberikan hak akses dinamis yang didukung ke akun pengguna di klaster DB Aurora MySQL.

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
```

```
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

Objek tersimpan dengan 'rdsadmin'@'localhost' sebagai pendefinisi

Fungsi, prosedur, tampilan, peristiwa, dan pemicu dengan `'rdsadmin'@'localhost'` sebagai pendefinisi tidak akan diimpor.

Anda dapat menggunakan skrip SQL berikut pada basis data MySQL sumber Anda untuk menampilkan daftar objek tersimpan yang memiliki pendefinisi yang tidak didukung.

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';
```

```
-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

## Migrasi logis dari My SQL ke Amazon Aurora My SQL dengan menggunakan mysqldump

Karena Amazon Aurora My SQL adalah database yang SQL kompatibel dengan Saya, Anda dapat menggunakan `mysqldump` utilitas untuk menyalin data dari database Saya atau SQL MariaDB ke cluster Aurora My DB yang ada. SQL

Untuk diskusi tentang cara melakukannya dengan SQL database Saya yang sangat besar, lihat Mengimpor data ke instans DB Saya atau SQL MariaDB dengan waktu henti yang dikurangi. Untuk SQL database Saya yang memiliki jumlah data yang lebih kecil, lihat Mengimpor data dari DB Saya atau SQL MariaDB ke instans DB Saya atau MariaDB. SQL

# Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL

Anda dapat memigrasikan (menyalin) data ke klaster DB Amazon Aurora MySQL dari instans DB RDS for MySQL.

Topik

- Memigrasikan snapshot RDS for MySQL ke Aurora
- Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora

> **Note**
>
> Karena Amazon Aurora MySQL kompatibel dengan MySQL, Anda dapat memigrasikan data dari basis data MySQL dengan mengatur replikasi antara basis data MySQL Anda dan klaster DB Amazon Aurora MySQL. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

## Memigrasikan snapshot RDS for MySQL ke Aurora

Anda dapat memigrasikan snapshot DB dari instans DB RDS for MySQL untuk membuat klaster DB Aurora MySQL. Klaster DB Aurora MySQL yang baru akan diisi dengan data dari instans DB RDS for MySQL asli. Snapshot DB ini harus dibuat dari instans DB Amazon RDS yang menjalankan versi MySQL yang kompatibel dengan Aurora MySQL.

Anda dapat memigrasikan snapshot DB manual atau otomatis. Setelah klaster DB dibuat, Anda dapat membuat Replika Aurora opsional.

> **Note**
>
> Anda juga dapat memigrasikan instans DB RDS for MySQL ke klaster DB Aurora MySQL dengan membuat replika baca Aurora dari instans DB RDS for MySQL sumber Anda. Untuk informasi selengkapnya, lihat Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora.

Anda tidak dapat bermigrasi ke MySQL versi 3.05 dan yang lebih tinggi dari beberapa versi MySQL 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan agar Anda meningkatkan ke MySQL 8.0.28 sebelum migrasi.

Langkah-langkah umum yang harus Anda ambil adalah sebagai berikut:

1. Tentukan jumlah ruang yang perlu disediakan untuk klaster DB Aurora MySQL Anda. Untuk informasi selengkapnya, silakan lihat Berapa banyak ruang yang saya butuhkan?
2. Gunakan konsol untuk membuat snapshot di AWS Wilayah tempat instance Amazon RDS MySQL berada. Untuk informasi tentang membuat snapshot DB, lihat Membuat snapshot DB.
3. Jika snapshot DB tidak berada di AWS Wilayah yang sama dengan cluster DB Anda, gunakan konsol Amazon RDS untuk menyalin snapshot DB ke Wilayah tersebut. AWS Untuk informasi tentang menyalin snapshot DB, lihat Menyalin snapshot DB.
4. Gunakan konsol untuk memigrasikan snapshot DB dan membuat klaster DB Aurora MySQL dengan basis data yang sama dengan instans DB MySQL asli.

⚠ Warning

Amazon RDS membatasi setiap AWS akun menjadi satu salinan snapshot ke setiap AWS Wilayah sekaligus.

Berapa banyak ruang yang saya butuhkan?

Saat Anda memigrasikan snapshot dari instans DB MySQL ke dalam klaster DB Aurora MySQL, Aurora akan menggunakan volume Amazon Elastic Block Store (Amazon EBS) untuk memformat data dari snapshot ini sebelum memigrasikannya. Dalam beberapa kasus, ruang tambahan diperlukan untuk memformat data untuk migrasi.

Tabel yang bukan berupa tabel MyISAM dan tidak dikompresi dapat berukuran hingga 16 TB. Jika Anda memiliki tabel MyISAM, Aurora harus menggunakan ruang tambahan dalam volume untuk mengonversi tabel ini agar kompatibel dengan Aurora MySQL. Jika Anda memiliki tabel yang dikompresi, Aurora harus menggunakan ruang tambahan dalam volume untuk memperluas tabel tersebut sebelum menyimpannya di volume klaster Aurora. Karena persyaratan ruang tambahan ini, Anda harus memastikan bahwa tidak ada tabel MyISAM dan tabel terkompresi yang dimigrasikan dari instans DB MySQL Anda yang berukuran lebih dari 8 TB.

### Mengurangi jumlah ruang yang diperlukan untuk memigrasikan data ke Amazon Aurora MySQL

Anda sebaiknya memodifikasi skema basis data Anda sebelum memigrasikannya ke Amazon Aurora. Modifikasi tersebut dapat membantu dalam kasus berikut:

- Anda ingin mempercepat proses migrasi.
- Anda tidak yakin berapa banyak ruang yang perlu disediakan.
- Anda telah mencoba memigrasikan data Anda dan migrasi gagal karena kurangnya ruang terprovisi.

Anda dapat membuat perubahan berikut untuk meningkatkan proses migrasi basis data ke Amazon Aurora.

> **Important**
>
> Pastikan untuk melakukan pembaruan ini pada instans DB baru yang dipulihkan dari snapshot basis data produksi, alih-alih pada instans produksi. Kemudian, Anda dapat memigrasikan data dari snapshot instans DB baru Anda ke dalam klaster DB Aurora untuk menghindari gangguan layanan apa pun pada basis data produksi Anda.

| Jenis tabel | Batasan atau pedoman |
| --- | --- |
| Tabel MyISAM | Aurora MySQL hanya mendukung tabel InnoDB. Jika Anda memiliki tabel MyISAM di basis data Anda, tabel tersebut harus dikonversi sebelum dimigrasikan ke Aurora MySQL. Proses konversi memerlukan ruang tambahan untuk konversi MyISAM ke InnoDB selama prosedur migrasi.<br><br>Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, konversikan semua tabel MyISAM Anda ke tabel InnoDB sebelum memigrasikannya. Ukuran tabel InnoDB yang dihasilkan akan setara dengan ukuran yang diperlukan oleh Aurora MySQL untuk tabel tersebut. Untuk mengonversi tabel MyISAM ke InnoDB, jalankan perintah berikut: |

| Jenis tabel | Batasan atau pedoman |
| --- | --- |
| | `alter table <schema>.<table_name> engine=innodb, algorithm=copy;` |
| Tabel terkompresi | Aurora MySQL tidak mendukung tabel terkompresi (yaitu, tabel yang dibuat dengan `ROW_FORMAT=COMPRESSED`). Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, perluas tabel terkompresi Anda dengan mengatur `ROW_FORMAT` ke `DEFAULT`, `COMPACT`, `DYNAMIC`, atau `REDUNDANT`. Untuk informasi selengkapnya, lihat InnoDB row formats dalam dokumentasi MySQL. |

Anda dapat menggunakan skrip SQL berikut pada instans DB MySQL yang ada untuk menampilkan daftar tabel dalam basis data Anda yang merupakan tabel MyISAM atau tabel terkompresi.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
```

```
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

Skrip ini menghasilkan output yang mirip dengan output dalam contoh berikut. Contoh ini menunjukkan dua tabel yang harus dikonversi dari MyISAM ke InnoDB. Output-nya juga berisi perkiraan ukuran setiap tabel dalam megabyte (MB).

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

### Memigrasikan snapshot DB RDS for MySQL ke klaster DB Aurora MySQL

Anda dapat memigrasikan snapshot DB dari instans RDS untuk MySQL DB untuk membuat cluster DB MySQL Aurora menggunakan atau. AWS Management Console AWS CLI Klaster DB Aurora MySQL yang baru akan diisi dengan data dari instans DB RDS for MySQL asli. Untuk informasi tentang membuat snapshot DB, lihat Membuat snapshot DB.

Jika snapshot DB tidak berada di AWS Wilayah tempat Anda ingin menemukan data Anda, salin snapshot DB ke Wilayah itu AWS . Untuk informasi tentang menyalin snapshot DB, lihat Menyalin snapshot DB.

Konsol

Saat Anda memigrasikan snapshot DB dengan menggunakan AWS Management Console, konsol mengambil tindakan yang diperlukan untuk hanya membuat cluster DB.

Anda juga dapat memilih klaster DB Aurora MySQL baru Anda untuk dienkripsi saat diam menggunakan AWS KMS key.

Untuk memigrasikan snapshot MySQL DB dengan menggunakan AWS Management Console

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Mulai migrasi dari instans DB MySQL atau dari snapshot:

   Untuk memulai migrasi dari instans DB:

   1. Di panel navigasi, pilih Basis data, lalu pilih instans DB MySQL.
   2. Untuk Tindakan, pilih Migrasikan snapshot terbaru.

   Untuk memulai migrasi dari snapshot:

   1. Pilih Snapshot.
   2. Di Snapshot, pilih snapshot yang ingin Anda migrasikan ke klaster DB Aurora MySQL.
   3. Pilih Tindakan Snapshot, lalu pilih Migrasikan Snapshot.

   Halaman Migrasikan Basis Data akan muncul.
3. Tetapkan nilai-nilai berikut pada halaman Migrasikan Basis Data:

- Migrasikan ke Mesin DB: Pilih `aurora`.
- Versi Mesin DB: Pilih versi mesin DB untuk klaster DB Aurora MySQL.
- Kelas Instans DB: Pilih kelas instans DB yang memiliki penyimpanan dan kapasitas yang diperlukan untuk basis data Anda, misalnya `db.r3.large`. Volume klaster Aurora secara otomatis bertambah seiring peningkatan jumlah data dalam basis data Anda. Volume klaster Aurora dapat bertambah hingga ukuran maksimum 128 tebibyte (TiB). Jadi, Anda hanya perlu memilih kelas instans DB yang memenuhi persyaratan penyimpanan Anda saat ini. Untuk informasi selengkapnya, lihat Gambaran umum penyimpanan Amazon Aurora.
- Pengenal Instans DB: Ketik nama untuk cluster DB yang unik untuk akun Anda di AWS Wilayah yang Anda pilih. Pengidentifikasi ini digunakan di alamat titik akhir untuk instans di klaster DB Anda. Anda dapat memilih untuk menambahkan beberapa kecerdasan ke nama, seperti menyertakan AWS Region dan mesin DB yang Anda pilih, misalnya**`aurora-cluster1`**.

  Pengidentifikasi instans DB memiliki batasan berikut:

  - Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung.
  - Karakter pertamanya harus berupa huruf.
  - Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.
  - Ini harus unik untuk semua instans DB per AWS akun, per AWS Wilayah.
- Cloud Privat Virtual (VPC): Jika Anda sudah memiliki VPC, Anda dapat menggunakan VPC tersebut dengan klaster DB Aurora MySQL dengan memilih pengidentifikasi VPC Anda, misalnya `vpc-a464d1c1`. Untuk informasi tentang membuat VPC, lihat Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .

  Atau, Anda dapat memilih agar Aurora membuat VPC untuk Anda dengan memilih Buat VPC baru.
- Grup subnet: Jika Anda memiliki grup subnet yang ada, Anda dapat menggunakan grup subnet tersebut dengan klaster DB Aurora MySQL Anda dengan memilih pengidentifikasi grup subnet Anda, misalnya `gs-subnet-group1`.

  Jika tidak, Anda dapat memilih agar Aurora membuat grup subnet untuk Anda dengan memilih Buat grup subnet baru.
- Aksesibilitas publik: Pilih Tidak untuk menentukan bahwa instans di klaster DB Anda hanya dapat diakses oleh sumber daya di dalam VPC Anda. Pilih Ya untuk menentukan bahwa

instans dalam klaster DB Anda dapat diakses oleh sumber daya di jaringan publik. Default-nya adalah Ya.

> **Note**
>
> Klaster DB produksi Anda mungkin tidak perlu berada di subnet publik karena hanya server aplikasi Anda yang memerlukan akses ke klaster DB Anda. Jika klaster DB Anda tidak perlu berada di subnet publik, tetapkan Dapat Diakses Publik ke Tidak.

- Ketersediaan Zona: Pilih Zona Ketersediaan guna meng-host instans primer untuk klaster DB Aurora MySQL Anda. Agar Aurora memilih Zona Ketersediaan untuk Anda, pilih Tidak Ada Preferensi.
- Port Basis Data: Ketik port default yang akan digunakan saat menghubungkan ke instans di klaster DB Aurora MySQL. Default-nya adalah `3306`.

> **Note**
>
> Anda mungkin berada di belakang firewall perusahaan yang tidak mengizinkan akses ke port default seperti port default MySQL, 3306. Dalam hal ini, berikan nilai port yang diizinkan oleh firewall perusahaan Anda. Ingat nilai port tersebut nanti saat Anda terhubung ke klaster DB Aurora MySQL.

- Enkripsi: Pilih Aktifkan Enkripsi untuk klaster DB Aurora MySQL baru Anda agar dienkripsi saat diam. Jika Anda memilih Aktifkan Enkripsi, Anda harus memilih kunci KMS sebagai nilai AWS KMS key.

  Jika snapshot DB Anda tidak terenkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam.

  Jika snapshot DB Anda dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi yang ditentukan. Anda dapat menentukan kunci enkripsi yang digunakan oleh snapshot DB atau kunci yang berbeda. Anda tidak dapat membuat klaster DB yang tidak terenkripsi dari snapshot DB terenkripsi.
- Peningkatan Versi Minor Otomatis: Pengaturan ini tidak berlaku untuk klaster DB Aurora MySQL.

  Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin database untuk Amazon Aurora My SQL.

4. Pilih Migrasikan untuk memigrasikan snapshot DB Anda.
5. Pilih Instans, lalu pilih ikon panah untuk menampilkan detail klaster DB dan memantau progres migrasi. Di halaman detail, Anda dapat menemukan titik akhir klaster yang digunakan untuk terhubung ke instans primer klaster DB. Untuk informasi selengkapnya tentang menghubungkan ke klaster DB Aurora MySQL, lihat Menghubungkan ke klaster DB Amazon Aurora.

AWS CLI

Anda dapat membuat klaster DB Aurora dari snapshot DB milik instans DB RDS for MySQL menggunakan perintah `restore-db-cluster-from-snapshot` dengan parameter berikut:

- `--db-cluster-identifier` – Nama klaster DB yang akan dibuat.
- `--engine aurora-mysql` – Untuk klaster DB yang kompatibel dengan MySQL 5.7 atau yang kompatibel dengan MySQL 8.0.
- `--kms-key-id`— AWS KMS key Untuk mengenkripsi cluster DB secara opsional, tergantung pada apakah snapshot DB Anda dienkripsi.
  - Jika snapshot DB Anda tidak terenkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam. Jika tidak, klaster DB Anda tidak dienkripsi.
  - Jika snapshot DB Anda dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi yang ditentukan. Jika tidak, klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi untuk snapshot DB.

  > **Note**
  >
  > Anda tidak dapat membuat klaster DB yang tidak terenkripsi dari snapshot DB terenkripsi.

- `--snapshot-identifier` – Nama Amazon Resource Name (ARN) untuk snapshot DB yang akan dimigrasikan. Untuk informasi selengkapnya tentang Amazon RDS ARNs, lihat Amazon Relational Database Service (Amazon RDS).

Saat Anda memigrasikan snapshot DB dengan menggunakan perintah `RestoreDBClusterFromSnapshot`, perintah ini akan membuat klaster DB dan instans primer.

Dalam contoh ini, Anda membuat kluster DB yang kompatibel dengan MySQL 5.7 yang *`mydbcluster`* dinamai dari snapshot DB dengan ARN yang disetel ke. *`mydbsnapshotARN`*

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

Dalam contoh ini, Anda membuat kluster DB yang kompatibel dengan MySQL 5.7 yang *mydbcluster* dinamai dari snapshot DB dengan ARN yang disetel ke. *mydbsnapshotARN*

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

## Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora

Aurora menggunakan fungsi replikasi log biner mesin My SQL DB untuk membuat tipe khusus cluster DB yang disebut replika baca Aurora untuk sumber instans My DB. RDS SQL Pembaruan yang dibuat ke sumber RDS untuk instans My SQL DB direplikasi secara asinkron ke replika baca Aurora.

Sebaiknya gunakan fungsi ini untuk bermigrasi dari instans RDS for My SQL DB ke cluster Aurora SQL My DB dengan membuat replika baca Aurora dari sumber Anda RDS untuk instans DB Saya. SQL Ketika jeda replika antara instans RDS for My SQL DB dan replika baca Aurora adalah 0, Anda dapat mengarahkan aplikasi klien Anda ke replika baca Aurora dan kemudian menghentikan replikasi untuk membuat replika baca Aurora menjadi cluster Aurora My DB mandiri. SQL Lakukan antisipasi sesuai kebutuhan karena migrasi dapat memerlukan waktu yang lama, sekitar beberapa jam per tebibyte (TiB) data.

Untuk daftar wilayah tempat Aurora tersedia, lihat Amazon Aurora dalam Referensi Umum AWS.

Saat Anda membuat replika baca Aurora dari instans RDS for My SQL DB, Amazon RDS membuat snapshot DB dari sumber Anda RDS untuk instans SQL DB Saya (pribadi ke AmazonRDS, dan tidak dikenakan biaya). Amazon RDS kemudian memigrasikan data dari snapshot DB ke replika baca Aurora. Setelah data dari snapshot DB dimigrasikan ke cluster Aurora My DB yang baru, Amazon RDS memulai replikasi antara SQL instans RDS for My DB Anda dan SQL cluster Aurora My DB. SQL Jika instans RDS for My SQL DB berisi tabel yang menggunakan mesin penyimpanan selain InnoDB, atau yang menggunakan format baris terkompresi, Anda dapat mempercepat proses pembuatan replika baca Aurora dengan mengubah tabel tersebut untuk menggunakan mesin penyimpanan InnoDB dan format baris dinamis sebelum Anda membuat replika baca Aurora Anda. Untuk informasi lebih lanjut tentang proses menyalin snapshot My SQL DB ke cluster Aurora My SQL DB, lihat. Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL

Anda hanya dapat memiliki satu replika baca Aurora untuk instance RDS for My SQL DB.

> **Note**
>
> Masalah replikasi dapat muncul karena perbedaan fitur antara Aurora SQL My dan versi mesin database SQL Saya untuk instance SQL My DB RDS Anda yang merupakan primer replikasi. Jika Anda mengalami kesalahan, Anda dapat menemukan bantuan di forum RDS komunitas Amazon atau dengan menghubungi AWS Dukungan.
>
> Anda tidak dapat membuat replika baca Aurora jika instans RDS for My SQL DB Anda sudah menjadi sumber untuk replika baca Lintas wilayah.

Anda tidak dapat bermigrasi ke Aurora SQL My versi 3.05 dan yang lebih tinggi dari yang lebih RDS lama untuk versi SQL My 8.0, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan Anda meningkatkan ke RDS SQL versi Saya 8.0.28 sebelum bermigrasi.

Untuk informasi lebih lanjut tentang replika SQL baca saya, lihat Bekerja dengan replika baca dari instance MariaDB, MySQL, dan Postgre DB. SQL

Membuat replika baca Aurora

Anda dapat membuat replika baca Aurora untuk instans RDS for My SQL DB dengan menggunakan konsol, file, atau file AWS CLI. RDS API

Konsol

Untuk membuat replika baca Aurora dari sumber instans My RDS DB SQL

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih instans SQL DB Saya yang ingin Anda gunakan sebagai sumber untuk replika baca Aurora Anda.
4. Untuk Tindakan, pilih Buat replika baca Aurora.
5. Pilih spesifikasi klaster DB yang ingin Anda gunakan untuk replika baca Aurora, seperti yang dijelaskan dalam tabel berikut.

| Opsi | Deskripsi |
| --- | --- |
| Kelas instans DB | Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk instans primer dalam DB klaster. Untuk informasi selengkapnya tentang opsi kelas instans DB, lihat Kelas instans Amazon Aurora DB. |
| Deployment Multi-AZ | Pilih Buat Replika di Zona yang Berbeda untuk membuat replika siaga klaster DB baru di Zona Ketersediaan lain di Wilayah AWS target untuk dukungan failover. Untuk informasi selengkapnya |

| Opsi | Deskripsi |
| --- | --- |
| | tentang beberapa Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan. |
| Pengidentifikasi instans DB | Ketikkan nama untuk instans primer di klaster DB replika baca Aurora Anda. Pengidentifikasi ini digunakan dalam alamat titik akhir untuk instans primer klaster DB baru.<br><br>Pengidentifikasi instans DB memiliki batasan berikut:<br><br>• Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung.<br>• Karakter pertamanya harus berupa huruf.<br>• Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.<br>• Ini harus unik untuk semua instans DB untuk setiap AWS akun, untuk setiap AWS Wilayah.<br><br>Karena klaster DB replika baca Aurora dibuat dari snapshot instans DB sumber, nama pengguna master dan kata sandi master untuk replika baca Aurora sama dengan nama pengguna master dan kata sandi master untuk instans DB sumber. |
| Awan Pribadi Virtual (VPC) | Pilih VPC untuk meng-host cluster DB. Pilih Buat baru VPC agar Aurora membuat VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. |
| Grup subnet DB | Pilih grup subnet DB yang akan digunakan untuk klaster DB. Pilih Buat grup DB subnet baru agar Aurora membuat grup DB subnet untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. |

| Opsi | Deskripsi |
| --- | --- |
| Aksesibilitas publik | Pilih `Yes` untuk memberikan alamat IP publik pada klaster DB; jika tidak, pilih `No`. Instans dalam klaster DB Anda dapat berupa campuran antara instans DB publik dan privat. Untuk informasi selengkapnya tentang menyembunyikan instans dari akses publik, lihat Menyembunyikan klaster DB dalam VPC dari internet. |
| Zona ketersediaan | Tentukan apakah Anda ingin menentukan Zona Ketersediaan tertentu. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan. |
| VPCgrup keamanan (firewall) | Pilih Buat grup VPC keamanan baru agar Aurora membuat grup VPC keamanan untuk Anda. Pilih Pilih grup VPC keamanan yang ada untuk menentukan satu atau beberapa grup VPC keamanan untuk mengamankan akses jaringan ke cluster DB. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. |
| Port basis data | Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Aurora My SQL DB cluster default ke default My SQL port, 3306. Firewall di beberapa perusahaan memblokir koneksi ke port ini. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru. |
| Grup parameter DB | Pilih grup parameter DB untuk cluster Aurora My SQL DB. Aurora memiliki grup parameter DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter DB, lihat . |

| Opsi | Deskripsi |
| --- | --- |
| Grup parameter klaster DB | Pilih grup parameter cluster DB untuk cluster Aurora My SQL DB. Aurora memiliki grup parameter klaster DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter klaster DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat . |
| Enkripsi | Pilih Nonaktifkan enkripsi jika Anda tidak ingin klaster DB Aurora baru Anda dienkripsi. Pilih Aktifkan Enkripsi untuk klaster DB Aurora baru Anda agar dienkripsi saat diam. Jika Anda memilih Aktifkan enkripsi, Anda harus memilih KMS kunci sebagai AWS KMS keynilainya.<br><br>Jika instans My SQL DB Anda tidak dienkripsi, tentukan kunci enkripsi agar cluster DB Anda dienkrips i saat istirahat.<br><br>Jika instans My SQL DB Anda dienkripsi, tentukan kunci enkripsi agar cluster DB Anda dienkripsi saat istirahat menggunakan kunci enkripsi yang ditentuka n. Anda dapat menentukan kunci enkripsi yang digunakan oleh instans My SQL DB atau kunci yang berbeda. Anda tidak dapat membuat cluster DB yang tidak terenkripsi dari instans My DB terenkripsi. SQL |
| Prioritas | Pilih prioritas failover untuk klaster DB. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora. |

| Opsi | Deskripsi |
| --- | --- |
| Periode retensi cadangan | Pilih durasi waktu, dari 1 hingga 35 hari, saat Aurora harus mempertahankan salinan cadangan basis data. Salinan cadangan dapat digunakan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua. |
| Pemantauan yang Ditingkatkan | Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |
| Peran Pemantauan | Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Pilih IAM peran yang Anda buat untuk mengizinkan Aurora berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default agar Aurora membuat peran untuk Anda beri nama. `rds-monitoring-role` Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |
| Granularitas | Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda. |
| Peningkatan versi minor otomatis | Pengaturan ini tidak berlaku untuk klaster Aurora My SQL DB.<br><br>Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat. Pembaruan mesin database untuk Amazon Aurora My SQL |

| Opsi | Deskripsi |
| --- | --- |
| Jendela pemeliharaan | Klik Pilih jendela dan tentukan rentang waktu mingguan saat pemeliharaan sistem dapat dilakukan . Atau, pilih Tidak ada preferensi bagi Aurora untuk menetapkan periode secara acak. |

6. Pilih Buat replika baca.

AWS CLI

Untuk membuat replika baca Aurora dari sumber RDS instans My SQL DB, gunakan `create-db-instance` AWS CLI perintah `create-db-cluster`and untuk membuat cluster Aurora My DB baru. SQL Saat Anda memanggil `create-db-cluster` perintah, sertakan `--replication-source-identifier` parameter untuk mengidentifikasi Amazon Resource Name (ARN) untuk instans SQL DB saya sumber. Untuk informasi selengkapnya tentang Amazon RDSARNs, lihat Amazon Relational Database Service (RDSAmazon).

Jangan tentukan nama pengguna utama, kata sandi master, atau nama basis data karena replika baca Aurora menggunakan nama pengguna master, kata sandi utama, dan nama database yang sama dengan sumber instans DB SayaSQL.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Jika Anda menggunakan konsol untuk membuat replika baca Aurora, Aurora secara otomatis membuat instans primer untuk replika baca Aurora klaster DB Anda. Jika Anda menggunakan AWS

CLI untuk membuat replika baca Aurora, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.

Anda dapat membuat instance utama untuk cluster DB Anda dengan menggunakan `create-db-instance` AWS CLI perintah dengan parameter berikut.

- `--db-cluster-identifier`

  Nama klaster DB Anda.

- `--db-instance-class`

  Nama kelas instans DB yang akan digunakan untuk instans primer Anda.

- `--db-instance-identifier`

  Nama instance primer Anda.

- `--engine aurora`

Dalam contoh ini, Anda membuat instance primer yang diberi nama *myreadreplicainstance* untuk DB cluster yang diberi nama *myreadreplicacluster*, menggunakan kelas DB instance yang ditentukan dalam *myinstanceclass*.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
    --engine aurora
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

RDS API

Untuk membuat replika baca Aurora dari sumber RDS instans My SQL DB, gunakan RDS API perintah dan `CreateDBClusterCreateDBInstance`Amazon untuk membuat cluster Aurora DB baru dan instance utama. Jangan tentukan nama pengguna utama, kata sandi master, atau nama basis data karena replika baca Aurora menggunakan nama pengguna master, kata sandi utama, dan nama database yang sama dengan sumber RDS untuk instans DB SayaSQL.

Anda dapat membuat cluster Aurora DB baru untuk replika baca Aurora dari sumber RDS instans My SQL DB dengan menggunakan RDS API perintah `CreateDBCluster`Amazon dengan parameter berikut:

- `DBClusterIdentifier`

  Nama klaster DB yang akan dibuat.

- `DBSubnetGroupName`

  Nama grup subnet DB yang akan dikaitkan dengan klaster DB ini.

- `Engine=aurora`
- `KmsKeyId`

  AWS KMS key Untuk mengenkripsi cluster DB secara opsional, tergantung pada apakah instans My SQL DB Anda dienkripsi.

  - Jika instans My SQL DB Anda tidak dienkripsi, tentukan kunci enkripsi agar cluster DB Anda dienkripsi saat istirahat. Jika tidak, klaster DB Anda akan dienkripsi saat diam menggunakan kunci enkripsi default untuk akun Anda.
  - Jika instans My SQL DB Anda dienkripsi, tentukan kunci enkripsi agar cluster DB Anda dienkripsi saat istirahat menggunakan kunci enkripsi yang ditentukan. Jika tidak, cluster DB Anda dienkripsi saat istirahat menggunakan kunci enkripsi untuk instans My SQL DB.

  > **Note**
  >
  > Anda tidak dapat membuat cluster DB yang tidak terenkripsi dari instans My DB terenkripsi. SQL

- `ReplicationSourceIdentifier`

Amazon Resource Name (ARN) untuk sumber instans My SQL DB. Untuk informasi selengkapnya tentang Amazon RDSARNs, lihat Amazon Relational Database Service (RDSAmazon).

- `VpcSecurityGroupIds`

  Daftar grup EC2 VPC keamanan untuk dikaitkan dengan cluster DB ini.

Dalam contoh ini, Anda membuat DB cluster yang diberi nama *`myreadreplicacluster`* dari sumber instans SQL DB saya dengan satu ARN set ke *`mysqlprimaryARN`*, berkaitan yang terkait dengan grup DB subnet yang diberi nama *`mysubnetgroup`* Sebuah kelompok VPC keamanan yang bernama *`mysecuritygroup`*.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

Jika Anda menggunakan konsol untuk membuat replika baca Aurora, Aurora secara otomatis membuat instans primer untuk replika baca Aurora klaster DB Anda. Jika Anda menggunakan AWS CLI untuk membuat replika baca Aurora, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.

Anda dapat membuat instance utama untuk cluster DB Anda dengan menggunakan RDS API perintah `CreateDBInstance`Amazon dengan parameter berikut:

- `DBClusterIdentifier`

  Nama klaster DB Anda.

- `DBInstanceClass`

  Nama kelas instans DB yang akan digunakan untuk instans primer Anda.

- `DBInstanceIdentifier`

  Nama instance primer Anda.

- `Engine=aurora`

Dalam contoh ini, Anda membuat instance primer yang diberi nama *`myreadreplicainstance`* untuk DB cluster yang diberi nama *`myreadreplicacluster`*, menggunakan kelas DB instance yang ditentukan dalam *`myinstanceclass`*.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-09-01
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
    &X-Amz-Date=20140424T194844Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

Menampilkan Aurora read replica

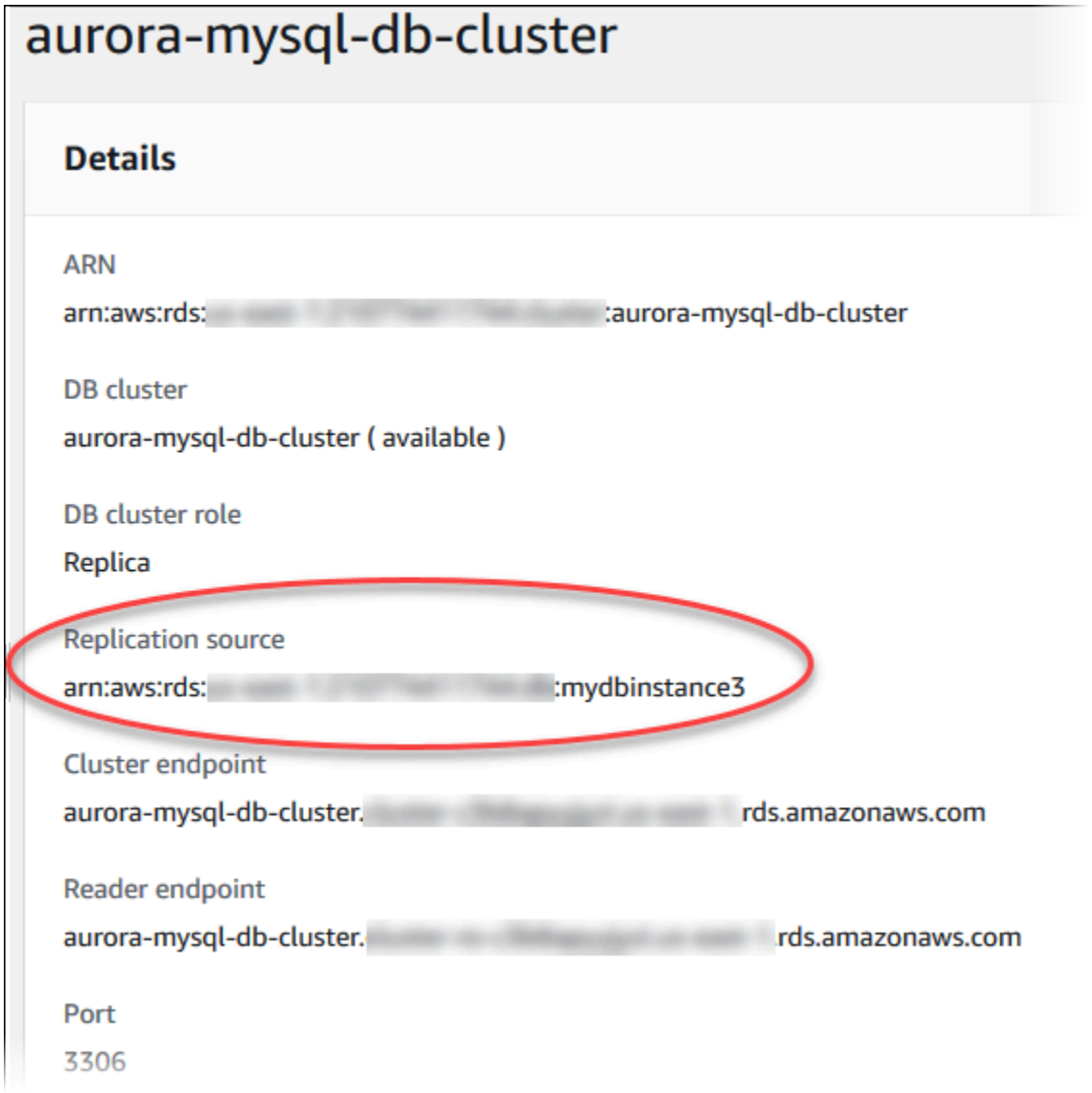
Anda dapat melihat hubungan SQL replikasi My SQL to Aurora My untuk cluster Aurora My SQL DB Anda dengan menggunakan atau. AWS Management Console AWS CLI

Konsol

Untuk melihat instans My SQL DB utama untuk replika baca Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.

3. Pilih klaster DB untuk replika baca Aurora untuk menampilkan detailnya. Informasi instans My SQL DB utama ada di bidang sumber Replikasi.




AWS CLI

Untuk melihat hubungan SQL replikasi My SQL to Aurora My untuk cluster Aurora My SQL DB Anda dengan menggunakan, gunakan perintah dan AWS CLI. `describe-db-clustersdescribe-db-instances`

Untuk menentukan instans My SQL DB mana yang utama, gunakan `describe-db-clusters`dan tentukan pengidentifikasi cluster replika baca Aurora untuk opsi tersebut. `--db-cluster-identifier` Lihat `ReplicationSourceIdentifier` elemen dalam output untuk instance DB yang merupakan primer replikasi. ARN

Untuk menentukan cluster DB mana yang merupakan replika baca Aurora, gunakan `describe-db-instances`dan tentukan pengenal instance instance My SQL DB untuk opsi tersebut. `--db-instance-identifier` Lihat elemen `ReadReplicaDBClusterIdentifiers` dalam output untuk pengidentifikasi klaster DB dari replika baca Aurora.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

Untuk Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

Mempromosikan replika baca Aurora

Setelah migrasi selesai, Anda dapat mempromosikan replika baca Aurora ke cluster DB yang berdiri sendiri menggunakan or. AWS Management Console AWS CLI

Kemudian, Anda dapat mengarahkan aplikasi klien Anda ke titik akhir untuk replika baca Aurora. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Koneksi titik akhir Amazon Aurora. Promosi akan selesai dengan cukup cepat, dan Anda dapat membaca dan menulis ke replika baca Aurora selama promosi. Namun, Anda tidak dapat menghapus instans My SQL DB utama atau memutuskan tautan Instans DB dan replika baca Aurora selama waktu ini.

Sebelum Anda mempromosikan replika baca Aurora Anda, hentikan transaksi apa pun agar tidak ditulis ke sumber instans My SQL DB, dan kemudian tunggu jeda replika pada replika baca Aurora mencapai 0. Anda dapat melihat lag replika untuk replika baca Aurora dengan memanggil perintah (Aurora My version 2) atau `SHOW SLAVE STATUS` `SHOW REPLICA STATUS` (Aurora SQL My SQL version 3) pada replika baca Aurora Anda. Periksa nilai Detik di belakang master.

Anda dapat mulai menulis ke replika baca Aurora setelah transaksi tulis ke primer berhenti dan lag replika adalah 0. Jika Anda menulis ke replika baca Aurora sebelum ini dan Anda memodifikasi tabel yang juga sedang dimodifikasi pada SQL primer Saya, Anda berisiko melanggar replikasi ke Aurora. Jika ini terjadi, Anda harus menghapus dan membuat ulang replika baca Aurora Anda.

Konsol

Untuk mempromosikan replika baca Aurora ke klaster DB Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB untuk replika baca Aurora.
4. Untuk Tindakan, pilih Promosikan.
5. Pilih Promosikan replika baca.

Setelah mempromosikan, konfirmasikan bahwa proses promosinya telah selesai dengan menggunakan prosedur berikut.

Untuk mengonfirmasi bahwa replika baca Aurora telah dipromosikan

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Peristiwa.
3. Di halaman Peristiwa, verifikasi bahwa ada peristiwa `Promoted Read Replica cluster to a stand-alone database cluster` untuk klaster yang Anda promosikan.

Setelah promosi selesai, instans My SQL DB utama dan replika baca Aurora dibatalkan tautannya, dan Anda dapat menghapus instans DB dengan aman jika Anda mau.

AWS CLI

Untuk mempromosikan replika baca Aurora ke cluster DB yang berdiri sendiri, gunakan perintah. `promote-read-replica-db-cluster` AWS CLI

Example

Untuk Linux, macOS, atau Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Untuk Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

# Mengelola Amazon Aurora MySQL

Bagian berikut membahas pengelolaan klaster DB Amazon Aurora MySQL.

Topik


- Mengelola performa dan penskalaan untuk Amazon Aurora MySQL
- Melakukan backtracking klaster DB Aurora
- Menguji Amazon Aurora My SQL menggunakan kueri injeksi kesalahan
- Mengubah tabel di Amazon Aurora menggunakan DDL Cepat
- Menampilkan status volume untuk klaster DB Aurora MySQL


## Mengelola performa dan penskalaan untuk Amazon Aurora MySQL

### Penskalaan instans DB Aurora MySQL

Anda dapat menskalakan instans DB Aurora MySQL dengan dua cara, penskalaan instans dan penskalaan baca. Untuk informasi selengkapnya tentang penskalaan baca, lihat Penskalaan baca.

Anda dapat menskalakan klaster DB Aurora MySQL Anda dengan mengubah kelas instans DB untuk setiap instans DB dalam klaster DB. Aurora MySQL mendukung beberapa kelas instans DB yang dioptimalkan untuk Aurora. Jangan gunakan kelas instans db.t2 atau db.t3 untuk klaster Aurora yang berukuran lebih dari 40 TB. Untuk spesifikasi kelas instans DB yang didukung Aurora MySQL, lihat Kelas instans Amazon Aurora DB.

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

### Koneksi maksimum ke instans DB Aurora MySQL

Jumlah maksimum koneksi yang diizinkan untuk instans DB Aurora MySQL ditentukan oleh parameter `max_connections` dalam grup parameter tingkat instans untuk instans DB.

Tabel berikut mencantumkan nilai default yang dihasilkan berupa `max_connections` untuk setiap kelas instans DB yang tersedia untuk Aurora MySQL. Anda dapat meningkatkan jumlah maksimum

koneksi ke instans DB Aurora MySQL Anda dengan menaikkan skala instans ke kelas instans DB dengan lebih banyak memori, atau dengan menetapkan nilai yang lebih besar untuk parameter `max_connections` di grup parameter DB untuk instans Anda, hingga 16.000.

> **Tip**
>
> Jika aplikasi Anda sering membuka dan menutup koneksi, atau membiarkan sejumlah besar koneksi yang lama aktif tetap terbuka, kami menyarankan agar Anda menggunakan Proksi Amazon RDS. Proksi RDS adalah proksi basis data yang sepenuhnya terkelola dengan ketersediaan tinggi yang menggunakan pooling koneksi untuk berbagi koneksi basis data dengan aman dan efisien. Untuk mempelajari tentang Proksi RDS, lihat Proksi Amazon RDS untuk Aurora.

Untuk detail tentang cara instans Aurora Serverless v2 menangani parameter ini, lihat Koneksi maksimum untuk Aurora Serverless v2.

| Kelas instans | Nilai default max_connections | | |
|---|---|---|---|
| db.t2.small | 45 | | |
| db.t2.medium | 90 | | |
| db.t3.small | 45 | | |
| db.t3.medium | 90 | | |
| db.t3.large | 135 | | |
| db.t4g.medium | 90 | | |
| db.t4g.large | 135 | | |
| db.r3.large | 1000 | | |
| db.r3.xlarge | 2000 | | |

| Kelas instans | Nilai default max_connections | | |
|---|---|---|---|
| db.r3.2xlarge | 3000 | | |
| db.r3.4xlarge | 4000 | | |
| db.r3.8xlarge | 5000 | | |
| db.r4.large | 1000 | | |
| db.r4.xlarge | 2000 | | |
| db.r4.2xlarge | 3000 | | |
| db.r4.4xlarge | 4000 | | |
| db.r4.8xlarge | 5000 | | |
| db.r4.16xlarge | 6000 | | |
| db.r5.large | 1000 | | |
| db.r5.xlarge | 2000 | | |
| db.r5.2xlarge | 3000 | | |
| db.r5.4xlarge | 4000 | | |
| db.r5.8xlarge | 5000 | | |
| db.r5.12xlarge | 6000 | | |
| db.r5.16xlarge | 6000 | | |
| db.r5.24xlarge | 7000 | | |
| db.r6g.large | 1000 | | |
| db.r6g.xlarge | 2000 | | |

| Kelas instans | Nilai default max_connections | | |
|---|---|---|---|
| db.r6g.2xlarge | 3000 | | |
| db.r6g.4xlarge | 4000 | | |
| db.r6g.8xlarge | 5000 | | |
| db.r6g.12xlarge | 6000 | | |
| db.r6g.16xlarge | 6000 | | |
| db.r6i.large | 1000 | | |
| db.r6i.xlarge | 2000 | | |
| db.r6i.2xlarge | 3000 | | |
| db.r6i.4xlarge | 4000 | | |
| db.r6i.8xlarge | 5000 | | |
| db.r6i.12xlarge | 6000 | | |
| db.r6i.16xlarge | 6000 | | |
| db.r6i.24xlarge | 7000 | | |
| db.r6i.32xlarge | 7000 | | |
| db.r7g.large | 1000 | | |
| db.r7g.xlarge | 2000 | | |
| db.r7g.2xlarge | 3000 | | |
| db.r7g.4xlarge | 4000 | | |
| db.r7g.8xlarge | 5000 | | |

| Kelas instans | Nilai default max_connections | | |
|---|---|---|---|
| db.r7g.12xlarge | 6000 | | |
| db.r7g.16xlarge | 6000 | | |
| db.r7i.large | 1000 | | |
| db.r7i.xlarge | 2000 | | |
| db.r7i.2xlarge | 3000 | | |
| db.r7i.4xlarge | 4000 | | |
| db.r7i.8xlarge | 5000 | | |
| db.r7i.12xlarge | 6000 | | |
| db.r7i.16xlarge | 6000 | | |
| db.r7i.24xlarge | 7000 | | |
| db.r7i.48xlarge | 8000 | | |
| db.r8g.large | 1000 | | |
| db.r8g.xlarge | 2000 | | |
| db.r8g.2xlarge | 3000 | | |
| db.r8g.4xlarge | 4000 | | |
| db.r8g.8xlarge | 5000 | | |
| db.r8g.12xlarge | 6000 | | |
| db.r8g.16xlarge | 6000 | | |
| db.r8g.24xlarge | 7000 | | |

| Kelas instans | Nilai default max_connections | | |
|---|---|---|---|
| db.r8g.48xlarge | 8000 | | |
| db.x2g.large | 2000 | | |
| db.x2g.xlarge | 3000 | | |
| db.x2g.2xlarge | 4000 | | |
| db.x2g.4xlarge | 5000 | | |
| db.x2g.8xlarge | 6000 | | |
| db.x2g.12xlarge | 7000 | | |
| db.x2g.16xlarge | 7000 | | |

**Tip**

Perhitungan `max_connections` parameter menggunakan basis log 2 (berbeda dari logaritma natural) dan `DBInstanceClassMemory` nilai dalam byte untuk kelas instance Aurora MySQL yang dipilih. Parameter hanya menerima nilai integer, dengan bagian desimal terpotong dari perhitungan. Rumus mengimplementasikan batas koneksi sebagai berikut:

- Peningkatan koneksi 1000 untuk instans R3, R4, dan R5 yang lebih besar
- 45 peningkatan koneksi untuk varian memori instans T2 dan T3

Contoh: Untuk db.r6g.large, sedangkan rumus menghitung 1069.2, sistem mengimplementasikan 1000 untuk mempertahankan pola inkremental yang konsisten.

Jika Anda membuat grup parameter baru untuk menyesuaikan nilai default Anda sendiri untuk batas koneksi, Anda akan melihat bahwa batas koneksi default akan diperoleh menggunakan formula berdasarkan nilai `DBInstanceClassMemory`. Seperti ditunjukkan dalam tabel sebelumnya, rumus

ini menghasilkan batas koneksi yang bertambah sebanyak 1000 saat memorinya berlipat ganda antara instans R3, R4, dan R5 yang makin besar, dan sebanyak 45 untuk ukuran memori yang berbeda-beda pada instans T2 dan T3.

Lihat Menentukan parameter DB untuk detail selengkapnya tentang cara `DBInstanceClassMemory` dihitung.

instans DB Aurora MySQL dan RDS for MySQL memiliki jumlah overhead memori yang berbeda. Oleh karena itu, nilai `max_connections` dapat berbeda untuk instans DB Aurora MySQL dan RDS for MySQL yang menggunakan kelas instans yang sama. Nilai-nilai dalam tabel ini hanya berlaku untuk instans DB Aurora MySQL.

> **Note**
>
> Batas konektivitas untuk instans T2 dan T3 jauh lebih rendah karena dengan Aurora, kelas instans tersebut hanya ditujukan untuk skenario pengembangan dan pengujian, bukan untuk beban kerja produksi.

Batas koneksi default disesuaikan untuk sistem yang menggunakan nilai default untuk pemakai memori besar lainnya, seperti pool buffer dan cache kueri. Jika Anda mengubah pengaturan lain tersebut untuk klaster Anda, pertimbangkan untuk menyesuaikan batas koneksi agar memperhitungkan peningkatan atau penurunan memori yang tersedia pada instans DB.

## Batas penyimpanan sementara untuk Aurora MySQL

Aurora MySQL menyimpan tabel dan indeks dalam subsistem penyimpanan Aurora. Aurora MySQL menggunakan penyimpanan sementara atau lokal terpisah untuk file sementara nonpersisten dan tabel sementara non-InnoDB. Penyimpanan lokal juga mencakup file yang digunakan untuk tujuan seperti mengurutkan set data besar selama pemrosesan kueri atau untuk operasi pembuatan indeks. Itu tidak termasuk tabel sementara InnoDB.

Untuk informasi selengkapnya tentang tabel sementara di Aurora MySQL versi 3, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3. Untuk informasi selengkapnya tentang tabel sementara di versi 2, lihat Perilaku tablespace sementara di Aurora Versi saya 2 SQL.

Data dan file sementara pada volume ini hilang saat memulai dan menghentikan instans DB, dan selama penggantian host.

Volume penyimpanan lokal ini didukung oleh Amazon Elastic Block Store (EBS) dan dapat diperluas dengan menggunakan kelas instans DB yang lebih besar. Untuk informasi selengkapnya tentang penyimpanan, lihat Penyimpanan Amazon Aurora.

Penyimpanan lokal juga digunakan untuk mengimpor data dari Amazon S3 `LOAD DATA FROM S3` menggunakan `LOAD XML FROM S3` atau, dan untuk mengekspor data ke S3 menggunakan SELECT INTO OUTFILE S3. Untuk informasi lebih lanjut tentang mengimpor dari dan mengekspor ke S3, lihat berikut ini:

- Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3
- Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3

Aurora MySQL menggunakan penyimpanan permanen terpisah untuk log kesalahan, log umum, log kueri lambat, dan log audit untuk sebagian besar kelas instans Aurora MySQL DB (tidak termasuk tipe kelas instance berkinerja burstable seperti db.t2, db.t3, dan db.t4g). Data pada volume ini dipertahankan saat memulai dan menghentikan instans DB, dan selama penggantian host.

Volume penyimpanan permanen ini juga didukung oleh Amazon EBS dan memiliki ukuran tetap sesuai dengan kelas instans DB. Itu tidak dapat diperpanjang dengan menggunakan kelas instans DB yang lebih besar.

Tabel berikut menunjukkan jumlah maksimum penyimpanan sementara dan permanen yang tersedia untuk setiap kelas instans MySQL Aurora MySQL DB. Untuk informasi selengkapnya tentang dukungan kelas instans DB untuk Aurora, lihat Kelas instans Amazon Aurora DB.

| Kelas instans DB | temporary/local Penyimpanan maksimum yang tersedia (GiB) | Penyimpanan maksimum tambahan tersedia untuk file log (GiB) |
|---|---|---|
| db.x2g.16xlarge | 1280 | 500 |
| db.x2g.12xlarge | 960 | 500 |
| db.x2g.8xlarge | 640 | 500 |
| db.x2g.4xlarge | 320 | 500 |
| db.x2g.2xlarge | 160 | 60 |

| Kelas instans DB | temporary/local Penyimpanan maksimum yang tersedia (GiB) | Penyimpanan maksimum tambahan tersedia untuk file log (GiB) |
|---|---|---|
| db.x2g.xlarge | 80 | 60 |
| db.x2g.large | 40 | 60 |
| db.r8g.48xlarge | 3840 | 500 |
| db.r8g.24xlarge | 1920 | 500 |
| db.r8g.16xlarge | 1280 | 500 |
| db.r8g.12xlarge | 960 | 500 |
| db.r8g.8xlarge | 640 | 500 |
| db.r8g.4xlarge | 320 | 500 |
| db.r8g.2xlarge | 160 | 60 |
| db.r8g.xlarge | 80 | 60 |
| db.r8g.large | 32 | 60 |
| db.r7i.48xlarge | 3840 | 500 |
| db.r7i.24xlarge | 1920 | 500 |
| db.r7i.16xlarge | 1280 | 500 |
| db.r7i.12xlarge | 960 | 500 |
| db.r7i.8xlarge | 640 | 500 |
| db.r7i.4xlarge | 320 | 500 |
| db.r7i.2xlarge | 160 | 60 |
| db.r7i.xlarge | 80 | 60 |

| Kelas instans DB | temporary/local Penyimpanan maksimum yang tersedia (GiB) | Penyimpanan maksimum tambahan tersedia untuk file log (GiB) |
|---|---|---|
| db.r7i.large | 32 | 60 |
| db.r7g.16xlarge | 1280 | 500 |
| db.r7g.12xlarge | 960 | 500 |
| db.r7g.8xlarge | 640 | 500 |
| db.r7g.4xlarge | 320 | 500 |
| db.r7g.2xlarge | 160 | 60 |
| db.r7g.xlarge | 80 | 60 |
| db.r7g.large | 32 | 60 |
| db.r6i.32xlarge | 2560 | 500 |
| db.r6i.24xlarge | 1920 | 500 |
| db.r6i.16xlarge | 1280 | 500 |
| db.r6i.12xlarge | 960 | 500 |
| db.r6i.8xlarge | 640 | 500 |
| db.r6i.4xlarge | 320 | 500 |
| db.r6i.2xlarge | 160 | 60 |
| db.r6i.xlarge | 80 | 60 |
| db.r6i.large | 32 | 60 |
| db.r6g.16xlarge | 1280 | 500 |
| db.r6g.12xlarge | 960 | 500 |

| Kelas instans DB | temporary/local Penyimpanan maksimum yang tersedia (GiB) | Penyimpanan maksimum tambahan tersedia untuk file log (GiB) |
|---|---|---|
| db.r6g.8xlarge | 640 | 500 |
| db.r6g.4xlarge | 320 | 500 |
| db.r6g.2xlarge | 160 | 60 |
| db.r6g.xlarge | 80 | 60 |
| db.r6g.large | 32 | 60 |
| db.r5.24xlarge | 1920 | 500 |
| db.r5.16xlarge | 1280 | 500 |
| db.r5.12xlarge | 960 | 500 |
| db.r5.8xlarge | 640 | 500 |
| db.r5.4xlarge | 320 | 500 |
| db.r5.2xlarge | 160 | 60 |
| db.r5.xlarge | 80 | 60 |
| db.r5.large | 32 | 60 |
| db.r4.16xlarge | 1280 | 500 |
| db.r4.8xlarge | 640 | 500 |
| db.r4.4xlarge | 320 | 500 |
| db.r4.2xlarge | 160 | 60 |
| db.r4.xlarge | 80 | 60 |
| db.r4.large | 32 | 60 |

| Kelas instans DB | temporary/local Penyimpanan maksimum yang tersedia (GiB) | Penyimpanan maksimum tambahan tersedia untuk file log (GiB) |
|---|---|---|
| db.t4g.large | 32 | – |
| db.t4g.medium | 32 | – |
| db.t3.large | 32 | – |
| db.t3.medium | 32 | – |
| db.t3.small | 32 | – |
| db.t2.medium | 32 | – |
| db.t2.small | 32 | – |

**Important**

Nilai-nilai ini merepresentasikan jumlah maksimum teoretis penyimpanan kosong di setiap instans DB. Penyimpanan lokal sebenarnya yang tersedia untuk Anda mungkin lebih rendah. Aurora menggunakan beberapa penyimpanan lokal untuk proses manajemennya, dan instans DB menggunakan beberapa penyimpanan lokal bahkan sebelum Anda memuat data apa pun. Anda dapat memantau penyimpanan sementara yang tersedia untuk instans DB tertentu dengan `FreeLocalStorage` CloudWatch metrik, yang dijelaskan dalam CloudWatch Metrik Amazon untuk Amazon Aurora. Anda dapat memeriksa jumlah penyimpanan kosong saat ini. Anda juga dapat memetakan jumlah penyimpanan kosong dari waktu ke waktu. Memantau penyimpanan kosong dari waktu ke waktu membantu Anda menentukan apakah nilainya meningkat atau menurun, atau untuk menemukan nilai minimum, maksimum, atau rata-rata.
(Hal ini tidak berlaku untuk Aurora Serverless v2.)

## Melakukan backtracking klaster DB Aurora

Dengan Amazon Aurora My SQL -Compatible Edition, Anda dapat mem-backtrack cluster DB ke waktu tertentu, tanpa memulihkan data dari cadangan.

Daftar Isi


- Gambaran umum backtracking
  - Jendela backtrack
  - Waktu backtracking
  - Batasan backtracking
- Wilayah dan ketersediaan versi
- Pertimbangan peningkatan untuk klaster yang memiliki backtrack aktif
- Mengonfigurasi backtracking cluster Aurora My DB SQL
- Melakukan backtrack untuk cluster Aurora SQL My DB
- Memantau backtracking untuk cluster Aurora My DB SQL
- Berlangganan peristiwa backtrack dengan konsol
- Mengambil backtrack yang ada
- Menonaktifkan backtracking untuk cluster Aurora My DB SQL


## Gambaran umum backtracking

Backtracking “memundurkan” klaster DB ke waktu yang Anda tentukan. Backtracking bukan pengganti untuk mencadangkan klaster DB sehingga Anda dapat memulihkannya ke suatu titik waktu. Namun, backtracking memberikan keuntungan berikut dibandingkan pencadangan dan pemulihan biasa:

- Anda dapat dengan mudah membatalkan kesalahan. Jika Anda keliru melakukan tindakan destruktif, seperti DELETE tanpa WHERE klausa, Anda dapat mundur cluster DB ke waktu sebelum tindakan destruktif dengan gangguan layanan minimal.
- Anda dapat melakukan backtracking klaster DB dengan cepat. Memulihkan klaster DB ke titik waktu akan meluncurkan klaster DB baru dan memulihkannya dari data cadangan atau snapshot klaster DB, yang dapat memakan waktu berjam-jam. Backtracking klaster DB tidak memerlukan klaster DB baru dan memundurkan klaster DB dalam hitungan menit.
- Anda dapat menjelajahi perubahan data sebelumnya. Anda dapat berulang kali melakukan backtracking klaster DB secara tepat waktu untuk membantu menentukan kapan perubahan data tertentu terjadi. Misalnya, Anda dapat melakukan backtracking klaster DB tiga jam lalu melacak maju dalam waktu satu jam. Dalam hal ini, waktu backtrack adalah dua jam sebelum waktu asli.

> **Note**
>
> Untuk informasi tentang memulihkan klaster DB ke suatu titik waktu, lihat Gambaran umum pencadangan dan pemulihan klaster DB Aurora.

### Jendela backtrack

Dengan backtrack, ada jendela backtrack target dan jendela backtrack aktual:

- Jendela backtrack target adalah jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking klaster DB Anda. Saat Anda mengaktifkan backtrack, Anda menentukan jendela backtrack target. Misalnya, Anda dapat menentukan jendela backtrack target selama 24 jam jika Anda ingin dapat melakukan backtracking klaster DB selama satu hari.
- Jendela backtrack aktual adalah jumlah waktu aktual yang Anda inginkan untuk dapat melakukan backtracking klaster DB, yang dapat lebih kecil daripada jendela backtrack target. Jendela backtrack aktual didasarkan pada beban kerja dan penyimpanan yang tersedia untuk menyimpan informasi tentang perubahan basis data, yang disebut catatan perubahan.

Saat Anda membuat pembaruan pada klaster Aurora DB dengan backtracking aktif, Anda menghasilkan catatan perubahan. Aurora menyimpan catatan perubahan untuk jendela backtrack target, dan Anda membayar tarif per jam untuk menyimpannya. Jendela backtrack target dan beban kerja di klaster DB Anda menentukan jumlah catatan perubahan yang Anda simpan. Beban kerja adalah jumlah perubahan yang Anda buat pada klaster DB Anda dalam waktu tertentu. Jika beban kerja Anda berat, Anda menyimpan lebih banyak perubahan catatan di jendela backtrack dibandingkan jika beban kerja Anda ringan.

Anda dapat membayangkan jendela backtrack target sebagai sasaran jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking klaster DB Anda. Dalam kebanyakan kasus, Anda dapat melakukan backtracking dalam jumlah waktu maksimum yang Anda tentukan. Namun, dalam beberapa kasus, klaster DB tidak dapat menyimpan cukup catatan untuk melakukan backtracking dalam jumlah waktu maksimum, dan jendela backtrack aktual Anda lebih kecil daripada target Anda. Biasanya, jendela backtrack aktual lebih kecil daripada target saat Anda memiliki beban kerja yang sangat berat pada klaster DB Anda. Saat jendela backtrack aktual Anda lebih kecil dari target, kami mengirimkan notifikasi kepada Anda.

Saat backtracking diaktifkan untuk klaster DB, dan Anda menghapus tabel yang disimpan dalam klaster DB, Aurora mempertahankan tabel tersebut di catatan perubahan backtrack. Ini dilakukan

agar Anda dapat kembali ke waktu sebelum Anda menghapus tabel. Jika Anda tidak memiliki cukup ruang di jendela backtrack Anda untuk menyimpan tabel, pada akhirnya tabel mungkin akan dihapus dari catatan perubahan backtrack.

### Waktu backtracking

Aurora selalu melakukan backtracking ke waktu yang konsisten untuk klaster DB. Tindakan ini menghilangkan kemungkinan transaksi yang tidak di-commit saat backtrack selesai. Saat Anda menentukan waktu untuk backtrack, Aurora akan secara otomatis memilih waktu konsisten sedekat mungkin. Pendekatan ini berarti bahwa backtrack yang diselesaikan mungkin tidak sama persis dengan waktu yang Anda tentukan, tetapi Anda dapat menentukan waktu yang tepat untuk backtrack dengan menggunakan describe-db-cluster-backtracks AWS CLIperintah. Untuk informasi selengkapnya, lihat Mengambil backtrack yang ada.

### Batasan backtracking

Batasan berikut berlaku untuk backtracking:

- Backtracking hanya tersedia untuk klaster DB yang dibuat dengan fitur Backtrack diaktifkan. Anda tidak dapat memodifikasi cluster DB untuk mengaktifkan fitur Backtrack. Anda dapat mengaktifkan fitur Backtrack saat membuat klaster DB baru atau memulihkan snapshot klaster DB.
- Batasan untuk jendela backtrack adalah 72 jam.
- Backtracking memengaruhi seluruh klaster DB. Misalnya, Anda tidak dapat memilih untuk melakukan backtracking satu tabel atau satu pembaruan data.
- Anda tidak dapat membuat replika baca lintas wilayah dari cluster yang mendukung backtrack, tetapi Anda masih dapat mengaktifkan replikasi log biner (binlog) di cluster. Jika Anda mencoba untuk mundur cluster DB yang logging biner diaktifkan, kesalahan biasanya terjadi kecuali Anda memilih untuk memaksa backtrack. Setiap upaya untuk memaksa backtrack akan merusak replika baca hilir dan mengganggu operasi lain seperti penerapan biru/hijau.
- Anda tidak dapat melakukan backtracking klon basis data ke waktu sebelum klon basis data tersebut dibuat. Namun, Anda dapat menggunakan basis data asli untuk melakukan backtracking ke suatu waktu sebelum klon dibuat. Untuk informasi selengkapnya tentang kloning basis data, lihat Mengkloning volume untuk klaster DB Amazon Aurora.
- Backtracking akan menyebabkan gangguan instans DB dalam waktu singkat. Anda harus menghentikan atau menjeda aplikasi sebelum memulai operasi backtrack untuk memastikan bahwa tidak ada permintaan baca atau tulis baru. Selama operasi backtrack, Aurora menjeda basis data, menutup semua koneksi yang terbuka, dan menghentikan operasi baca dan tulis yang tidak di-commit. Aurora kemudian menunggu operasi backtrack selesai.

- Anda tidak dapat memulihkan snapshot Lintas wilayah dari kluster berkemampuan backtrack di AWS Wilayah yang tidak mendukung backtracking.
- Jika Anda melakukan pemutakhiran di tempat untuk cluster berkemampuan backtrack dari Aurora SQL My versi 2 ke versi 3, Anda tidak dapat mundur ke titik waktu sebelum pemutakhiran terjadi.

## Wilayah dan ketersediaan versi

Backtrack tidak tersedia untuk SQL Aurora Postgre.

Berikut ini adalah mesin yang didukung dan ketersediaan Wilayah untuk Backtrack dengan SQL Aurora My.

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| AS Timur (Virginia Utara) | Semua versi | Semua versi |
| AS Timur (Ohio) | Semua versi | Semua versi |
| AS Barat (California Utara) | Semua versi | Semua versi |
| AS Barat (Oregon) | Semua versi | Semua versi |
| Afrika (Cape Town) | – | – |
| Asia Pasifik (Hong Kong) | – | – |
| Asia Pasifik (Jakarta) | – | – |
| Asia Pasifik (Malaysia) | – | – |
| Asia Pasifik (Melbourne) | – | – |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| Asia Pasifik (Mumbai) | Semua versi | Semua versi |
| Asia Pasifik (Osaka) | Semua versi | Versi 2.07.3 dan lebih tinggi |
| Asia Pasifik (Seoul) | Semua versi | Semua versi |
| Asia Pasifik (Singapura) | Semua versi | Semua versi |
| Asia Pasifik (Sydney) | Semua versi | Semua versi |
| Asia Pasifik (Tokyo) | Semua versi | Semua versi |
| Kanada (Pusat) | Semua versi | Semua versi |
| Kanada Barat (Calgary) | – | – |
| China (Beijing) | – | – |
| Tiongkok (Ningxia) | – | – |
| Eropa (Frankfurt) | Semua versi | Semua versi |
| Eropa (Irlandia) | Semua versi | Semua versi |
| Eropa (London) | Semua versi | Semua versi |
| Eropa (Milan) | – | – |
| Eropa (Paris) | Semua versi | Semua versi |
| Eropa (Spanyol) | – | – |

| Wilayah | Aurora Versi saya 3 SQL | Aurora Versi saya 2 SQL |
|---|---|---|
| Eropa (Stockholm) | – | – |
| Eropa (Zürich) | – | – |
| Israel (Tel Aviv) | – | – |
| Timur Tengah (Bahrain) | – | – |
| Timur Tengah (UAE) | – | – |
| Amerika Selatan (Sao Paulo) | – | – |
| AWS GovCloud (AS-Timur) | – | – |
| AWS GovCloud (AS-Barat) | – | – |

## Pertimbangan peningkatan untuk klaster yang memiliki backtrack aktif

Anda dapat memutakhirkan cluster DB berkemampuan backtrack dari Aurora My SQL versi 2 ke versi 3, karena semua versi minor Aurora My versi 3 didukung untuk Backtrack. SQL

## Mengonfigurasi backtracking cluster Aurora My DB SQL

Untuk menggunakan fitur Backtrack, Anda harus mengaktifkan backtracking dan menentukan jendela backtrack target. Jika tidak, backtracking dinonaktifkan.

Untuk Jendela Backtrack Target, tentukan jumlah waktu yang Anda inginkan untuk dapat memundurkan basis data Anda menggunakan Backtrack. Aurora mencoba mempertahankan catatan perubahan yang cukup untuk mendukung periode waktu tersebut.

Konsol

Anda dapat menggunakan konsol untuk mengonfigurasi backtracking saat Anda membuat klaster DB baru. Anda juga dapat memodifikasi klaster DB untuk mengubah jendela backtrack untuk klaster yang memiliki backtrack aktif. Jika Anda menonaktifkan backtrack sepenuhnya untuk klaster dengan mengatur jendela backtrack ke 0, Anda tidak dapat mengaktifkan backtrack lagi untuk klaster tersebut.

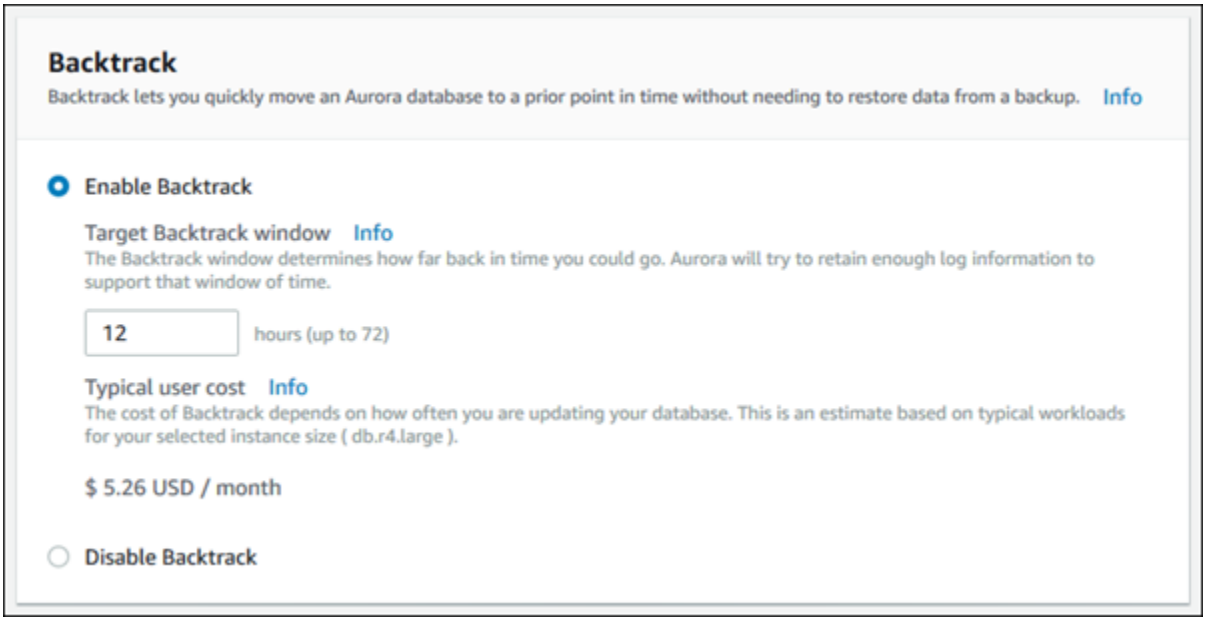
Topik

- Mengonfigurasi backtracking dengan konsol saat membuat klaster DB
- Mengonfigurasi backtrack dengan konsol saat memodifikasi klaster DB

## Mengonfigurasi backtracking dengan konsol saat membuat klaster DB

Saat Anda membuat cluster Aurora My SQL DB baru, backtracking dikonfigurasi saat Anda memilih Aktifkan Backtrack dan tentukan nilai jendela Target Backtrack yang lebih besar dari nol di bagian Backtrack.

Untuk membuat klaster DB, ikuti petunjuk dalam Membuat klaster DB Amazon Aurora. Gambar berikut menunjukkan bagian Backtrack.




Ketika Anda membuat klaster DB baru, Aurora tidak memiliki data untuk beban kerja klaster DB. Jadi, Aurora tidak memperkirakan biaya secara khusus untuk klaster DB baru. Sebagai gantinya, konsol akan menampilkan biaya pengguna standar untuk jendela backtrack target yang ditentukan

berdasarkan beban kerja standar. Biaya standar dimaksudkan untuk memberikan referensi umum untuk biaya fitur Backtrack.

> **Important**
>
> Biaya Anda sebenarnya mungkin tidak sama dengan biaya standar karena biaya Anda sebenarnya didasarkan pada beban kerja klaster DB Anda.

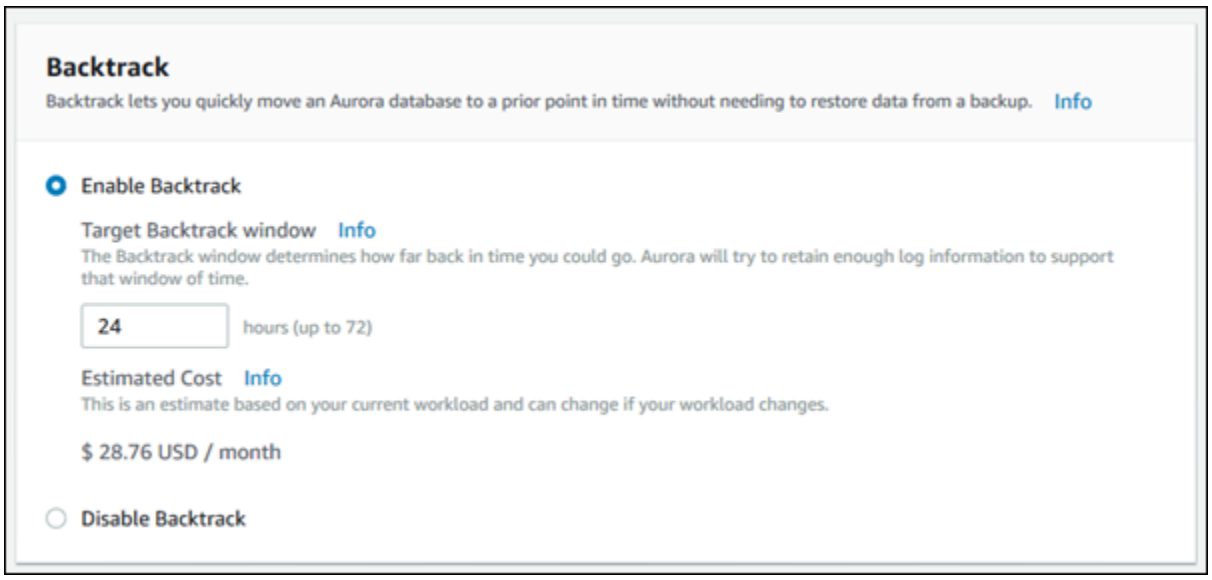
### Mengonfigurasi backtrack dengan konsol saat memodifikasi klaster DB

Anda dapat memodifikasi backtrack untuk klaster DB menggunakan konsol.

> **Note**
>
> Saat ini, Anda dapat memodifikasi backtracking hanya untuk klaster DB yang memiliki fitur Backtrack aktif. Bagian Backtrack ini tidak muncul untuk klaster DB yang dibuat dengan fitur Backtrack dinonaktifkan atau jika fitur Backtrack telah dinonaktifkan untuk klaster DB tersebut.

#### Untuk memodifikasi backtracking untuk klaster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih klaster yang ingin Anda ubah, dan pilih Modifikasi.
4. Untuk Jendela Backtrack Target, ubah jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking. Batasannya adalah 72 jam.




Konsol menunjukkan perkiraan biaya untuk jumlah waktu yang Anda tentukan berdasarkan beban kerja terdahulu di klaster DB:

- Jika backtracking dinonaktifkan pada cluster DB, perkiraan biaya didasarkan pada `VolumeWriteIOPS` metrik untuk cluster DB di Amazon CloudWatch.
- Jika backtracking diaktifkan sebelumnya di cluster DB, perkiraan biaya didasarkan pada `BacktrackChangeRecordsCreationRate` metrik untuk cluster DB di Amazon CloudWatch.

5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih salah satu dari berikut ini:

   - Terapkan selama jendela pemeliharaan terjadwal berikutnya – Tunggu untuk menerapkan modifikasi Jendela Backtrack Target hingga periode pemeliharaan berikutnya.
   - Terapkan segera – Terapkan modifikasi Jendela Backtrack Target sesegera mungkin.
7. Pilih Ubah klaster.

AWS CLI

Saat Anda membuat cluster Aurora My SQL DB baru menggunakan create-db-cluster AWS CLIperintah, backtracking dikonfigurasi saat Anda menentukan `--backtrack-window` nilai yang lebih besar dari nol. Nilai `--backtrack-window` menentukan jendela backtrack target. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Anda juga dapat menentukan `--backtrack-window` nilai menggunakan AWS CLI perintah berikut:

- modify-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Prosedur berikut menjelaskan cara memodifikasi jendela backtrack target untuk klaster DB menggunakan AWS CLI.

Untuk memodifikasi jendela backtrack target untuk cluster DB menggunakan AWS CLI

- Panggil modify-db-cluster AWS CLIperintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--backtrack-window` – Jumlah maksimum detik yang Anda inginkan untuk melakukan backtracking klaster DB.

  Contoh berikut mengatur jendela backtrack target untuk `sample-cluster` ke satu hari (86.400 detik).

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

  Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
>
> Saat ini, Anda dapat mengaktifkan backtracking hanya untuk klaster DB yang dibuat dengan fitur Backtrack aktif.

RDS API

Saat Anda membuat cluster Aurora My SQL DB baru menggunakan RDS API operasi C reateDBCluster Amazon, backtracking dikonfigurasi saat Anda menentukan `BacktrackWindow` nilai yang lebih besar dari nol. Nilai `BacktrackWindow` menentukan jendela backtrack target untuk klaster DB yang ditentukan dalam nilai `DBClusterIdentifier`. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Anda juga dapat menentukan `BacktrackWindow` nilai menggunakan API operasi berikut:

- M odifyDBCluster
- R estoreDBCluster dari3
- R estoreDBCluster FromSnapshot
- R estoreDBCluster ToPointInTime

> **Note**
>
> Saat ini, Anda dapat mengaktifkan backtracking hanya untuk klaster DB yang dibuat dengan fitur Backtrack aktif.

## Melakukan backtrack untuk cluster Aurora SQL My DB

Anda dapat melakukan backtracking klaster DB ke stempel waktu backtrack tertentu. Jika stempel waktu backtrack tidak lebih awal dari waktu backtrack yang paling awal, dan tidak di masa depan, klaster DB akan di-backtracking ke stempel waktu tersebut.

Jika tidak, biasanya terjadi kesalahan. Selain itu, jika Anda mencoba untuk melakukan backtracking klaster DB yang memiliki pencatatan log biner aktif, kesalahan biasanya terjadi kecuali jika Anda telah memilih untuk memaksa backtrack untuk terjadi. Memaksa backtrack agar terjadi dapat mengganggu operasi lain yang menggunakan pencatatan log biner.

> **Important**
>
> Backtracking tidak menghasilkan entri binlog untuk perubahan yang dibuatnya. Jika Anda mengaktifkan pencatatan log biner untuk klaster DB, backtracking mungkin tidak kompatibel dengan implementasi binlog Anda.

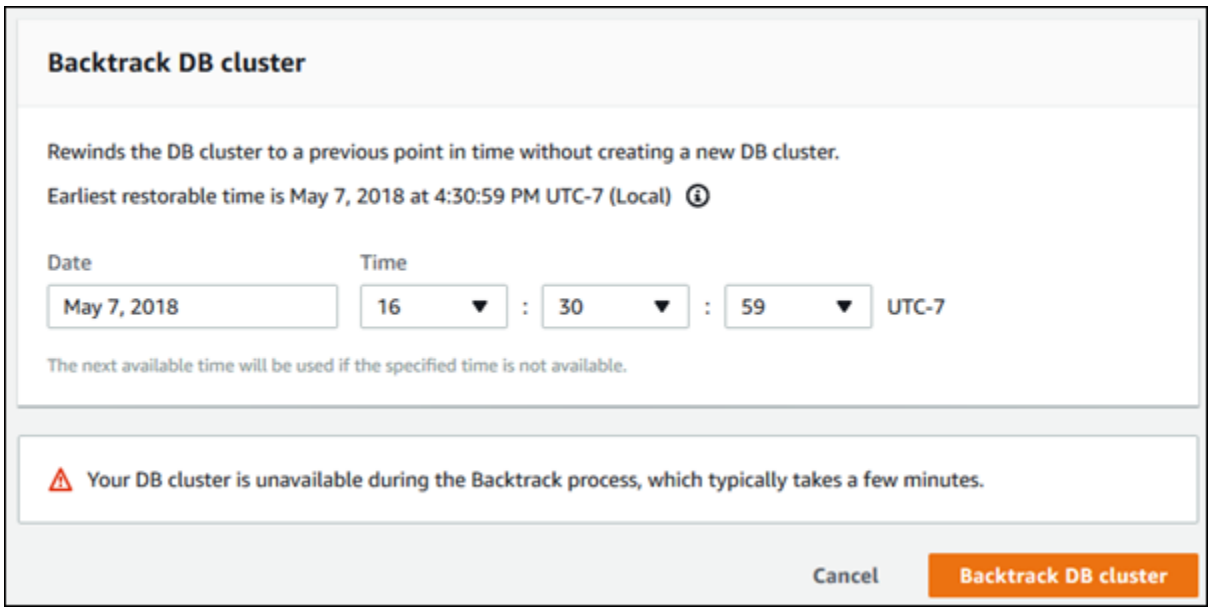
> **Note**
>
> Untuk klon basis data, Anda tidak dapat melakukan backtracking klaster DB lebih awal dari tanggal dan waktu ketika klon dibuat. Untuk informasi selengkapnya tentang kloning basis data, lihat Mengkloning volume untuk klaster DB Amazon Aurora.

Konsol

Prosedur berikut menjelaskan cara melakukan operasi backtrack untuk klaster DB menggunakan konsol.

Untuk melakukan operasi backtrack menggunakan konsol

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans.
3. Pilih instans primer untuk klaster DB yang ingin Anda backtrack.
4. Untuk Tindakan, pilih Klaster DB Backtrack.
5. Di halaman Klaster DB Backtrack, masukkan stempel waktu backtrack untuk melakukan backtracking klaster DB.




6. Pilih Klaster DB Backtrack.

AWS CLI

Prosedur berikut menjelaskan cara melakukan backtracking klaster DB menggunakan AWS CLI.

Untuk mundur cluster DB menggunakan AWS CLI

- Panggil backtrack-db-cluster AWS CLIperintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--backtrack-to`— Cap waktu backtrack untuk mundur cluster DB ke, ditentukan dalam ISO format 8601.

  Contoh berikut melakukan backtracking klaster DB `sample-cluster` ke 19 Maret 2018, pukul 10.00.

  Untuk Linux, macOS, atau Unix:

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-to 2018-03-19T10:00:00+00:00
```

  Untuk Windows:

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

RDS API

Untuk mundur cluster DB menggunakan Amazon RDSAPI, gunakan acktrackDBCluster operasi B. Operasi ini melakukan backtracking klaster DB yang ditentukan dalam nilai `DBClusterIdentifier` ke waktu yang ditentukan.

## Memantau backtracking untuk cluster Aurora My DB SQL

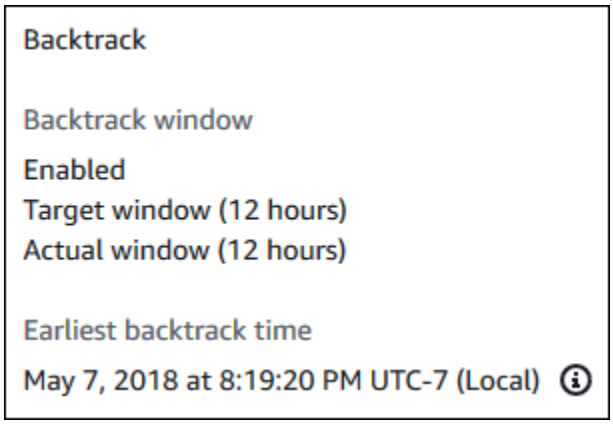
Anda dapat melihat informasi backtracking dan memantau metrik backtracking untuk klaster DB.

Konsol

Untuk melihat informasi backtracking dan memantau metrik backtracking menggunakan konsol

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih nama klaster DB untuk membuka informasi tentangnya.

   Informasi backtrack ada di bagian Backtrack.

   Saat backtrack diaktifkan, informasi berikut tersedia:

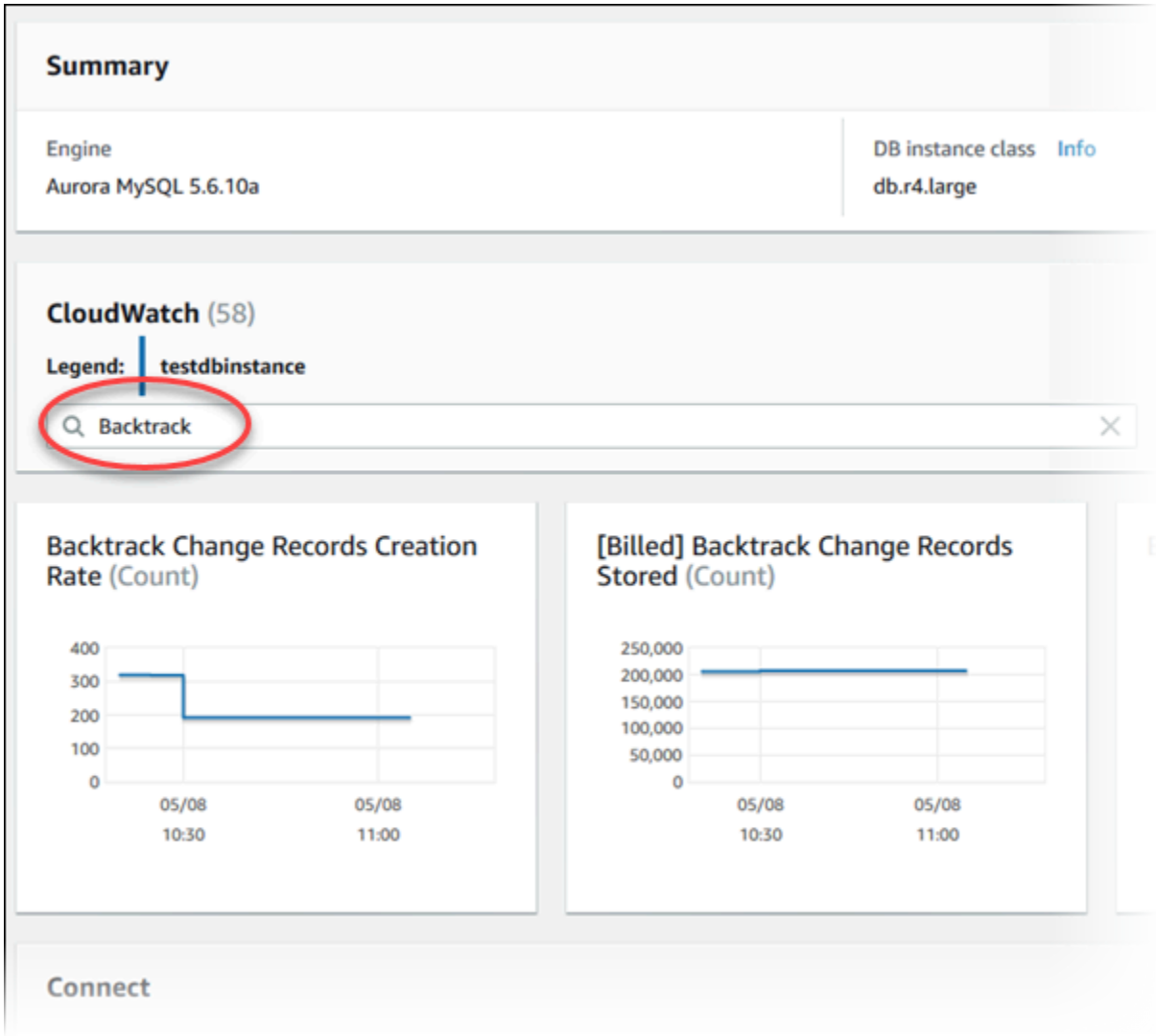
- Jendela target – Jumlah waktu saat ini yang ditentukan untuk jendela backtrack target. Target adalah jumlah waktu maksimum yang Anda inginkan untuk melakukan backtracking jika ada penyimpanan yang memadai.
- Jendela aktual – Jumlah waktu aktual yang Anda inginkan untuk dapat melakukan backtracking, yang dapat lebih kecil daripada jendela backtrack target. Jendela backtrack aktual didasarkan pada beban kerja Anda dan penyimpanan yang tersedia untuk mempertahankan catatan perubahan backtrack.
- Waktu backtrack paling awal – Waktu backtrack paling awal yang memungkinkan untuk klaster DB. Anda tidak dapat melakukan backtracking klaster DB ke waktu sebelum waktu yang ditampilkan.

4. Lakukan hal berikut untuk melihat metrik backtracking untuk klaster DB:

   a. Di panel navigasi, pilih Instans.

   b. Pilih nama instans primer untuk klaster DB untuk menampilkan detailnya.

   c. Di CloudWatchbagian ini, ketik **`Backtrack`** ke dalam CloudWatchkotak untuk hanya menampilkan metrik Backtrack.




Metrik berikut ditampilkan:

- Laju Pembuatan Catatan Perubahan Backtrack (Hitungan) – Metrik ini menunjukkan jumlah catatan perubahan backtrack yang dibuat selama lima menit untuk klaster DB Anda. Anda dapat menggunakan metrik ini untuk memperkirakan biaya backtrack untuk jendela backtrack target Anda.
- [Ditagih] Catatan Perubahan Backtrack Disimpan (Hitungan) – Metrik ini menunjukkan jumlah aktual RDS perubahan backtrack yang digunakan oleh klaster DB Anda.
- Jendela Backtrack Aktual (Menit) – Metrik ini menunjukkan apakah terdapat perbedaan antara jendela backtrack target dan jendela backtrack aktual. Sebagai contoh, jika jendela backtrack target Anda adalah 2 jam (120 menit), dan metrik ini menunjukkan bahwa jendela backtrack aktual adalah 100 menit, berarti jendela backtrack aktual lebih kecil dari target.

- Peringatan Jendela Backtrack (Hitungan) – Metrik ini menunjukkan jumlah berapa kali jendela backtrack aktual lebih kecil daripada jendela backtrack target untuk periode waktu tertentu.

> **Note**
>
> Metrik berikut mungkin mengalami lag dari waktu saat ini:
>
> - Laju Pembuatan Catatan Perubahan Backtrack (Hitungan)
> - [Ditagih] Catatan Perubahan Backtrack Disimpan (Hitungan)

AWS CLI

Prosedur berikut menjelaskan cara menampilkan informasi backtrack untuk klaster DB menggunakan AWS CLI.

Untuk melihat informasi backtrack untuk cluster DB menggunakan AWS CLI

- Panggil describe-db-clusters AWS CLIperintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.

  Contoh berikut mencantumkan informasi backtrack untuk `sample-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  Untuk Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS API

Untuk melihat informasi backtrack untuk cluster DB menggunakan Amazon RDSAPI, gunakan escribeDBClusters operasi D. Operasi ini menampilkan informasi backtrack untuk klaster DB yang ditentukan dalam nilai `DBClusterIdentifier`.

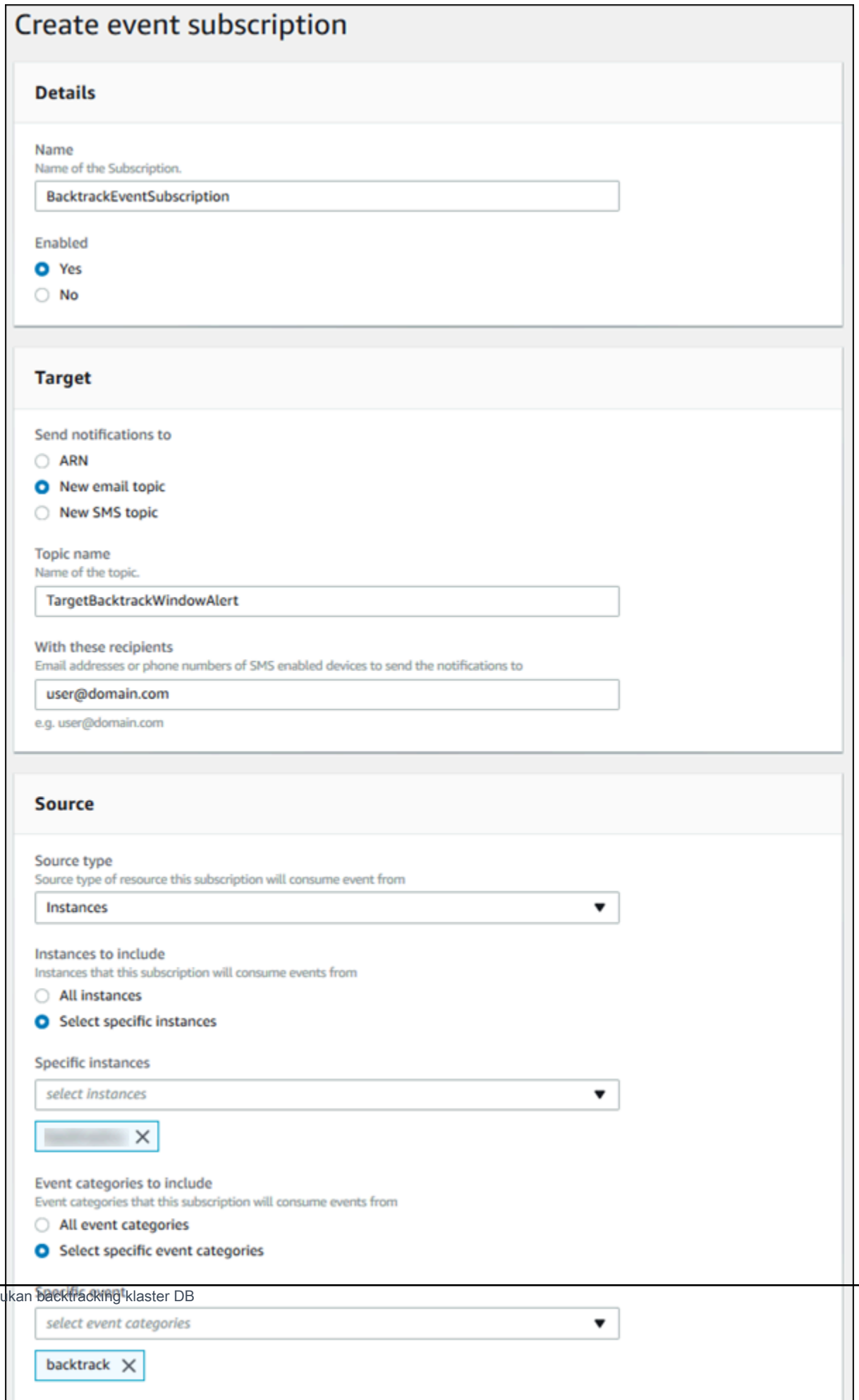
## Berlangganan peristiwa backtrack dengan konsol

Prosedur berikut menjelaskan cara berlangganan peristiwa backtrack menggunakan konsol. Peristiwa tersebut mengirimkan notifikasi email atau pesan teks saat jendela backtrack aktual Anda lebih kecil dari jendela backtrack target Anda.

Untuk melihat informasi backtrack menggunakan konsol

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Langganan peristiwa.
3. Pilih Buat langganan peristiwa.
4. Di kotak Nama, ketik nama untuk langganan peristiwa, dan pastikan bahwa Ya dipilih untuk Diaktifkan.
5. Di bagian Target, pilih Topik email baru.
6. Untuk Nama topik, ketik nama untuk topik, dan untuk Dengan penerima ini, masukkan alamat email atau nomor telepon untuk menerima notifikasi.
7. Di bagian Sumber, pilih Instans untuk Jenis sumber.
8. Untuk Instans untuk disertakan, klik Pilih instans tertentu, dan pilih instans DB Anda.
9. Untuk Kategori peristiwa untuk disertakan, klik Pilih kategori peristiwa tertentu, dan pilih backtrack.

   Halaman Anda akan terlihat serupa dengan halaman berikut.


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
select event categories
backtrack

10. Pilih Buat.

## Mengambil backtrack yang ada

Anda dapat mengambil informasi tentang backtrack yang ada untuk klaster DB. Informasi ini mencakup pengidentifikasi unik backtrack, rentang tanggal dan waktu backtracking, tanggal dan waktu backtrack diminta, dan status backtrack saat ini.

> **Note**
>
> Saat ini, Anda tidak dapat mengambil backtrack yang ada menggunakan konsol.

### AWS CLI

Prosedur berikut menjelaskan cara mengambil backtrack yang ada untuk cluster DB menggunakan AWS CLI.

Untuk mengambil backtrack yang ada menggunakan AWS CLI

- Panggil describe-db-cluster-backtracks AWS CLIperintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.

  Contoh berikut mengambil backtrack yang ada untuk `sample-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  Untuk Windows:

```
aws rds describe-db-cluster-backtracks ^
    --db-cluster-identifier sample-cluster
```

RDS API

Untuk mengambil informasi tentang backtrack untuk cluster DB menggunakan Amazon RDSAPI, gunakan operasi D escribeDBCluster Backtracks. Operasi ini menampilkan informasi tentang backtrack untuk klaster DB yang ditentukan dalam nilai `DBClusterIdentifier`.

## Menonaktifkan backtracking untuk cluster Aurora My DB SQL

Anda dapat menonaktifkan fitur Backtrack untuk klaster DB.

Konsol

Anda dapat menonaktifkan backtracking untuk klaster DB menggunakan konsol. Setelah Anda menonaktifkan backtracking sepenuhnya untuk sebuah klaster, Anda tidak dapat mengaktifkannya lagi untuk klaster tersebut.

Untuk menonaktifkan fitur Backtrack untuk klaster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih klaster yang ingin Anda ubah, dan pilih Modifikasi.
4. Di bagian Backtrack, pilih Nonaktifkan Backtrack.
5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih salah satu dari berikut ini:
   - Terapkan selama jendela pemeliharaan terjadwal berikutnya – Tunggu untuk menerapkan modifikasi hingga periode pemeliharaan berikutnya.
   - Terapkan segera – Terapkan modifikasi sesegera mungkin.
7. Pilih Ubah Klaster.

AWS CLI

Anda dapat menonaktifkan fitur Backtrack untuk cluster DB menggunakan AWS CLI dengan mengatur jendela target backtrack ke `0` (nol). Setelah Anda menonaktifkan backtracking sepenuhnya untuk sebuah klaster, Anda tidak dapat mengaktifkannya lagi untuk klaster tersebut.

Untuk memodifikasi jendela backtrack target untuk cluster DB menggunakan AWS CLI

- Panggil modify-db-cluster AWS CLIperintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--backtrack-window` – tentukan `0` untuk menonaktifkan backtrack.

  Contoh berikut menonaktifkan fitur Backtrack untuk `sample-cluster` dengan mengatur `--backtrack-window` ke `0`.

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

  Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

RDS API

Untuk menonaktifkan fitur Backtrack untuk cluster DB menggunakan Amazon RDSAPI, gunakan odifyDBCluster operasi M. Atur nilai `BacktrackWindow` ke `0` (nol), dan tentukan klaster DB dalam nilai `DBClusterIdentifier`. Setelah Anda menonaktifkan backtracking sepenuhnya untuk sebuah klaster, Anda tidak dapat mengaktifkannya lagi untuk klaster tersebut.

## Menguji Amazon Aurora My SQL menggunakan kueri injeksi kesalahan

Anda dapat menguji toleransi kesalahan cluster Aurora My SQL DB Anda dengan menggunakan kueri injeksi kesalahan. Kueri injeksi kesalahan dikeluarkan sebagai SQL perintah ke instans Amazon Aurora. Kueri ini memungkinkan Anda menjadwalkan simulasi kemunculan salah satu peristiwa berikut:

- Crash instans DB penulis atau pembaca

- Kegagalan Replika Aurora
- Kegagalan disk
- Kepadatan disk

Ketika kueri injeksi kesalahan menentukan crash, itu memaksa crash instance Aurora SQL My DB. Kueri injeksi kesalahan lainnya menghasilkan simulasi kejadian kegagalan, tetapi tidak menyebabkan kejadian terjadi. Jika Anda mengirim kueri injeksi kesalahan, Anda juga perlu menentukan durasi waktu untuk simulasi peristiwa kegagalan yang akan terjadi.

Anda dapat mengirim kueri injeksi kesalahan ke salah satu instans Replika Aurora dengan menghubungkan ke titik akhir untuk Replika Aurora tersebut. Untuk informasi selengkapnya, lihat Koneksi titik akhir Amazon Aurora.

Untuk menjalankan kueri injeksi kesalahan, diperlukan semua hak akses pengguna master. Untuk informasi selengkapnya, lihat Hak akses akun pengguna master.

### Menguji crash instans

Anda dapat memaksa crash instans Amazon Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM CRASH`.

Untuk kueri injeksi kesalahan, failover tidak akan terjadi. Jika Anda ingin menguji failover, maka Anda dapat memilih tindakan instance Failover untuk cluster DB Anda di RDS konsol, atau menggunakan failover-db-cluster AWS CLI perintah atau operasi F ailoverDBCluster RDSAPI.

Sintaksis

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

Opsi

Kueri injeksi kesalahan ini menggunakan salah satu dari jenis crash berikut:

- **`INSTANCE`**— Kecelakaan database yang SQL kompatibel dengan Saya untuk instans Amazon Aurora disimulasikan.
- **`DISPATCHER`** — Crash dispatcher pada instans penulis untuk klaster DB Aurora disimulasikan. Dispatcher menulis pembaruan ke volume klaster untuk klaster DB Amazon Aurora.
- **`NODE`**— Kerusakan database My SQL -compatible dan dispatcher untuk instance Amazon Aurora disimulasikan. Untuk simulasi injeksi kesalahan ini, cache juga dihapus.

Jenis crash default adalah `INSTANCE`.

## Menguji kegagalan replika Aurora

Anda dapat menyimulasikan kegagalan Replika Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM SIMULATE READ REPLICA FAILURE`.

Kegagalan Replika Aurora akan memblokir semua permintaan dari instans penulis ke Replika Aurora atau semua Replika Aurora dalam klaster DB untuk interval waktu yang ditentukan. Ketika interval waktu selesai, Replika Aurora yang terpengaruh akan secara otomatis disinkronkan dengan instance penulis.

Sintaksis

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

- **`percentage_of_failure`** – Persentase permintaan yang akan diblokir selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada permintaan yang akan diblokir. Jika Anda menentukan 100, semua permintaan akan diblokir.
- failure_type – Jenis kegagalan yang akan disimulasikan. Tentukan `TO ALL` untuk menyimulasikan kegagalan untuk semua Replika Aurora dalam klaster DB. Tentukan `TO` dan nama Replika Aurora untuk menyimulasikan kegagalan Replika Aurora tunggal. Jenis kegagalan default adalah `TO ALL`.
- **`quantity`** – Jumlah waktu untuk menyimulasikan kegagalan Replika Aurora. Interval adalah jumlah yang diikuti oleh unit waktu. Simulasi akan terjadi untuk jumlah unit yang ditentukan tersebut. Misalnya, `20 MINUTE` akan menghasilkan simulasi berjalan selama 20 menit.

> **Note**
>
> Berhati-hatilah saat menentukan interval waktu untuk peristiwa kegagalan Replika Aurora Anda. Jika Anda menentukan interval waktu yang terlalu panjang, dan instans penulis Anda menulis data dalam jumlah besar selama peristiwa kegagalan, maka klaster DB Aurora Anda mungkin menganggap bahwa Replika Aurora telah crash lalu menggantinya.

## Menguji kegagalan disk

Anda dapat menyimulasikan kegagalan disk untuk klaster DB Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM SIMULATE DISK FAILURE`.

Selama simulasi kegagalan disk, klaster DB Aurora secara acak menandai segmen disk sebagai mengalami kesalahan. Permintaan ke segmen tersebut akan diblokir selama durasi simulasi.

Sintaksis

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

- **`percentage_of_failure`** – Persentase disk yang akan ditandai sebagai mengalami kesalahan selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menentukan 0, tidak ada disk yang akan ditandai sebagai mengalami kesalahan. Jika Anda menentukan 100, seluruh disk akan ditandai sebagai mengalami kesalahan.
- **`DISK index`** – Blok data logis data tertentu untuk menyimulasikan peristiwa kegagalan. Jika Anda melampaui rentang blok data logis yang tersedia, Anda akan menerima kesalahan yang memberi tahu Anda nilai indeks maksimum yang dapat Anda tentukan. Untuk informasi selengkapnya, lihat Menampilkan status volume untuk klaster DB Aurora MySQL.
- **`NODE index`** – Simpul penyimpanan spesifik untuk menyimulasikan peristiwa kegagalan. Jika Anda melebihi rentang simpul penyimpanan yang tersedia, Anda akan menerima kesalahan yang memberi tahu Anda nilai indeks maksimum yang dapat Anda tentukan. Untuk informasi selengkapnya, lihat Menampilkan status volume untuk klaster DB Aurora MySQL.
- **`quantity`** – Jumlah waktu untuk menyimulasikan kegagalan disk. Interval adalah jumlah yang diikuti oleh unit waktu. Simulasi akan terjadi untuk jumlah unit yang ditentukan tersebut. Misalnya, `20 MINUTE` akan menghasilkan simulasi berjalan selama 20 menit.

## Menguji kepadatan disk

Anda dapat menyimulasikan kegagalan disk untuk klaster DB Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM SIMULATE DISK CONGESTION`.

Selama simulasi kepadatan disk, klaster DB Aurora secara acak menandai segmen disk sebagai padat. Permintaan ke segmen tersebut akan ditunda antara waktu penundaan minimum dan maksimum yang ditentukan untuk durasi simulasi.

Sintaksis

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

- **`percentage_of_failure`** – Persentase disk yang akan ditandai sebagai padat selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada disk yang ditandai sebagai padat. Jika Anda menetapkan 100, seluruh disk akan ditandai sebagai padat.
- **`DISK index`** Atau **`NODE index`** – Disk atau simpul tertentu untuk menyimulasikan peristiwa kegagalan. Jika Anda melebihi rentang indeks untuk disk atau simpul, Anda akan menerima kesalahan yang memberi tahu Anda nilai indeks maksimum yang dapat Anda tentukan.
- **`minimum`** Dan **`maximum`** – Jumlah penundaan kepadatan minimum dan maksimum, dalam milidetik. Segmen disk yang ditandai sebagai padat akan ditunda selama jumlah waktu acak dalam rentang jumlah minimum dan maksimum milidetik untuk durasi simulasi.
- **`quantity`** – Jumlah waktu untuk menyimulasikan kepadatan disk. Interval adalah jumlah yang diikuti oleh unit waktu. Simulasi akan terjadi untuk jumlah unit waktu yang ditentukan. Misalnya, `20 MINUTE` akan menghasilkan simulasi yang berjalan selama 20 menit.

## Mengubah tabel di Amazon Aurora menggunakan DDL Cepat

Amazon Aurora menyediakan pengoptimalan untuk menjalankan operasi `ALTER TABLE` di tempat, hampir seketika. Operasi ini akan selesai tanpa mengharuskan tabel disalin dan tanpa memiliki dampak besar pada pernyataan DML lainnya. Karena operasi ini tidak menggunakan penyimpanan sementara untuk salinan tabel, hal ini membuat pernyataan DDL praktis bahkan untuk tabel besar pada kelas instans kecil.

Aurora MySQL versi 3 kompatibel dengan fitur MySQL 8.0 yang disebut DDL instan. Aurora MySQL versi 2 menggunakan implementasi berbeda yang disebut DDL Cepat.

Topik

- DDL instan (Aurora MySQL versi 3)
- DDL Cepat (Aurora MySQL versi 2)

## DDL instan (Aurora MySQL versi 3)

Optimisasi yang dilakukan oleh Aurora MySQL versi 3 untuk meningkatkan efisiensi beberapa operasi DDL disebut DDL instan.

Aurora MySQL versi 3 kompatibel dengan DDL instan dari MySQL 8.0 komunitas. Anda melakukan operasi DDL instan dengan menggunakan klausa `ALGORITHM=INSTANT` dengan pernyataan `ALTER TABLE`. Untuk detail sintaksis dan penggunaan DDL instan, lihat ALTER TABLE dan Online DDL Operations dalam dokumentasi MySQL.

Contoh berikut menunjukkan fitur DDL instan. Pernyataan `ALTER TABLE` menambahkan kolom dan mengubah nilai kolom default. Contoh ini mencakup kolom reguler dan virtual, dan tabel reguler dan berpartisi. Pada setiap langkah, Anda dapat melihat hasilnya dengan mengeluarkan pernyataan `SHOW CREATE TABLE` dan `DESCRIBE`.

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

## DDL Cepat (Aurora MySQL versi 2)

Fast DDL di Aurora MySQL adalah optimasi yang dirancang untuk meningkatkan kinerja perubahan skema tertentu, seperti menambahkan atau menjatuhkan kolom, dengan mengurangi waktu henti dan penggunaan sumber daya. Hal ini memungkinkan operasi ini diselesaikan lebih efisien dibandingkan dengan metode DDL tradisional.

> **Important**
>
> Saat ini, Anda harus mengaktifkan mode lab Aurora untuk menggunakan Fast DDL. Untuk informasi tentang mengaktifkan mode lab, lihatMode lab Amazon Aurora MySQL.
> Optimasi Fast DDL awalnya diperkenalkan dalam mode lab pada Aurora MySQL versi 2 untuk meningkatkan efisiensi operasi DDL tertentu. Di Aurora MySQL versi 3, mode lab telah dihentikan, dan Fast DDL telah digantikan oleh fitur MySQL 8.0 Instant DDL.

Di MySQL, banyak operasi bahasa definisi data (DDL) memiliki dampak performa yang signifikan.

Misalnya, anggaplah bahwa Anda menggunakan operasi `ALTER TABLE` untuk menambahkan kolom ke tabel. Bergantung pada algoritma yang ditentukan untuk operasi, operasi ini dapat memerlukan hal-hal berikut:

- Membuat salinan lengkap tabel
- Membuat tabel sementara untuk memproses operasi bahasa manipulasi data (DML) konkuren
- Membuat kembali semua indeks untuk tabel
- Menerapkan kunci tabel sambil menerapkan perubahan DML konkuren
- Memperlambat throughput DML konkuren

Dampak kinerja ini bisa sangat menantang di lingkungan dengan tabel besar atau volume transaksi yang tinggi. Fast DDL membantu mengurangi tantangan ini dengan mengoptimalkan perubahan skema, yang memungkinkan operasi yang lebih cepat dan kurang intensif sumber daya.

Batasan DDL Cepat

Saat ini, DDL Cepat memiliki batasan berikut:

- DDL Cepat hanya mendukung penambahan kolom yang dapat dibuat null, tanpa nilai default, ke akhir tabel yang ada.
- DDL Cepat tidak berfungsi untuk tabel yang dipartisi.

- DDL Cepat tidak berfungsi untuk tabel InnoDB yang menggunakan format baris REDUNDANT.
- DDL Cepat tidak berfungsi untuk tabel dengan indeks pencarian teks lengkap.
- Jika ukuran catatan maksimum yang mungkin untuk operasi DDL terlalu besar, DDL Cepat tidak digunakan. Ukuran catatan terlalu besar jika lebih besar dari setengah ukuran halaman. Ukuran maksimum catatan dihitung dengan menjumlahkan ukuran maksimum semua kolom. Untuk kolom dengan ukuran bervariasi, yang mengikuti standar InnoDB, byte extern tidak disertakan untuk perhitungan.

### Sintaksis DDL Cepat

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

Pernyataan ini mengambil opsi berikut:

- **`tbl_name`** – Nama tabel yang akan diubah.
- **`col_name`** – Nama kolom yang akan ditambahkan.
- **`col_definition`** – Definisi kolom yang akan ditambahkan.

> **Note**
>
> Anda harus menentukan kolom yang dapat dibuat null tanpa nilai default. Jika tidak, DDL Cepat tidak digunakan.

### Contoh DDL Cepat

Contoh berikut menunjukkan percepatan dari operasi DDL Cepat. Contoh SQL pertama menjalankan pernyataan `ALTER TABLE` pada tabel besar tanpa menggunakan DDL Cepat. Operasi ini membutuhkan waktu yang cukup lama. Salah satu contoh CLI ini menunjukkan cara mengaktifkan DDL Cepat untuk klaster. Kemudian, contoh SQL lainnya menjalankan pernyataan `ALTER TABLE` yang sama pada tabel yang identik. Dengan mengaktifkan DDL Cepat, operasinya sangat cepat.

Contoh ini menggunakan tabel `ORDERS` dari tolok ukur TPC-H, yang berisi 150 juta baris. Klaster ini sengaja menggunakan kelas instans yang relatif kecil untuk menunjukkan waktu yang diperlukan pernyataan `ALTER TABLE` saat Anda tidak dapat menggunakan DDL Cepat. Contoh ini membuat klon dari tabel asli yang berisi data identik. Dengan memeriksa pengaturan `aurora_lab_mode`, akan dikonfirmasi bahwa klaster tidak dapat menggunakan DDL Cepat karena mode lab tidak

diaktifkan. Kemudian, pernyataan `ALTER TABLE ADD COLUMN` membutuhkan banyak waktu untuk menambahkan kolom baru di akhir tabel.

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 0 |
+-------------------+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

Contoh ini melakukan persiapan tabel besar yang sama seperti contoh sebelumnya. Namun, Anda tidak bisa begitu saja mengaktifkan mode lab dalam sesi SQL interaktif. Pengaturan tersebut harus diaktifkan dalam grup parameter kustom. Melakukannya membutuhkan peralihan dari `mysql` sesi dan menjalankan beberapa perintah AWS CLI atau menggunakan. AWS Management Console

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

Pengaktifan mode lab untuk klaster memerlukan beberapa pekerjaan dengan grup parameter. Contoh AWS CLI ini menggunakan grup parameter cluster, untuk memastikan bahwa semua instance DB di cluster menggunakan nilai yang sama untuk pengaturan mode lab.

```
$ aws rds create-db-cluster-parameter-group \
```

```
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}

# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

Contoh berikut menunjukkan langkah-langkah yang tersisa setelah perubahan grup parameter berlaku. Contoh ini menguji pengaturan `aurora_lab_mode` untuk memastikan bahwa klaster dapat menggunakan DDL Cepat. Contoh ini kemudian menjalankan pernyataan `ALTER TABLE` untuk menambahkan kolom ke akhir tabel besar lainnya. Kali ini, pernyataan selesai dengan sangat cepat.

```
mysql> select @@aurora_lab_mode;
+------------------+
| @@aurora_lab_mode |
+------------------+
|                1 |
+------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

## Menampilkan status volume untuk klaster DB Aurora MySQL

Di Amazon Aurora, volume klaster DB terdiri dari sekumpulan blok logis. Masing-masing blok merepresentasikan 10 gigabyte penyimpanan yang dialokasikan. Blok-blok ini disebut grup perlindungan.

Data di setiap grup perlindungan direplikasi di enam perangkat penyimpanan fisik, yang disebut simpul penyimpanan. Simpul penyimpanan ini dialokasikan di tiga Zona Ketersediaan (AZ) di Wilayah AWS tempat klaster DB berada. Pada gilirannya, setiap simpul penyimpanan berisi satu atau beberapa blok logis data untuk volume klaster DB. Untuk informasi selengkapnya tentang grup perlindungan dan simpul penyimpanan, lihat Memperkenalkan mesin penyimpanan Aurora dalam Blog Basis Data AWS.

Anda dapat menyimulasikan kegagalan seluruh simpul penyimpanan atau satu blok logis data dalam simpul penyimpanan. Untuk melakukannya, Anda menggunakan pernyataan injeksi kegagalan `ALTER SYSTEM SIMULATE DISK FAILURE`. Untuk pernyataan tersebut, Anda menentukan nilai indeks blok logis data atau simpul penyimpanan tertentu. Namun, jika Anda menentukan nilai indeks yang lebih besar dari jumlah blok logis simpul data atau penyimpanan yang digunakan oleh volume klaster DB, pernyataan tersebut akan menampilkan kesalahan. Untuk informasi selengkapnya tentang kueri injeksi kesalahan, lihat Menguji Amazon Aurora My SQL menggunakan kueri injeksi kesalahan.

Anda dapat menghindari kesalahan tersebut dengan menggunakan pernyataan `SHOW VOLUME STATUS`. Pernyataan ini menghasilkan dua variabel status server, `Disks` dan `Nodes`. Variabel ini merepresentasikan jumlah total blok logis simpul data dan penyimpanan, masing-masing, untuk volume klaster DB.

## Sintaksis

```
SHOW VOLUME STATUS
```

## Contoh

Contoh berikut menggambarkan hasil SHOW VOLUME STATUS yang biasa.

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
+---------------+-------+
```

# Menyesuaikan Aurora MySQL

Peristiwa tunggu dan status thread adalah alat penyesuaian penting untuk Aurora MySQL. Jika Anda dapat mengetahui alasan mengapa sesi menunggu sumber daya dan operasi yang sedang dilakukannya, Anda akan lebih mampu mengurangi bottleneck. Anda dapat menggunakan informasi di bagian ini untuk menemukan kemungkinan penyebab dan tindakan korektif.

Amazon DevOps Guru untuk RDS dapat secara proaktif menentukan apakah basis data Aurora MySQL Anda mengalami kondisi bermasalah yang dapat menyebabkan masalah yang lebih besar nantinya. Amazon DevOps Guru for RDS menerbitkan penjelasan dan rekomendasi untuk tindakan korektif dalam wawasan proaktif. Bagian ini berisi wawasan untuk masalah umum.

> **Important**
>
> Peristiwa tunggu dan status thread di bagian ini dikhususkan untuk Aurora MySQL. Gunakan informasi di bagian ini untuk menyesuaikan Amazon Aurora saja, bukan Amazon RDS for MySQL.
> Beberapa peristiwa tunggu di bagian ini tidak memiliki padanan di versi sumber terbuka dari mesin basis data ini. Peristiwa tunggu lainnya memiliki nama yang sama dengan peristiwa di mesin sumber terbuka, tetapi berperilaku berbeda. Misalnya, penyimpanan Amazon Aurora berfungsi secara berbeda dari penyimpanan sumber terbuka, sehingga peristiwa tunggu terkait penyimpanan menunjukkan kondisi sumber daya yang berbeda.

Topik


- Konsep penting untuk penyesuaian Aurora MySQL
- Menyesuaikan Aurora MySQL dengan peristiwa tunggu
- Menyesuaikan Aurora MySQL dengan status thread
- Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru


## Konsep penting untuk penyesuaian Aurora MySQL

Sebelum Anda menyesuaikan basis data Aurora MySQL Anda, pastikan untuk mempelajari peristiwa tunggu dan status thread dan mengapa itu terjadi. Tinjau juga arsitektur memori dan disk dasar Aurora MySQL saat menggunakan mesin penyimpanan InnoDB. Untuk diagram arsitektur yang bermanfaat, lihat Panduan Referensi MySQL.

Topik


- Peristiwa tunggu Aurora MySQL
- Status thread Aurora MySQL
- Memori Aurora MySQL
- Proses Aurora MySQL


## Peristiwa tunggu Aurora MySQL

Peristiwa tunggu menunjukkan sumber daya yang sedang ditunggu oleh sesi. Misalnya, peristiwa tunggu `io/socket/sql/client_connection` menunjukkan bahwa thread sedang dalam proses untuk menangani koneksi baru. Sumber daya biasa yang ditunggu oleh sesi meliputi yang berikut:

- Akses thread tunggal ke buffer, misalnya, saat sesi mencoba memodifikasi buffer
- Baris yang saat ini dikunci oleh sesi lain
- Pembacaan file data
- Penulisan file data

Misalnya, untuk memenuhi kueri, sesi mungkin melakukan pemindaian tabel lengkap. Jika data belum ada dalam memori, sesi akan menunggu I/O disk selesai. Ketika buffer dibaca ke dalam memori, sesi mungkin perlu menunggu karena sesi lain mengakses buffer yang sama. Basis data mencatat peristiwa tunggu dengan menggunakan peristiwa tunggu standar. Peristiwa tersebut dikelompokkan ke dalam kategori.

Peristiwa tunggu tidak dengan sendirinya menunjukkan masalah performa. Misalnya, jika data yang diminta tidak ada dalam memori, data perlu dibaca dari disk. Jika satu sesi mengunci baris untuk pembaruan, sesi lain akan menunggu baris tersebut dibuka sehingga dapat memperbaruinya. Commit perlu menunggu penulisan ke file log selesai. Peristiwa tunggu merupakan bagian integral dari fungsi normal basis data.

Sejumlah besar peristiwa tunggu biasanya menunjukkan masalah performa. Dalam kasus seperti itu, Anda dapat menggunakan data peristiwa tunggu untuk menentukan tempat sesi menghabiskan waktu. Misalnya, jika laporan yang biasanya berjalan dalam hitungan menit sekarang berjalan selama berjam-jam, Anda dapat mengidentifikasi peristiwa tunggu yang berkontribusi paling besar terhadap total waktu tunggu. Jika Anda dapat menentukan penyebab peristiwa tunggu teratas, terkadang Anda dapat membuat perubahan yang meningkatkan performa. Misalnya, jika sesi Anda menunggu baris yang telah dikunci oleh sesi lain, Anda dapat mengakhiri sesi penguncian ini.

## Status thread Aurora MySQL

Status thread umum adalah nilai `State` yang terkait dengan pemrosesan kueri umum. Misalnya, status thread `sending data` menunjukkan bahwa thread membaca dan memfilter baris untuk kueri guna menentukan set hasil yang benar.

Anda dapat menggunakan status thread untuk menyesuaikan Aurora MySQL dengan cara yang mirip dengan cara Anda menggunakan peristiwa tunggu. Misalnya, kemunculan `sending data` yang sering biasanya menunjukkan bahwa kueri tidak menggunakan indeks. Untuk informasi selengkapnya tentang status thread, lihat General Thread States dalam Panduan Referensi MySQL.

Saat Anda menggunakan Wawasan Performa, salah satu kondisi berikut ini akan berlaku:

- Skema Performa diaktifkan – Aurora MySQL menunjukkan peristiwa tunggu, bukan status thread.
- Skema Performa tidak diaktifkan – Aurora MySQL menunjukkan status thread.

Sebaiknya konfigurasikan Skema Performa untuk manajemen otomatis. Skema Performa memberikan wawasan tambahan dan alat yang lebih baik untuk menyelidiki potensi masalah performa. Untuk informasi selengkapnya, lihat Ikhtisar Skema Kinerja untuk Performance Insights di Aurora.

## Memori Aurora MySQL

Di Aurora MySQL, area memori yang paling penting adalah pool buffer dan log buffer.

Topik

- Pool buffer

### Pool buffer

Pool buffer adalah area memori bersama tempat Aurora MySQL menyimpan tabel dan data indeks. Kueri dapat mengakses data yang sering digunakan langsung dari memori tanpa membaca dari disk.

Pool buffer disusun sebagai daftar halaman yang ditautkan. Halaman dapat menyimpan beberapa baris. Aurora MySQL menggunakan algoritma least recently used (LRU) untuk mengeluarkan halaman dari pool berdasarkan umurnya.

Untuk informasi selengkapnya, lihat Buffer Pool dalam Panduan Referensi MySQL.

## Proses Aurora MySQL

Aurora MySQL menggunakan model proses yang sangat berbeda dari Aurora PostgreSQL.

Topik

- Server MySQL (mysqld)
- Thread
- Pool thread

Server MySQL (mysqld)

Server MySQL adalah proses sistem operasi tunggal bernama mysqld. Server MySQL tidak memunculkan proses tambahan. Dengan demikian, basis data Aurora MySQL menggunakan mysqld untuk melakukan sebagian besar pekerjaannya.

Ketika server MySQL dimulai, server tersebut akan mendengarkan koneksi jaringan dari klien MySQL. Ketika klien terhubung ke basis data, mysqld membuka thread.

Thread

Thread pengelola koneksi mengaitkan setiap koneksi klien dengan thread khusus. Thread ini mengelola autentikasi, menjalankan pernyataan, dan memberikan hasil ke klien. Pengelola koneksi membuat thread baru jika diperlukan.

Cache thread adalah kumpulan thread yang tersedia. Ketika koneksi berakhir, MySQL mengembalikan thread ke cache thread jika cache tidak penuh. Variabel sistem `thread_cache_size` menentukan ukuran cache thread.

Pool thread

Pool thread terdiri dari sejumlah grup thread. Setiap grup mengelola satu set koneksi klien. Ketika klien terhubung ke basis data, pool thread menetapkan koneksi ke grup thread dengan cara round-robin. Pool thread memisahkan koneksi dan thread. Tidak ada relasi tetap antara koneksi dan thread yang menjalankan pernyataan yang diterima dari koneksi tersebut.

# Menyesuaikan Aurora MySQL dengan peristiwa tunggu

Tabel berikut merangkum peristiwa tunggu Aurora MySQL yang paling sering menunjukkan masalah performa. Peristiwa tunggu berikut adalah bagian dari daftar dalam Aurora Acara tunggu saya SQL.

| Peristiwa tunggu | Deskripsi |
| --- | --- |
| cpu | Peristiwa ini terjadi ketika thread aktif di CPU atau sedang menunggu CPU. |
| io/aurora_redo_log_flush | Peristiwa ini terjadi ketika sesi akan menulis data yang persisten ke penyimpanan Aurora. |
| io/aurora_respond_to_client | Peristiwa ini terjadi ketika thread menunggu untuk memberikan set hasil ke klien. |
| io/redo_log_flush | Peristiwa ini terjadi ketika sesi akan menulis data yang persisten ke penyimpanan Aurora. |
| io/socket/sql/client_koneksi | Peristiwa ini terjadi ketika thread sedang dalam proses untuk menangani koneksi baru. |
| io/table/sql/handler | Peristiwa ini terjadi ketika pekerjaan telah didelegasikan ke mesin penyimpanan. |
| synch/cond/innodb/row_lock_tunggu | Peristiwa ini terjadi ketika satu sesi telah mengunci baris untuk pembaruan, dan sesi lain mencoba memperbarui baris yang sama. |
| synch/cond/innodb/row_lock_wait_cond | Peristiwa ini terjadi ketika satu sesi telah mengunci baris untuk pembaruan, dan sesi lain mencoba memperbarui baris yang sama. |
| synch/cond/sql/MDL_konteks:: COND _wait_status | Peristiwa ini terjadi ketika ada thread yang menunggu kunci metadata tabel. |
| synch/mutex/innodb/aurora_lock_thread_slot_futex | Peristiwa ini terjadi ketika satu sesi telah mengunci baris untuk pembaruan, dan sesi lain mencoba memperbarui baris yang sama. |
| synch/mutex/innodb/buf_pool_mutex | Peristiwa ini terjadi ketika thread telah mendapat kunci pada pool buffer InnoDB untuk mengakses halaman dalam memori. |

| Peristiwa tunggu | Deskripsi |
| --- | --- |
| synch/mutex/innodb/fil_system_mutex | Peristiwa ini terjadi ketika sesi menunggu untuk mengakses cache memori ruang tabel. |
| synch/mutex/innodb/trx_sys_mutex | Peristiwa ini terjadi ketika ada aktivitas basis data tinggi dengan sejumlah besar transaksi. |
| synch/sxlock/innodb/hash_table_locks | Peristiwa ini terjadi ketika halaman yang tidak ditemukan di pool buffer harus dibaca dari file. |

### cpu

Peristiwa `cpu` tunggu terjadi ketika utas aktif CPU atau sedang menungguCPU.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

#### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora SQL Versi saya 2 dan 3

#### Konteks

Untuk setiap vCPU, koneksi dapat berjalan bekerja pada iniCPU. Dalam beberapa situasi, jumlah koneksi aktif yang siap dijalankan lebih tinggi dari jumlahvCPUs. Ketidakseimbangan ini mengakibatkan koneksi menunggu CPU sumber daya. Jika jumlah koneksi aktif tetap lebih tinggi secara konsisten daripada jumlahvCPUs, maka instans Anda mengalami CPU pertengkaran. Pertentangan tersebut menyebabkan peristiwa tunggu `cpu` terjadi.

> **Note**
>
> Metrik Performance Insights untuk CPU adalah. `DBLoadCPU` Nilai untuk `DBLoadCPU` dapat berbeda dari nilai untuk CloudWatch metrik`CPUUtilization`. Metrik terakhir dikumpulkan dari HyperVisor untuk instance database.

Metrik Performance Insights OS memberikan informasi terperinci tentang pemanfaatan. CPU Misalnya, Anda dapat menampilkan metrik berikut:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Performance Insights melaporkan CPU penggunaan oleh mesin database sebagai. `os.cpuUtilization.nice.avg`

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini terjadi lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

- Kueri analitik
- Transaksi bersamaan yang tinggi
- Transaksi yang berjalan lama
- Peningkatan mendadak dalam jumlah koneksi, yang dikenal sebagai login storm
- Peningkatan peralihan konteks

Tindakan

Jika peristiwa tunggu `cpu` mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Tanggapi peristiwa ini hanya saat performa menurun.

Bergantung pada penyebab peningkatan CPU pemanfaatan, pertimbangkan strategi berikut:

- Tingkatkan CPU kapasitas tuan rumah. Pendekatan ini biasanya hanya memberikan bantuan sementara.
- Identifikasi kueri teratas untuk pengoptimalan potensial.
- Arahkan ulang sebagian beban kerja hanya-baca ke simpul pembaca jika berlaku.

Topik

- Mengidentifikasi sesi atau kueri yang menyebabkan masalah
- Menganalisis dan mengoptimalkan beban CPU kerja yang tinggi

Mengidentifikasi sesi atau kueri yang menyebabkan masalah

Untuk menemukan sesi dan kueri, lihat SQL tabel Teratas di Performance Insights untuk pernyataan SQL yang memiliki beban tertinggi. CPU Untuk informasi selengkapnya, lihat Menganalisis metrik dengan dasbor Wawasan Performa.

Biasanya, satu atau dua SQL pernyataan mengkonsumsi sebagian besar CPU siklus. Konsentrasikan upaya Anda pada pernyataan ini. Misalkan instans DB Anda memiliki 2 vCPUs dengan beban DB 3,1 sesi aktif rata-rata (AAS), semuanya dalam CPU keadaan. Dalam hal ini, instance Anda CPU terikat. Pertimbangkan strategi berikut:

- Tingkatkan ke kelas instance yang lebih besar dengan lebih banyakvCPUs.
- Setel kueri Anda untuk memiliki CPU beban yang lebih rendah.

Dalam contoh ini, SQL kueri teratas memiliki beban DB 1,5AAS, semuanya dalam CPU keadaan. SQLPernyataan lain memiliki beban 0,1 di CPU negara bagian. Dalam contoh ini, jika Anda menghentikan SQL pernyataan beban terendah, Anda tidak mengurangi beban database secara signifikan. Namun, jika Anda mengoptimalkan dua kueri beban tinggi menjadi dua kali lebih efisien, Anda menghilangkan kemacetanCPU. Jika Anda mengurangi CPU beban 1,5 hingga 50 AAS persen, AAS untuk setiap pernyataan berkurang menjadi 0,75. Total beban DB yang dihabiskan CPU sekarang 1,6AAS. Nilai ini berada di bawah CPU garis v maksimum 2.0.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights. Lihat juga artikel AWS Support Bagaimana cara memecahkan masalah dan mengatasi CPU pemanfaatan tinggi di Amazon RDS untuk instans Saya? SQL .

Menganalisis dan mengoptimalkan beban CPU kerja yang tinggi

Setelah Anda mengidentifikasi kueri atau kueri yang meningkatkan CPU penggunaan, Anda dapat mengoptimalkannya atau mengakhiri koneksi. Contoh berikut menunjukkan cara mengakhiri koneksi.

```
CALL mysql.rds_kill(processID);
```

Untuk informasi selengkapnya, lihat mysql.rds_kill.

Jika Anda mengakhiri sesi, tindakan tersebut dapat memicu rollback yang panjang.

Mengikuti panduan untuk mengoptimalkan kueri

Untuk mengoptimalkan kueri, pertimbangkan panduan berikut:

- Jalankan pernyataan `EXPLAIN`.

  Perintah ini menunjukkan setiap langkah yang terlibat dalam menjalankan kueri. Untuk informasi selengkapnya, lihat Mengoptimalkan Kueri dengan EXPLAIN di dokumentasi SayaSQL.
- Jalankan pernyataan `SHOW PROFILE`.

  Gunakan pernyataan ini untuk meninjau detail profil yang dapat menunjukkan penggunaan sumber daya untuk pernyataan yang dijalankan selama sesi saat ini. Untuk informasi selengkapnya, lihat SHOWPROFILEPernyataan dalam SQL dokumentasi Saya.
- Jalankan pernyataan `ANALYZE TABLE`.

  Gunakan pernyataan ini untuk menyegarkan statistik indeks untuk tabel yang diakses oleh kueri dengan CPU konsumsi tinggi. Dengan menganalisis pernyataan, Anda dapat membantu pengoptimal memilih rencana eksekusi yang sesuai. Untuk informasi selengkapnya, lihat ANALYZETABLEPernyataan dalam SQL dokumentasi Saya.

Ikuti pedoman untuk meningkatkan CPU penggunaan

Untuk meningkatkan CPU penggunaan dalam instance database, ikuti panduan ini:

- Pastikan semua kueri telah menggunakan indeks yang tepat.
- Cari tahu apakah Anda dapat menggunakan kueri paralel Aurora. Anda dapat menggunakan teknik ini untuk mengurangi CPU penggunaan pada node kepala dengan menekan pemrosesan fungsi, pemfilteran baris, dan proyeksi kolom untuk klausa. `WHERE`

- Cari tahu apakah jumlah SQL eksekusi per detik memenuhi ambang batas yang diharapkan.
- Cari tahu apakah pemeliharaan indeks atau pembuatan indeks baru membutuhkan CPU siklus yang dibutuhkan oleh beban kerja produksi Anda. Jadwalkan aktivitas pemeliharaan di luar waktu aktivitas puncak.
- Cari tahu apakah Anda dapat menggunakan partisi untuk membantu mengurangi set data kueri. Untuk informasi selengkapnya, lihat posting blog Cara merencanakan dan mengoptimalkan Amazon Aurora dengan SQL kompatibilitas Saya untuk beban kerja terkonsolidasi.

Memeriksa connection storm

Jika `DBLoadCPU` metriknya tidak terlalu tinggi, tetapi `CPUUtilization` metriknya tinggi, penyebab CPU pemanfaatan yang tinggi terletak di luar mesin database. Contoh klasik adalah connection storm.

Periksa apakah kondisi berikut benar:

- Ada peningkatan dalam metrik Performance Insights dan `CPUUtilization` metrik Amazon CloudWatch`DatabaseConnections`.
- Jumlah utas di CPU lebih besar dari jumlahvCPUs.

Jika kondisi di atas benar, coba kurangi jumlah koneksi basis data. Misalnya, Anda dapat menggunakan kumpulan koneksi seperti RDS Proxy. Untuk mempelajari praktik terbaik pengelolaan dan penskalaan koneksi yang efektif, lihat whitepaper Amazon Aurora My SQL DBA Handbook for Connection Management.

## io/aurora_redo_log_flush

Peristiwa `io/aurora_redo_log_flush` terjadi ketika sesi menulis data persisten ke penyimpanan Amazon Aurora.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 2 SQL

Konteks

`io/aurora_redo_log_flush`Acara ini untuk operasi tulisinput/output (I/O) di Aurora My. SQL

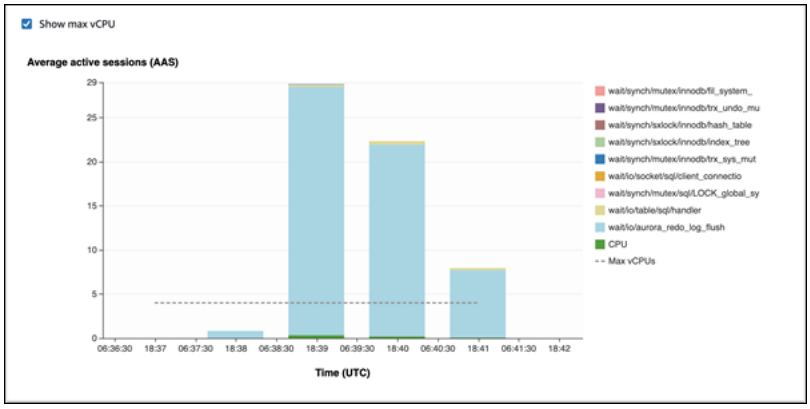
> **Note**
>
> Di Aurora My SQL versi 3, acara tunggu ini bernama io/redo_log_flush.

Kemungkinan penyebab peningkatan peristiwa tunggu

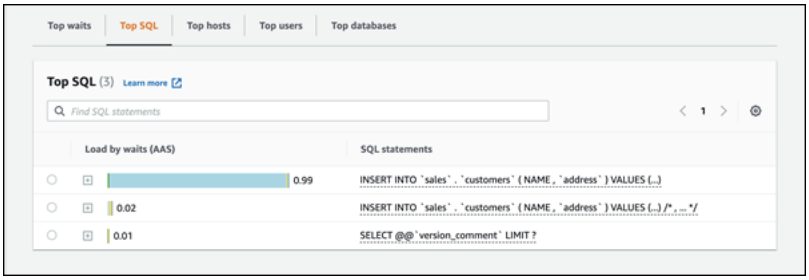
Untuk persistensi data, commit memerlukan penulisan durabel ke penyimpanan yang stabil. Jika basis data terlalu banyak melakukan commit, terdapat peristiwa tunggu pada operasi I/O tulis, yakni peristiwa tunggu `io/aurora_redo_log_flush`.

Dalam contoh berikut, 50.000 catatan dimasukkan ke dalam cluster Aurora My SQL DB menggunakan kelas instans db.r5.xlarge DB:

- Pada contoh pertama, setiap sesi menyisipkan 10.000 catatan baris demi baris. Secara default, jika perintah bahasa manipulasi data (DML) tidak ada dalam transaksi, Aurora My SQL menggunakan commit implisit. Autocommit diaktifkan. Ini berarti terdapat commit untuk setiap penyisipan baris. Wawasan Performa menunjukkan bahwa koneksi menghabiskan sebagian besar waktunya menunggu peristiwa tunggu `io/aurora_redo_log_flush`.

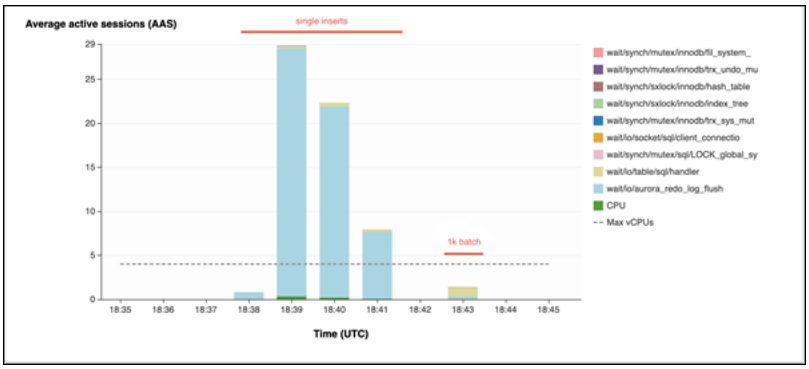



  Hal ini disebabkan oleh pernyataan sisipan sederhana yang digunakan.




Diperlukan waktu 3,5 menit untuk menyisipkan 50.000 catatan.

- Dalam contoh kedua, sisipan dibuat dalam 1.000 batch, yakni setiap koneksi melakukan 10 commit, bukan 10.000. Wawasan Performa menunjukkan bahwa koneksi tidak menghabiskan sebagian besar waktunya untuk peristiwa tunggu `io/aurora_redo_log_flush`.




Diperlukan waktu 4 detik untuk menyisipkan 50.000 catatan.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

### Topik

- Mengidentifikasi sesi dan kueri yang bermasalah
- Mengelompokkan operasi tulis
- Menonaktifkan autocommit
- Menggunakan transaksi
- Menggunakan batch

### Mengidentifikasi sesi dan kueri yang bermasalah

Jika instans DB Anda mengalami kemacetan, langkah pertama adalah menemukan sesi dan kueri yang menjadi penyebabnya. Untuk posting Blog AWS Database yang berguna, lihat Menganalisis SQL Beban Kerja Saya Amazon Aurora dengan Performance Insights.

Untuk mengidentifikasi sesi dan kueri yang menyebabkan kemacetan:

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB Anda.
4. Di Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Kueri di bagian atas daftar adalah penyebab beban tertinggi pada basis data.

### Mengelompokkan operasi tulis

Contoh berikut memicu peristiwa tunggu `io/aurora_redo_log_flush`. (Autocommit diaktifkan.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

Dalam upaya mengurangi waktu yang dihabiskan untuk menunggu peristiwa tunggu `io/aurora_redo_log_flush`, kelompokkan operasi tulis secara logis ke dalam satu commit guna mengurangi panggilan persisten ke penyimpanan.

Menonaktifkan autocommit

Nonaktifkan autocommit sebelum membuat perubahan besar yang tidak ada dalam transaksi, seperti yang ditunjukkan pada contoh berikut.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

Menggunakan transaksi

Anda dapat menggunakan transaksi, seperti yang ditunjukkan pada contoh berikut.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

Menggunakan batch

Anda dapat membuat perubahan dalam batch, seperti yang ditunjukkan pada contoh berikut. Namun, menggunakan batch yang terlalu besar dapat menyebabkan masalah kinerja, terutama pada replika baca atau saat melakukan point-in-time recovery ()PITR.

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

## io/aurora_respond_to_client

Peristiwa `io/aurora_respond_to_client` terjadi ketika thread menunggu untuk mengembalikan set hasil ke klien.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 2 SQL

### Konteks

Peristiwa `io/aurora_respond_to_client` menunjukkan bahwa thread menunggu untuk mengembalikan set hasil ke klien.

Pemrosesan kueri selesai, dan hasilnya dikembalikan ke klien aplikasi. Namun, karena bandwidth jaringan pada klaster DB tidak cukup, thread menunggu untuk mengembalikan set hasil.

### Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `io/aurora_respond_to_client` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Kelas instans DB tidak cukup untuk beban kerja

Kelas instans DB yang digunakan oleh klaster DB tidak memiliki bandwidth jaringan yang diperlukan untuk memproses beban kerja secara efisien.

Set hasil besar

Terjadi peningkatan ukuran set hasil yang dikembalikan karena kueri mengembalikan jumlah baris yang lebih tinggi. Set hasil yang lebih besar menghabiskan lebih banyak bandwidth jaringan.

Peningkatan beban pada klien

Mungkin ada CPU tekanan, tekanan memori, atau saturasi jaringan pada klien. Peningkatan beban pada klien menunda penerimaan data dari cluster Aurora SQL My DB.

Peningkatan latensi jaringan

Mungkin ada peningkatan latensi jaringan antara cluster Aurora SQL My DB dan klien. Latensi jaringan yang lebih tinggi akan menambah waktu yang diperlukan klien untuk menerima data.

### Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Menskalakan kelas instans DB
- Memeriksa beban kerja untuk hasil yang tidak terduga
- Mendistribusikan beban kerja dengan instans pembaca
- Gunakan RESULT pengubah SQL BUFFER _ _

#### Mengidentifikasi sesi dan kueri penyebab peristiwa

Anda dapat menggunakan Wawasan Performa untuk menampilkan kueri yang diblokir oleh peristiwa tunggu `io/aurora_respond_to_client`. Biasanya, basis data dengan beban sedang hingga

signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan SQL kueri yang bertanggung jawab atas beban tinggi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting Blog Database Menganalisis AWS Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Menskalakan kelas instans DB

Periksa peningkatan nilai CloudWatch metrik Amazon yang terkait dengan throughput jaringan, seperti `NetworkReceiveThroughput` dan. `NetworkTransmitThroughput` Jika bandwidth jaringan kelas instans DB tercapai, Anda dapat menskalakan kelas instans DB yang digunakan oleh klaster DB dengan memodifikasi klaster DB tersebut. Kelas instans DB dengan bandwidth jaringan yang lebih besar mengembalikan data ke klien secara lebih efisien.

Untuk informasi tentang memantau CloudWatch metrik Amazon, lihatMelihat metrik di konsol Amazon RDS. Untuk informasi tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB. Untuk informasi tentang mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Memeriksa beban kerja untuk hasil yang tidak terduga

Periksa beban kerja pada klaster DB dan pastikan beban kerja tidak memberikan hasil yang tidak terduga. Misalnya, mungkin terdapat kueri yang mengembalikan jumlah baris lebih tinggi dari yang

diperkirakan. Dalam hal ini, Anda dapat menggunakan metrik penghitung Wawasan Performa seperti `Innodb_rows_read`. Untuk informasi selengkapnya, lihat Metrik penghitung Wawasan Performa.

Mendistribusikan beban kerja dengan instans pembaca

Anda dapat mendistribusikan beban kerja hanya-baca dengan Aurora Replicas. Anda dapat menskalakan secara horizontal dengan menambahkan lebih banyak Aurora Replicas. Dengan demikian, Anda dapat memperoleh peningkatan batas throttling untuk bandwidth jaringan. Untuk informasi selengkapnya, lihat Klaster DB Amazon Aurora.

Gunakan RESULT pengubah SQL BUFFER _ _

Anda dapat menambahkan pengubah `SQL_BUFFER_RESULT` ke pernyataan `SELECT` untuk memaksakan hasilnya ke tabel sementara sebelum dikembalikan ke klien. Pengubah ini dapat membantu mengatasi masalah performa saat kunci InnoDB tidak dibebaskan karena kueri berada dalam status tunggu `io/aurora_respond_to_client`. Untuk informasi selengkapnya, lihat SELECTPernyataan dalam SQL dokumentasi Saya.

## io/redo_log_flush

Peristiwa `io/redo_log_flush` terjadi ketika sesi menulis data persisten ke penyimpanan Amazon Aurora.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 3 SQL

Konteks

`io/redo_log_flush`Acara ini untuk operasi tulisinput/output (I/O) di Aurora My. SQL

> **Note**
>
> Di Aurora My SQL versi 2, acara tunggu ini bernama io/aurora_redo_log_flush.

Kemungkinan penyebab peningkatan peristiwa tunggu

Untuk persistensi data, commit memerlukan penulisan durabel ke penyimpanan yang stabil. Jika basis data terlalu banyak melakukan commit, terdapat peristiwa tunggu pada operasi I/O tulis, yakni peristiwa tunggu `io/redo_log_flush`.

Untuk contoh perilaku peristiwa tunggu ini, lihat io/aurora_redo_log_flush.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri yang bermasalah
- Mengelompokkan operasi tulis
- Menonaktifkan autocommit
- Menggunakan transaksi
- Menggunakan batch

Mengidentifikasi sesi dan kueri yang bermasalah

Jika instans DB Anda mengalami kemacetan, langkah pertama adalah menemukan sesi dan kueri yang menjadi penyebabnya. Untuk posting Blog AWS Database yang berguna, lihat Menganalisis SQL Beban Kerja Saya Amazon Aurora dengan Performance Insights.

Untuk mengidentifikasi sesi dan kueri yang menyebabkan kemacetan:

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB Anda.
4. Di Beban basis data, pilih Potong berdasarkan masa tunggu.

5. Di bagian bawah halaman, pilih Top SQL.

   Kueri di bagian atas daftar adalah penyebab beban tertinggi pada basis data.

### Mengelompokkan operasi tulis

Contoh berikut memicu peristiwa tunggu `io/redo_log_flush`. (Autocommit diaktifkan.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

Dalam upaya mengurangi waktu yang dihabiskan untuk menunggu peristiwa tunggu `io/redo_log_flush`, kelompokkan operasi tulis secara logis ke dalam satu commit guna mengurangi panggilan persisten ke penyimpanan.

### Menonaktifkan autocommit

Nonaktifkan autocommit sebelum membuat perubahan besar yang tidak ada dalam transaksi, seperti yang ditunjukkan pada contoh berikut.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
```

```
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

Menggunakan transaksi

Anda dapat menggunakan transaksi, seperti yang ditunjukkan pada contoh berikut.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

Menggunakan batch

Anda dapat membuat perubahan dalam batch, seperti yang ditunjukkan pada contoh berikut. Namun, menggunakan batch yang terlalu besar dapat menyebabkan masalah kinerja, terutama pada replika baca atau saat melakukan point-in-time recovery ()PITR.

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

## io/socket/sql/client_koneksi

Peristiwa `io/socket/sql/client_connection` terjadi saat thread sedang dalam proses penanganan koneksi baru.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora SQL Versi saya 2 dan 3

### Konteks

Peristiwa `io/socket/sql/client_connection` menunjukkan bahwa mysqld sibuk membuat thread untuk menangani koneksi klien baru yang masuk. Dalam skenario ini, pemrosesan layanan permintaan koneksi klien baru melambat, sementara koneksi menunggu agar thread ditetapkan. Untuk informasi selengkapnya, lihat Server MySQL (mysqld).

### Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

- Ada peningkatan mendadak dalam koneksi pengguna baru dari aplikasi ke RDS instans Amazon Anda.
- Instans DB Anda tidak dapat memproses koneksi baru karena jaringanCPU,, atau memori sedang dibatasi.

### Tindakan

Jika `io/socket/sql/client_connection` mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Dalam basis data yang tidak idle, peristiwa tunggu selalu berada di atas. Lakukan tindakan hanya bila performa menurun. Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri yang bermasalah
- Mengikuti praktik terbaik untuk manajemen koneksi
- Menskalakan instans jika sumber daya sedang dibatasi
- Memeriksa host teratas dan pengguna teratas
- Membuat kueri tabel performance_schema
- Memeriksa status thread kueri
- Mengaudit permintaan dan kueri
- Menggabungkan koneksi basis data

Mengidentifikasi sesi dan kueri yang bermasalah

Jika instans DB Anda mengalami kemacetan, langkah pertama adalah menemukan sesi dan kueri yang menjadi penyebabnya. Untuk posting blog yang bermanfaat, lihat Menganalisis SQL Beban Kerja Saya Amazon Aurora dengan Performance Insights.

Untuk mengidentifikasi sesi dan kueri yang menyebabkan kemacetan:

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB Anda.
4. Di Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Kueri di bagian atas daftar adalah penyebab beban tertinggi pada basis data.

Mengikuti praktik terbaik untuk manajemen koneksi

Untuk mengelola koneksi Anda, pertimbangkan strategi berikut:

- Gunakan penggabungan koneksi.

  Anda dapat secara bertahap meningkatkan jumlah koneksi sesuai yang diperlukan. Untuk informasi selengkapnya, lihat whitepaper Amazon Aurora SQL My Database Administrator's Handbook.

- Gunakan simpul pembaca untuk mendistribusikan kembali lalu lintas hanya-baca.

  Untuk informasi selengkapnya, silakan lihat Replika Aurora dan Koneksi titik akhir Amazon Aurora.

Menskalakan instans jika sumber daya sedang dibatasi

Cari contoh throttling dalam sumber daya berikut:

- CPU

  Periksa CloudWatch metrik Amazon Anda untuk CPU penggunaan tinggi.

- Jaringan

  Periksa peningkatan nilai CloudWatch metrik `network receive throughput` dan`network transmit throughput`. Jika instans Anda telah mencapai batas bandwidth jaringan untuk kelas instans Anda, pertimbangkan untuk meningkatkan RDS instance Anda ke jenis kelas instans yang lebih tinggi. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB.

- Memori yang dapat dibebaskan

  Periksa penurunan CloudWatch metrik`FreeableMemory`. Pertimbangkan juga untuk mengaktifkan Peningkatan Pemantauan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan.

Memeriksa host teratas dan pengguna teratas

Gunakan Wawasan Performa untuk memeriksa host teratas dan pengguna teratas. Untuk informasi selengkapnya, lihat Menganalisis metrik dengan dasbor Wawasan Performa.

Membuat kueri tabel performance_schema

Untuk mendapatkan jumlah koneksi total dan saat ini yang akurat, buat kueri tabel `performance_schema`. Dengan teknik ini, Anda mengidentifikasi pengguna sumber atau host yang bertanggung jawab untuk membuat koneksi dalam jumlah besar. Misalnya, buat kueri tabel `performance_schema` sebagai berikut:

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

Memeriksa status thread kueri

Jika masalah performa terus berlanjut, periksa status thread kueri. Di klien `mysql`, keluarkan perintah berikut.

```
show processlist;
```

Mengaudit permintaan dan kueri

Untuk memeriksa sifat permintaan dan kueri dari akun pengguna, gunakan Audit SQL Lanjutan AuroraAurora Saya. Untuk mempelajari cara mengaktifkan audit, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.

Menggabungkan koneksi basis data

Pertimbangkan untuk menggunakan Amazon RDS Proxy untuk manajemen koneksi. Dengan menggunakan RDS Proxy, Anda dapat mengizinkan aplikasi Anda untuk mengumpulkan dan berbagi koneksi database untuk meningkatkan kemampuan mereka dalam skala. RDS Proxy membuat aplikasi lebih tahan terhadap kegagalan database dengan secara otomatis menghubungkan ke instans DB siaga sambil mempertahankan koneksi aplikasi. Untuk informasi selengkapnya, lihat Proksi Amazon RDS untuk Aurora.

## io/table/sql/handler

Peristiwa `io/table/sql/handler` terjadi ketika pekerjaan telah didelegasikan ke mesin penyimpanan.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora SQL Versi saya 2 dan 3

Konteks

Peristiwa `io/table` menunjukkan peristiwa tunggu untuk akses ke tabel. Peristiwa ini terjadi terlepas dari apakah data di-cache di pool buffer atau diakses pada disk. Peristiwa `io/table/sql/handler` menunjukkan peningkatan aktivitas beban kerja.

Handler adalah rutinitas yang berspesialisasi dalam jenis data tertentu atau berfokus pada tugas khusus tertentu. Misalnya, handler peristiwa menerima dan mencerna peristiwa serta sinyal dari sistem operasi atau antarmuka pengguna. Handler memori melakukan tugas-tugas yang berkaitan dengan memori. Handler input file adalah fungsi yang menerima input file dan melakukan tugas khusus pada data, sesuai dengan konteksnya.

Tampilan seperti `performance_schema.events_waits_current` sering kali menunjukkan `io/table/sql/handler` ketika peristiwa tunggu sebenarnya adalah peristiwa tunggu bersarang seperti penguncian. Jika peristiwa tunggu sebenarnya bukan `io/table/sql/handler`, Wawasan Performa akan melaporkan peristiwa tunggu bersarang. Saat melaporkan `io/table/sql/handler`, Wawasan Performa mewakili pemrosesan InnoDB atas permintaan I/O, bukan peristiwa tunggu bersarang yang tersembunyi. Untuk informasi lebih lanjut, lihat Peristiwa Atom dan Molekul Skema Kinerja di Manual SQL Referensi Saya.

`io/table/sql/handler`Acara ini sering muncul di acara tunggu teratas dengan menunggu I/O seperti. `io/aurora_redo_log_flush`

Kemungkinan penyebab peningkatan peristiwa tunggu

Dalam Wawasan Performa, lonjakan mendadak dalam peristiwa `io/table/sql/handler` menunjukkan adanya peningkatan aktivitas beban kerja. Peningkatan aktivitas berarti peningkatan I/O.

Performance Insights memfilter peristiwa bersarang IDs dan tidak melaporkan `io/table/sql/handler` penantian saat peristiwa bersarang yang mendasarinya adalah menunggu kunci. Misalnya, jika peristiwa akar penyebabnya adalah synch/mutex/innodb/aurora_lock_thread_slot_futex, Wawasan Performa akan menampilkan peristiwa tunggu ini dalam peristiwa tunggu teratas, bukan `io/table/sql/handler`.

Dalam tampilan seperti `performance_schema.events_waits_current`, peristiwa tunggu untuk `io/table/sql/handler` sering kali muncul ketika peristiwa tunggu sebenarnya adalah peristiwa tunggu bersarang seperti penguncian. Jika peristiwa tunggu sebenarnya berbeda dengan `io/table/sql/handler`, Wawasan Performa akan mencari peristiwa tunggu bersarang dan

melaporkan peristiwa tunggu sebenarnya, bukan `io/table/sql/handler`. Saat Wawasan Performa melaporkan `io/table/sql/handler`, peristiwa tunggu sebenarnya adalah `io/table/sql/handler`, bukan peristiwa tunggu bersarang yang tersembunyi. Untuk informasi lebih lanjut, lihat Peristiwa Atom dan Molekul Skema Kinerja di Manual Referensi SQL 5.7 Saya.

Tindakan

Jika peristiwa tunggu ini mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Peristiwa tunggu selalu berada di atas saat basis data aktif. Anda perlu melakukan tindakan hanya bila performa menurun.

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Memeriksa korelasi dengan metrik penghitung Wawasan Performa
- Memeriksa peristiwa tunggu berkorelasi lainnya

### Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan SQL kueri yang bertanggung jawab atas beban tinggi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Memeriksa korelasi dengan metrik penghitung Wawasan Performa

Periksa metrik penghitung Wawasan Performa seperti `Innodb_rows_changed`. Jika metrik penghitung berkorelasi dengan `io/table/sql/handler`, ikuti langkah-langkah berikut:

1. Dalam Performance Insights, cari SQL laporan yang memperhitungkan peristiwa tunggu `io/table/sql/handler` teratas. Jika memungkinkan, optimalkan pernyataan ini sehingga mengembalikan lebih sedikit baris.
2. Ambil tabel teratas dari tampilan `schema_table_statistics` dan `x $schema_table_statistics`. Tampilan ini menunjukkan jumlah waktu yang dihabiskan per tabel. Untuk informasi selengkapnya, lihat Tampilan schema_table_statistics dan x $schema_table_statistics di Manual Referensi Saya. SQL

   Secara default, baris diurutkan berdasarkan total waktu tunggu menurun. Tabel dengan pertentangan terbanyak ditampilkan terlebih dahulu. Output menunjukkan apakah waktu dihabiskan untuk membaca, menulis, mengambil, menyisipkan, memperbarui, atau menghapus.

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
    fetch_latency: 39.14 m
    rows_inserted: 14833
   insert_latency: 5.78 m
     rows_updated: 30470
   update_latency: 5.39 m
     rows_deleted: 14833
   delete_latency: 3.81 m
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
 io_misc_requests: NULL
```

```
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

Memeriksa peristiwa tunggu berkorelasi lainnya

Jika `synch/sxlock/innodb/btr_search_latch` dan `io/table/sql/handler` bersama-sama berkontribusi paling besar terhadap anomali beban DB, periksa apakah variabel `innodb_adaptive_hash_index` diaktifkan. Jika ya, coba tingkatkan nilai parameter `innodb_adaptive_hash_index_parts`.

Jika Adaptive Hash Index dinonaktifkan, coba untuk mengaktifkannya. Untuk mempelajari lebih lanjut tentang Indeks Hash SQL Adaptif Saya, lihat sumber daya berikut:

- Artikel Is Adaptive Hash Index in InnoDB right for my workload? di situs web Percona
- Indeks Hash Adaptif dalam Manual Referensi Saya SQL
- Artikel Pertikaian di SQL InnoDB Saya: Info Berguna Dari Bagian Semaphores di situs web Percona

> **Note**
>
> Adaptive Hash Index tidak didukung pada instans DB pembaca Aurora.
> Dalam kasus tertentu, performa mungkin buruk pada instans pembaca saat `synch/sxlock/innodb/btr_search_latch` dan `io/table/sql/handler` bersifat dominan. Jika demikian, coba alihkan beban kerja sementara ke instans DB penulis dan aktifkan Adaptive Hash Index.

## synch/cond/innodb/row_lock_tunggu

Peristiwa `synch/cond/innodb/row_lock_wait` terjadi ketika satu sesi telah mengunci baris untuk pembaruan dan sesi lain mencoba memperbarui baris yang sama. Untuk informasi selengkapnya, lihat InnoDB locking di dokumentasi SayaSQL.

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 3 SQL

Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris atau baris yang sama secara bersamaan.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Temukan dan tanggapi SQL pernyataan yang bertanggung jawab atas acara tunggu ini
- Menemukan dan merespons sesi pemblokiran

Temukan dan tanggapi SQL pernyataan yang bertanggung jawab atas acara tunggu ini

Gunakan Performance Insights untuk mengidentifikasi SQL pernyataan yang bertanggung jawab atas peristiwa tunggu ini. Pertimbangkan strategi berikut:

- Jika kunci baris merupakan masalah yang terus-menerus, coba tulis ulang aplikasi untuk menggunakan penguncian optimis.
- Gunakan pernyataan multibaris.
- Bagikan beban kerja ke objek basis data yang berbeda-beda. Anda dapat melakukannya melalui partisi.
- Periksa nilai parameter `innodb_lock_wait_timeout`. Parameter ini mengontrol durasi transaksi menunggu sebelum menghasilkan kesalahan batas waktu.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Menemukan dan merespons sesi pemblokiran

Cari tahu apakah sesi pemblokiran idle atau aktif. Cari tahu juga apakah sesi tersebut berasal dari aplikasi atau pengguna aktif.

Untuk mengidentifikasi sesi yang memegang kunci, Anda dapat menjalankan `SHOW ENGINE INNODB STATUS`. Contoh berikut menunjukkan output sampel.

```
mysql> SHOW ENGINE INNODB STATUS;
```

```
---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

Sebagai alternatif, Anda dapat menggunakan kueri berikut untuk mengekstrak detail tentang kunci saat ini.

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G
```

```
*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang mengidentifikasi transaksi pemblokiran, lihat Menggunakan transaksi InnoDB dan mengunci informasi dalam dokumentasi SayaSQL.

### synch/cond/innodb/row_lock_wait_cond

Peristiwa `synch/cond/innodb/row_lock_wait_cond` terjadi ketika satu sesi telah mengunci baris untuk pembaruan dan sesi lain mencoba memperbarui baris yang sama. Untuk informasi selengkapnya, lihat InnoDB mengunci di dokumentasi SayaSQL.

#### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 2 SQL

Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris atau baris yang sama secara bersamaan.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Temukan dan tanggapi SQL pernyataan yang bertanggung jawab atas acara tunggu ini
- Menemukan dan merespons sesi pemblokiran

Temukan dan tanggapi SQL pernyataan yang bertanggung jawab atas acara tunggu ini

Gunakan Performance Insights untuk mengidentifikasi SQL pernyataan yang bertanggung jawab atas peristiwa tunggu ini. Pertimbangkan strategi berikut:

- Jika kunci baris merupakan masalah yang terus-menerus, coba tulis ulang aplikasi untuk menggunakan penguncian optimis.
- Gunakan pernyataan multibaris.
- Bagikan beban kerja ke objek basis data yang berbeda-beda. Anda dapat melakukannya melalui partisi.
- Periksa nilai parameter `innodb_lock_wait_timeout`. Parameter ini mengontrol durasi transaksi menunggu sebelum menghasilkan kesalahan batas waktu.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Menemukan dan merespons sesi pemblokiran

Cari tahu apakah sesi pemblokiran idle atau aktif. Cari tahu juga apakah sesi tersebut berasal dari aplikasi atau pengguna aktif.

Untuk mengidentifikasi sesi yang memegang kunci, Anda dapat menjalankan `SHOW ENGINE INNODB STATUS`. Contoh berikut menunjukkan output sampel.

```
mysql> SHOW ENGINE INNODB STATUS;
```

```
---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

Sebagai alternatif, Anda dapat menggunakan kueri berikut untuk mengekstrak detail tentang kunci saat ini.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
```

```
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang mengidentifikasi transaksi pemblokiran, lihat Menggunakan transaksi InnoDB dan mengunci informasi dalam dokumentasi SayaSQL.

## synch/cond/sql/MDL_konteks:: COND _wait_status

Peristiwa `synch/cond/sql/MDL_context::COND_wait_status` terjadi saat terdapat thread yang menunggu pada kunci metadata tabel.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora SQL Versi saya 2 dan 3

### Konteks

Peristiwa `synch/cond/sql/MDL_context::COND_wait_status` menunjukkan bahwa terdapat thread yang menunggu pada kunci metadata tabel. Dalam kasus tertentu, satu sesi mempertahankan kunci metadata pada tabel, sementara sesi lainnya mencoba mendapatkan kunci yang sama pada tabel yang sama. Jika demikian, sesi kedua akan menunggu pada peristiwa tunggu `synch/cond/sql/MDL_context::COND_wait_status`.

Saya SQL menggunakan penguncian metadata untuk mengelola akses bersamaan ke objek database dan untuk memastikan konsistensi data. Penguncian metadata berlaku untuk tabel, skema, acara terjadwal, tablespace, dan penguncian pengguna yang diperoleh dengan fungsi `get_lock`, serta program yang disimpan. Program yang disimpan mencakup prosedur, fungsi, dan pemicu. Untuk informasi selengkapnya, lihat Penguncian metadata di dokumentasi SayaSQL.

Daftar SQL proses saya menunjukkan sesi ini di negara bagian`waiting for metadata lock`. Dalam Wawasan Performa, jika `Performance_schema` diaktifkan, maka peristiwa `synch/cond/sql/MDL_context::COND_wait_status` akan muncul.

Batas waktu default untuk kueri yang menunggu pada penguncian metadata didasarkan pada nilai parameter `lock_wait_timeout`, dengan nilai default 31.536.000 detik (365 hari).

Untuk detail selengkapnya tentang berbagai kunci InnoDB dan jenis kunci yang dapat menyebabkan konflik, lihat Penguncian InnoDB di dokumentasi Saya. SQL

### Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `synch/cond/sql/MDL_context::COND_wait_status` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

#### Transaksi yang berjalan lama

Satu atau lebih transaksi mengubah data dalam jumlah besar data dan menahan kunci pada tabel untuk waktu yang sangat lama.

Transaksi idle

Satu atau lebih transaksi tetap terbuka untuk waktu yang lama, tanpa di-commit atau di-roll back.

DDLpernyataan pada tabel besar

Satu atau lebih pernyataan definisi data (DDL) pernyataan, seperti `ALTER TABLE` perintah, dijalankan pada tabel yang sangat besar.

Kunci tabel eksplisit

Terdapat kunci eksplisit pada tabel yang tidak dirilis tepat waktu. Misalnya, sebuah aplikasi mungkin menjalankan pernyataan `LOCK TABLE` secara salah.

### Tindakan

Kami merekomendasikan tindakan yang berbeda tergantung pada penyebab acara tunggu Anda dan pada versi cluster Aurora My SQL DB.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Memeriksa peristiwa masa lalu
- Jalankan kueri di Aurora Versi saya 2 SQL
- Merespons sesi pemblokiran

### Mengidentifikasi sesi dan kueri penyebab peristiwa

Anda dapat menggunakan Wawasan Performa untuk menampilkan kueri yang diblokir oleh peristiwa tunggu `synch/cond/sql/MDL_context::COND_wait_status`. Namun, untuk mengidentifikasi sesi pemblokiran, buat kueri tabel metadata dari `performance_schema` dan `information_schema` di klaster DB.

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan SQL kueri yang bertanggung jawab atas beban tinggi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa untuk instans DB tersebut akan muncul.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting Blog Database Menganalisis AWS Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Memeriksa peristiwa masa lalu

Anda dapat memperoleh wawasan tentang peristiwa tunggu ini untuk memeriksa peristiwa di masa lalu. Untuk melakukannya, selesaikan tindakan berikut:

- Periksa bahasa manipulasi data (DML) dan DDL throughput dan latensi untuk melihat apakah ada perubahan dalam beban kerja.

  Anda dapat menggunakan Wawasan Performa untuk menemukan kueri yang menunggu pada peristiwa ini saat masalah terjadi. Anda juga dapat melihat inti sari kueri yang berjalan menjelang waktu masalah terjadi.
- Jika log audit atau log umum diaktifkan untuk klaster DB, Anda dapat memeriksa semua kueri yang berjalan pada objek (schema.table) yang terlibat dalam transaksi tunggu. Anda juga dapat memeriksa kueri yang selesai berjalan sebelum transaksi.

Informasi yang tersedia untuk memecahkan masalah peristiwa masa lalu terbatas. Melakukan pemeriksaan ini tidak menunjukkan objek yang menunggu informasi. Namun, Anda dapat mengidentifikasi tabel dengan beban berat saat peristiwa terjadi dan kumpulan baris yang sering dioperasikan yang menyebabkan konflik pada saat masalah terjadi. Anda kemudian dapat

menggunakan informasi ini untuk mereproduksi masalah di lingkungan pengujian dan memberikan wawasan tentang penyebabnya.

Jalankan kueri di Aurora Versi saya 2 SQL

Di Aurora My SQL versi 2, Anda dapat mengidentifikasi sesi yang diblokir secara langsung dengan menanyakan `performance_schema` tabel atau `sys` tampilan skema. Sebuah contoh dapat mengilustrasikan cara membuat kueri tabel untuk mengidentifikasi kueri dan sesi pemblokiran.

Dalam output daftar proses berikut, ID koneksi 89 sedang menunggu pada kunci metadata, dan menjalankan perintah `TRUNCATE TABLE`. Dalam kueri pada tabel `performance_schema` atau tampilan skema `sys`, output menunjukkan bahwa sesi pemblokiran adalah 76.

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.11.5           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
| Id | User            | Host               | db        | Command | Time | State
                    | Info                           |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
|  2 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
|  4 | rdsadmin        | localhost          | NULL      | Sleep   |    2 | NULL
                    | NULL                           |
|  5 | rdsadmin        | localhost          | NULL      | Sleep   |    1 | NULL
                    | NULL                           |
| 20 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
| 21 | rdsadmin        | localhost          | NULL      | Sleep   |  261 | NULL
                    | NULL                           |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
```

```
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query   |    0 | starting
                    | show processlist              |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query   |   22 | User sleep
                    | select sleep(10000)           |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query   |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+-------------------------------+
18 rows in set (0.00 sec)
```

Selanjutnya, sebuah kueri pada tabel `performance_schema` atau tampilan skema `sys` menunjukkan bahwa sesi pemblokiran adalah 76.

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+----------------------
+-------------------------------+--------------------+-----------------------------
+-----------------------------+--------------------+--------------
+-----------------------------+--------------------+-----------------------
+-------------------------+-----------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
          | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account             | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
```

```
+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+-----------------------
+------------------------------+--------------------+----------------------------
+------------------------------+--------------------+--------------
+------------------------------+--------------------+-----------------------
+--------------------------+-------------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                            0 |
               0 |                108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
              |
+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+-----------------------
+------------------------------+--------------------+----------------------------
+------------------------------+--------------------+--------------
+------------------------------+--------------------+-----------------------
+--------------------------+-------------------------------+
1 row in set (0.00 sec)
```

Merespons sesi pemblokiran

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang mengidentifikasi transaksi pemblokiran, lihat Menggunakan Transaksi InnoDB dan Mengunci Informasi dalam dokumentasi SayaSQL.

## synch/mutex/innodb/aurora_lock_thread_slot_futex

Peristiwa `synch/mutex/innodb/aurora_lock_thread_slot_futex` terjadi ketika satu sesi telah mengunci baris untuk pembaruan dan sesi lain mencoba memperbarui baris yang sama. Untuk informasi selengkapnya, lihat penguncian InnoDB di Referensi Saya SQL.

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 2 SQL

Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris atau baris yang sama secara bersamaan.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Temukan dan tanggapi SQL pernyataan yang bertanggung jawab atas acara tunggu ini
- Menemukan dan merespons sesi pemblokiran

Temukan dan tanggapi SQL pernyataan yang bertanggung jawab atas acara tunggu ini

Gunakan Performance Insights untuk mengidentifikasi SQL pernyataan yang bertanggung jawab atas peristiwa tunggu ini. Pertimbangkan strategi berikut:

- Jika kunci baris merupakan masalah yang terus-menerus, coba tulis ulang aplikasi untuk menggunakan penguncian optimis.
- Gunakan pernyataan multibaris.
- Bagikan beban kerja ke objek basis data yang berbeda-beda. Anda dapat melakukannya melalui partisi.
- Periksa nilai parameter `innodb_lock_wait_timeout`. Parameter ini mengontrol durasi transaksi menunggu sebelum menghasilkan kesalahan batas waktu.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Menemukan dan merespons sesi pemblokiran

Cari tahu apakah sesi pemblokiran idle atau aktif. Cari tahu juga apakah sesi tersebut berasal dari aplikasi atau pengguna aktif.

Untuk mengidentifikasi sesi yang memegang kunci, Anda dapat menjalankan `SHOW ENGINE INNODB STATUS`. Contoh berikut menunjukkan output sampel.

```
mysql> SHOW ENGINE INNODB STATUS;

----------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

Sebagai alternatif, Anda dapat menggunakan kueri berikut untuk mengekstrak detail tentang kunci saat ini.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
```

```
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang mengidentifikasi transaksi pemblokiran, lihat Menggunakan Transaksi InnoDB dan Mengunci Informasi di Manual SQLReferensi Saya.

## synch/mutex/innodb/buf_pool_mutex

Peristiwa `synch/mutex/innodb/buf_pool_mutex` terjadi saat thread telah mendapat kunci pada kumpulan buffer InnoDB untuk mengakses halaman dalam memori.

Topik

- Versi mesin yang relevan
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang relevan

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora Versi saya 2 SQL

Konteks

Mutex `buf_pool` adalah mutex tunggal yang melindungi struktur data kontrol dari pool buffer.

Untuk informasi selengkapnya, lihat Memantau InnoDB Mutex Waits Using Performance Schema di dokumentasi Saya. SQL

Kemungkinan penyebab peningkatan peristiwa tunggu

Ini adalah peristiwa tunggu khusus beban kerja. Penyebab umum `synch/mutex/innodb/buf_pool_mutex` muncul di antara peristiwa tunggu teratas mencakup yang berikut:

- Ukuran pool buffer tidak cukup besar untuk menampung set data kerja.
- Beban kerja lebih spesifik untuk halaman tertentu dari tabel tertentu dalam basis data, yang mengarah ke pertentangan dalam pool buffer.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Menggunakan Wawasan Performa
- Membuat Aurora Replicas
- Memeriksa ukuran pool buffer
- Memantau riwayat status global

Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana

basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk melihat SQL bagan Teratas di Konsol AWS Manajemen

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bawah bagan beban Database, pilih Top. SQL

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Menggunakan Wawasan Performa

Peristiwa ini terkait dengan beban kerja. Anda dapat menggunakan Wawasan Performa untuk melakukan hal berikut:

- Mengidentifikasi kapan peristiwa tunggu dimulai dan apakah ada perubahan beban kerja pada saat itu dari log aplikasi atau sumber terkait.
- Identifikasi SQL pernyataan yang bertanggung jawab atas acara tunggu ini. Periksa rencana eksekusi kueri untuk memastikan bahwa kueri ini dioptimalkan dan menggunakan indeks yang sesuai.

  Jika kueri teratas yang bertanggung jawab atas peristiwa tunggu berkaitan dengan objek atau tabel basis data yang sama, coba buat partisi untuk objek atau tabel tersebut.

Membuat Aurora Replicas

Anda dapat membuat Aurora Replicas untuk menyajikan lalu lintas hanya-baca. Anda juga dapat menggunakan Aurora Auto Scaling untuk menangani lonjakan lalu lintas baca. Pastikan untuk menjalankan tugas hanya-baca terjadwal dan pencadangan logis pada Aurora Replicas.

Untuk informasi selengkapnya, lihat Auto Scaling Amazon Aurora dengan Replika Aurora.

Memeriksa ukuran pool buffer

Periksa apakah ukuran pool buffer cukup untuk beban kerja dengan melihat metrik `innodb_buffer_pool_wait_free`. Jika nilai metrik ini tinggi dan terus bertambah, berarti ukuran pool buffer tidak cukup untuk menangani beban kerja. Jika `innodb_buffer_pool_size` telah diatur dengan benar, nilai `innodb_buffer_pool_wait_free` pasti kecil. Untuk informasi selengkapnya, lihat InnodB_Buffer_POOL_WAIT_FREE di dokumentasi Saya. SQL

Tingkatkan ukuran pool buffer jika instans DB memiliki cukup memori untuk buffer sesi dan tugas sistem operasi. Jika tidak, ubah instans DB ke kelas instans DB yang lebih besar untuk memperoleh memori tambahan yang dapat dialokasikan ke pool buffer.

> **Note**
>
> Aurora My SQL secara otomatis menyesuaikan nilai `innodb_buffer_pool_instances` berdasarkan konfigurasi. `innodb_buffer_pool_size`

Memantau riwayat status global

Dengan memantau tingkat perubahan variabel status, Anda dapat mendeteksi masalah penguncian atau memori pada instans DB Anda. Aktifkan Global Status History (GoSH) jika belum diaktifkan. Untuk informasi selengkapnya tentang GoSH, lihat Mengelola riwayat status global.

Anda juga dapat membuat CloudWatch metrik Amazon khusus untuk memantau variabel status. Untuk informasi selengkapnya, lihat Memublikasikan metrik kustom.

## synch/mutex/innodb/fil_system_mutex

Peristiwa `synch/mutex/innodb/fil_system_mutex` terjadi ketika sesi menunggu untuk mengakses cache memori tablespace.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu

- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora SQL Versi saya 2 dan 3

Konteks

InnoDB menggunakan tablespace untuk mengelola area penyimpanan untuk tabel dan file log. Cache memori tablespace adalah struktur memori global yang memelihara informasi tentang tablespace. SQLPenggunaan saya `synch/mutex/innodb/fil_system_mutex` menunggu untuk mengontrol akses bersamaan ke cache memori tablespace.

Peristiwa `synch/mutex/innodb/fil_system_mutex` menunjukkan bahwa saat ini ada lebih dari satu operasi yang perlu mengambil dan memanipulasi informasi dalam cache memori tablespace untuk tablespace yang sama.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `synch/mutex/innodb/fil_system_mutex` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, hal ini biasanya terjadi ketika terdapat semua kondisi berikut:

- Peningkatan operasi bahasa manipulasi data (DML) bersamaan yang memperbarui atau menghapus data dalam tabel yang sama.
- Tablespace untuk tabel ini sangat besar dan memiliki banyak halaman data.
- Faktor pengisian untuk halaman data ini rendah.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Mengatur ulang tabel besar di luar jam sibuk

Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan SQL kueri yang bertanggung jawab atas beban tinggi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa untuk instans DB tersebut akan muncul.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Cara lain untuk mengetahui kueri yang menyebabkan tingginya jumlah peristiwa tunggu `synch/mutex/innodb/fil_system_mutex` adalah dengan memeriksa `performance_schema`, seperti pada contoh berikut.

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
```

```
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
```

```
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

Mengatur ulang tabel besar di luar jam sibuk

Atur ulang tabel besar yang Anda identifikasi sebagai sumber tingginya jumlah peristiwa tunggu `synch/mutex/innodb/fil_system_mutex` selama periode pemeliharaan di luar jam produksi. Dengan melakukannya, Anda akan memastikan bahwa pembersihan peta tablespace internal tidak terjadi saat akses cepat ke tabel sangat penting. Untuk informasi tentang mengatur ulang tabel, lihat OPTIMIZETABLEPernyataan di SQLReferensi Saya.

## synch/mutex/innodb/trx_sys_mutex

Peristiwa `synch/mutex/innodb/trx_sys_mutex` terjadi ketika terdapat aktivitas basis data tinggi dengan transaksi dalam jumlah besar.

Topik

- Versi mesin yang relevan
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang relevan

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora SQL Versi saya 2 dan 3

### Konteks

Secara internal, mesin basis data InnoDB menggunakan tingkat isolasi baca berulang dengan snapshot untuk menghadirkan konsistensi baca. Ini memberi Anda point-in-time tampilan database pada saat snapshot dibuat.

Di InnoDB, semua perubahan diterapkan ke basis data segera setelah tersedia, terlepas dari apakah perubahan tersebut di-commit. Pendekatan ini berarti bahwa tanpa kontrol konkurensi multiversi (MVCC), semua pengguna yang terhubung ke database melihat semua perubahan dan baris terbaru. Karena itu, InnoDB memerlukan cara untuk melacak perubahan guna memahami strategi rollback bila diperlukan.

Untuk melakukannya, InnoDB menggunakan sistem transaksi (`trx_sys`) untuk melacak snapshot. Sistem transaksi melakukan hal berikut:

- Melacak ID transaksi untuk setiap baris dalam log pembatalan.
- Menggunakan struktur InnoDB internal `ReadView` yang disebut yang membantu mengidentifikasi transaksi mana IDs yang terlihat untuk snapshot.

Kemungkinan penyebab peningkatan peristiwa tunggu

Setiap operasi database yang memerlukan penanganan yang konsisten dan terkontrol (membuat, membaca, memperbarui, dan menghapus) transaksi IDs menghasilkan panggilan dari `trx_sys` mutex.

Panggilan ini terjadi di dalam tiga fungsi:

- `trx_sys_mutex_enter` – Menciptakan mutex.
- `trx_sys_mutex_exit` – Melepaskan mutex.
- `trx_sys_mutex_own` – Menguji apakah mutex dimiliki.

Instrumentasi Skema Performa InnoDB melacak semua panggilan mutex `trx_sys`. Pelacakan mencakup, tetapi tidak terbatas pada, manajemen `trx_sys` pada pengaktifan atau penonaktifan basis data, operasi rollback, pembersihan pembatalan, akses baca baris, dan pemuatan pool buffer. Aktivitas basis data yang tinggi dengan transaksi dalam jumlah besar mengakibatkan kemunculan `synch/mutex/innodb/trx_sys_mutex` di antara peristiwa tunggu teratas.

Untuk informasi selengkapnya, lihat Memantau InnoDB Mutex Waits Using Performance Schema di dokumentasi Saya. SQL

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Memeriksa peristiwa tunggu lainnya

### Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi terhadap beban tertinggi. Cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk melihat SQL grafik Teratas di AWS Management Console

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bawah Bagan beban database, pilih Top SQL.

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting blog Menganalisis Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

### Memeriksa peristiwa tunggu lainnya

Periksa peristiwa tunggu lain yang terkait dengan peristiwa tunggu `synch/mutex/innodb/trx_sys_mutex`. Dengan melakukannya, Anda dapat memperoleh informasi selengkapnya tentang sifat beban kerja. Sejumlah besar transaksi dapat mengurangi throughput, tetapi beban kerja mungkin juga mengharuskannya.

Untuk informasi selengkapnya tentang cara mengoptimalkan transaksi, lihat Mengoptimalkan Manajemen Transaksi InnoDB di dokumentasi Saya. SQL

## synch/sxlock/innodb/hash_table_locks

Peristiwa `synch/sxlock/innodb/hash_table_locks` terjadi ketika halaman yang tidak ditemukan di pool buffer harus dibaca dari penyimpanan.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi berikut:

- Aurora SQL Versi saya 2 dan 3

### Konteks

Peristiwa `synch/sxlock/innodb/hash_table_locks` menunjukkan bahwa beban kerja sering kali mengakses data yang tidak disimpan dalam pool buffer. Peristiwa tunggu ini dikaitkan dengan penambahan halaman baru dan pengosongan data lama dari pool buffer. Data yang disimpan dalam pool buffer usang dan data baru harus di-cache, jadi halaman usang dikosongkan untuk memungkinkan caching halaman baru. Saya SQL menggunakan algoritma (LRU) yang paling jarang digunakan untuk mengusir halaman dari kumpulan buffer. Beban kerja mencoba mengakses data yang belum dimuat ke dalam pool buffer atau data yang telah dikosongkan dari pool buffer.

Peristiwa tunggu ini terjadi ketika beban kerja harus mengakses data dalam file pada disk atau ketika blok dibebaskan dari atau ditambahkan ke daftar kumpulan buffer. LRU Operasi ini menunggu untuk memperoleh kunci yang dikecualikan bersama (SX-lock). SX-lock ini digunakan untuk sinkronisasi melalui tabel hash, yakni tabel dalam memori yang dirancang untuk meningkatkan performa akses pool buffer.

Untuk informasi selengkapnya, lihat Buffer Pool di SQL dokumentasi Saya.

### Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa tunggu `synch/sxlock/innodb/hash_table_locks` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Pool buffer berukuran sangat kecil

Ukuran pool buffer terlalu kecil untuk menyimpan semua halaman yang sering diakses dalam memori.

Beban kerja berat

Beban kerja menyebabkan seringnya pengosongan dan halaman data dimuat ulang di cache buffer.

Kesalahan membaca halaman

Terjadi kesalahan saat membaca halaman di pool buffer, yang mungkin mengindikasikan adanya kerusakan data.

### Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Meningkatkan ukuran pool buffer
- Meningkatkan pola akses data
- Mengurangi atau mencegah pemindaian tabel penuh
- Memeriksa log kesalahan untuk kerusakan halaman

### Meningkatkan ukuran pool buffer

Pastikan pool buffer memiliki ukuran yang sesuai untuk beban kerja. Untuk melakukannya, Anda dapat memeriksa laju hit cache pool buffer. Biasanya, jika nilainya turun di bawah 95%, coba tingkatkan ukuran pool buffer. Pool buffer yang lebih besar dapat lebih lama menyimpan halaman yang sering diakses dalam memori. Untuk meningkatkan ukuran pool buffer, ubah nilai parameter `innodb_buffer_pool_size`. Nilai default parameter ini didasarkan pada ukuran kelas instans DB. Untuk informasi selengkapnya, lihat Praktik terbaik untuk konfigurasi SQL database Amazon Aurora Saya.

### Meningkatkan pola akses data

Periksa kueri yang terpengaruh oleh peristiwa tunggu ini dan rencana eksekusinya. Coba tingkatkan pola akses data. Misalnya, jika menggunakan mysqli_result::fetch_array, Anda dapat mencoba meningkatkan ukuran pengambilan array.

Anda dapat menggunakan Wawasan Performa untuk menampilkan kueri dan sesi yang mungkin menyebabkan peristiwa tunggu `synch/sxlock/innodb/hash_table_locks`.

Untuk menemukan SQL kueri yang bertanggung jawab atas beban tinggi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih Top SQL.

   Bagan mencantumkan SQL kueri yang bertanggung jawab atas pemuatan. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar berguna tentang pemecahan masalah menggunakan Performance Insights, lihat posting Blog Database Menganalisis AWS Beban Kerja SQL Saya Amazon Aurora dengan Performance Insights.

Mengurangi atau mencegah pemindaian tabel penuh

Pantau beban kerja untuk melihat apakah pemindaian tabel penuh dijalankan, dan jika ya, kurangi atau cegah pemindaian tersebut. Misalnya, Anda dapat memantau variabel status seperti `Handler_read_rnd_next`. Untuk informasi selengkapnya, lihat Variabel Status Server dalam SQL dokumentasi Saya.

Memeriksa log kesalahan untuk kerusakan halaman

Anda dapat memeriksa mysql-error.log untuk pesan terkait kerusakan yang terdeteksi menjelang waktu terjadinya masalah. Pesan yang dapat Anda tangani untuk menyelesaikan masalah terdapat di log kesalahan. Anda mungkin perlu membuat ulang objek yang dilaporkan rusak.

# Menyesuaikan Aurora MySQL dengan status thread

Tabel berikut merangkum status thread paling umum untuk Aurora MySQL.

| Status thread umum | Deskripsi |
| --- | --- |
| ??? | Status thread ini menunjukkan bahwa sebuah thread sedang memproses pernyataan `SELECT` yang memerlukan penggunaan tabel sementara internal untuk mengurutkan data. |
| ??? | Status thread ini menunjukkan bahwa sebuah thread sedang membaca dan memfilter baris untuk kueri guna menentukan set hasil yang benar. |

## creating sort index

Status thread `creating sort index` menunjukkan bahwa thread sedang memproses pernyataan `SELECT` yang memerlukan penggunaan tabel sementara internal untuk mengurutkan data.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi status thread ini didukung untuk versi berikut:

- Aurora SQL Versi saya 2 hingga 2.09.2

### Konteks

Status `creating sort index` muncul saat kueri dengan klausa `ORDER BY` atau `GROUP BY` tidak dapat menggunakan indeks yang ada untuk melakukan operasi. Dalam hal ini, SQL kebutuhan saya

untuk melakukan `filesort` operasi yang lebih mahal. Operasi ini biasanya dilakukan dalam memori jika set hasil tidak terlalu besar. Jika set hasil besar, operasi akan melibatkan pembuatan file pada disk.

Kemungkinan penyebab peningkatan peristiwa tunggu

Kemunculan `creating sort index` tidak dengan sendirinya menunjukkan adanya suatu masalah. Jika performa buruk, dan Anda sering melihat instans `creating sort index`, penyebab yang paling mungkin adalah kueri lambat dengan operator `ORDER BY` atau `GROUP BY`.

Tindakan

Pedoman umumnya adalah menemukan kueri dengan klausa `ORDER BY` atau `GROUP BY` yang terkait dengan peningkatan dalam status `creating sort index`. Lalu, lihat apakah menambahkan indeks atau meningkatkan ukuran buffer urutan akan memecahkan masalah.

Topik

- Mengaktifkan Skema Performa jika tidak aktif
- Mengidentifikasi kueri masalah
- Memeriksa "jelaskan rencana" untuk penggunaan filesort
- Meningkatkan ukuran buffer urutan

Mengaktifkan Skema Performa jika tidak aktif

Wawasan Performa akan melaporkan status thread hanya jika instrumen Skema Performa tidak aktif. Bila instrumen Skema Performa diaktifkan, Wawasan Performa akan melaporkan peristiwa tunggu. Instrumen Skema Performa memberikan wawasan tambahan dan alat yang lebih baik untuk menyelidiki potensi masalah performa. Oleh karena itu, sebaiknya Anda mengaktifkan Skema Performa. Untuk informasi selengkapnya, lihat Ikhtisar Skema Kinerja untuk Performance Insights di Aurora.

Mengidentifikasi kueri masalah

Untuk mengidentifikasi kueri saat ini yang menyebabkan peningkatan dalam status `creating sort index`, jalankan `show processlist` dan lihat apakah kueri memiliki `ORDER BY` atau `GROUP BY`. Cara lainnya, jalankan `explain for connection N`, dengan `N` adalah ID daftar proses kueri dengan `filesort`.

Untuk mengidentifikasi kueri sebelumnya yang menyebabkan peningkatan ini, aktifkan log kueri lambat dan temukan kueri dengan `ORDER BY`. Jalankan `EXPLAIN` pada kueri lambat dan cari "menggunakan filesort". Untuk informasi selengkapnya, lihat Memeriksa "jelaskan rencana" untuk penggunaan filesort.

Memeriksa "jelaskan rencana" untuk penggunaan filesort

Identifikasi pernyataan dengan klausa `ORDER BY` atau `GROUP BY` yang menghasilkan status `creating sort index`.

Contoh berikut menunjukkan cara menjalankan `explain` pada kueri. Kolom `Extra` menunjukkan bahwa kueri ini menggunakan `filesort`.

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

Contoh berikut menunjukkan hasil menjalankan `EXPLAIN` pada kueri yang sama setelah indeks dibuat pada kolom `c1`.

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
```

```
          key: c1
      key_len: 1023
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

Untuk informasi tentang penggunaan indeks untuk pengoptimalan urutan urutan, lihat ORDERBY Optimization di SQL dokumentasi Saya.

Meningkatkan ukuran buffer urutan

Untuk mengetahui apakah kueri tertentu memerlukan proses `filesort` yang membuat file pada disk, periksa nilai variabel `sort_merge_passes` setelah menjalankan kueri. Berikut adalah contohnya.

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

Jika nilai `sort_merge_passes` tinggi, coba tingkatkan ukuran buffer urutan. Terapkan peningkatan pada tingkat sesi, karena meningkatkannya secara global dapat secara signifikan meningkatkan jumlah SQL penggunaan RAM Saya. Contoh berikut menunjukkan cara mengubah ukuran buffer urutan sebelum menjalankan kueri.

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
-- run query
```

## sending data

Status thread `sending data` menunjukkan bahwa thread membaca dan memfilter baris untuk kueri guna menentukan set hasil yang benar. Namanya agak membingungkan karena menyiratkan status mentransfer data, bukan mengumpulkan dan menyiapkan data untuk dikirim nanti.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi status thread ini didukung untuk versi berikut:

- Aurora MySQL versi 2 hingga 2.09.2

### Konteks

Banyak status thread tidak bertahan lama. Operasi yang terjadi selama `sending data` cenderung melakukan pembacaan disk atau cache dalam jumlah besar. Oleh karena itu, `sending data` sering kali merupakan status yang paling lama berjalan selama masa kueri tertentu. Status ini muncul saat Aurora MySQL melakukan hal berikut:

- Membaca dan memproses baris untuk pernyataan `SELECT`
- Menjalankan pembacaan dalam jumlah besar baik dari disk atau memori
- Menyelesaikan pembacaan lengkap semua data dari kueri tertentu
- Membaca data dari tabel, indeks, atau pekerjaan prosedur yang disimpan
- Menyortir, mengelompokkan, atau menyusun data

Setelah status `sending data` selesai menyiapkan data, status thread `writing to net` akan menunjukkan pengembalian data ke klien. Biasanya, `writing to net` diambil hanya ketika set hasil sangat besar atau latensi jaringan yang parah memperlambat transfer.

Kemungkinan penyebab peningkatan peristiwa tunggu

Kemunculan `sending data` tidak dengan sendirinya menunjukkan adanya suatu masalah. Jika performa buruk, dan Anda sering melihat instans `sending data`, berikut adalah penyebab yang paling memungkinkan.

Topik

- Kueri yang tidak efisien
- Konfigurasi server suboptimal

Kueri yang tidak efisien

Dalam sebagian besar kasus, status ini terutama disebabkan oleh kueri yang tidak menggunakan indeks yang sesuai untuk menemukan set hasil kueri tertentu. Sebagai contoh, pertimbangkan kueri yang membaca tabel berisi 10 juta data untuk semua pesanan yang ditempatkan di California, dengan kolom status tidak diindeks atau diindeks dengan buruk. Pada contoh terakhir, indeks mungkin ada, tetapi pengoptimal mengabaikannya karena kardinalitas rendah.

Konfigurasi server suboptimal

Jika beberapa kueri muncul dalam status `sending data`, server basis data mungkin dikonfigurasi dengan buruk. Secara khusus, server mungkin memiliki masalah berikut:

- Server basis data tidak memiliki kapasitas komputasi yang cukup: I/O disk, jenis dan kecepatan disk, CPU, atau jumlah CPU.
- Server kekurangan sumber daya yang dialokasikan, seperti pool buffer InnoDB untuk tabel InnoDB atau buffer kunci untuk tabel MyISAM.
- Pengaturan memori per-thread seperti `sort_buffer`, `read_buffer`, dan `join_buffer` menghabiskan lebih banyak RAM dari yang dibutuhkan, sehingga membuat server fisik kekurangan sumber daya memori.

Tindakan

Pedoman umumnya adalah menemukan kueri yang mengembalikan baris dalam jumlah besar dengan memeriksa Skema Performa. Jika pencatatan kueri yang tidak menggunakan indeks diaktifkan, Anda juga dapat memeriksa hasil dari log lambat.

Topik

- Mengaktifkan Skema Performa jika tidak aktif
- Memeriksa pengaturan memori
- Memeriksa "jelaskan rencana" untuk penggunaan indeks
- Memeriksa volume data yang dikembalikan
- Memeriksa masalah konkurensi
- Memeriksa struktur kueri

Mengaktifkan Skema Performa jika tidak aktif

Wawasan Performa akan melaporkan status thread hanya jika instrumen Skema Performa tidak aktif. Bila instrumen Skema Performa diaktifkan, Wawasan Performa akan melaporkan peristiwa tunggu. Instrumen Skema Performa memberikan wawasan tambahan dan alat yang lebih baik untuk menyelidiki potensi masalah performa. Oleh karena itu, sebaiknya Anda mengaktifkan Skema Performa. Untuk informasi selengkapnya, lihat Ikhtisar Skema Kinerja untuk Performance Insights di Aurora.

Memeriksa pengaturan memori

Periksa pengaturan memori untuk pool buffer utama. Pastikan pool ini memiliki ukuran yang sesuai untuk beban kerja. Jika basis data Anda menggunakan beberapa instans pool buffer, pastikan instans tersebut tidak dibagi menjadi banyak pool buffer kecil. Thread hanya dapat menggunakan satu pool buffer pada satu waktu.

Pastikan pengaturan memori berikut yang digunakan untuk setiap thread memiliki ukuran yang benar:

- read_buffer
- read_rnd_buffer
- sort_buffer
- join_buffer
- binlog_cache

Gunakan nilai default, kecuali jika Anda memiliki alasan khusus untuk mengubah pengaturan.

Memeriksa "jelaskan rencana" untuk penggunaan indeks

Untuk kueri dalam status thread `sending data`, periksa rencana untuk mengetahui apakah indeks yang sesuai telah digunakan. Jika kueri tidak menggunakan indeks yang berguna, coba tambahkan petunjuk seperti `USE INDEX` atau `FORCE INDEX`. Petunjuk dapat menambah atau mengurangi banyak waktu yang diperlukan untuk menjalankan kueri, jadi berhati-hatilah sebelum menambahkannya.

Memeriksa volume data yang dikembalikan

Periksa tabel yang ditanyakan dan jumlah data yang ada di dalamnya. Apakah data ini dapat diarsipkan? Dalam banyak kasus, penyebab waktu eksekusi kueri yang buruk bukanlah hasil dari rencana kueri, tetapi volume data yang akan diproses. Banyak developer sangat efisien dalam menambahkan data ke basis data, tetapi jarang mempertimbangkan siklus proses set data dalam fase desain dan pengembangan.

Cari kueri yang berperforma baik dalam basis data volume rendah, tetapi berperforma buruk di sistem Anda saat ini. Terkadang, developer yang mendesain kueri tertentu mungkin tidak menyadari bahwa kueri tersebut mengembalikan 350.000 baris. Developer mungkin telah mengembangkan kueri di lingkungan volume yang lebih rendah, dengan set data lebih kecil dari yang dimiliki lingkungan produksi.

Memeriksa masalah konkurensi

Periksa apakah beberapa kueri dari jenis yang sama berjalan pada waktu yang sama. Format kueri tertentu berjalan secara efisien bila dijalankan sendiri. Namun, jika format kueri serupa berjalan bersama, atau dalam volume tinggi, hal tersebut dapat menimbulkan masalah konkurensi. Masalah ini sering kali muncul bila basis data menggunakan tabel sementara untuk merender hasil. Tingkat isolasi transaksi yang ketat juga dapat menyebabkan masalah konkurensi.

Jika tabel dibaca dan ditulis secara bersamaan, basis data mungkin menggunakan kunci. Untuk membantu mengidentifikasi periode performa yang buruk, periksa penggunaan basis data melalui proses batch skala besar. Untuk melihat kunci dan rollback terbaru, periksa output dari perintah `SHOW ENGINE INNODB STATUS`.

Memeriksa struktur kueri

Periksa apakah kueri yang diambil dari status ini menggunakan subkueri. Jenis kueri ini sering kali menimbulkan performa yang buruk karena basis data mengompilasi hasil secara internal, lalu

menggantinya kembali ke kueri untuk merender data. Proses ini merupakan langkah ekstra untuk basis data. Dalam banyak kasus, langkah ini dapat menimbulkan performa yang buruk pada kondisi pemuatan yang sangat bersamaan.

Periksa juga apakah kueri Anda menggunakan klausa `ORDER BY` dan `GROUP BY` dalam jumlah besar. Dalam operasi semacam ini, sering kali basis data harus terlebih dahulu membentuk set data secara keseluruhan dalam memori. Selanjutnya, basis data harus menyusun atau mengelompokkannya secara khusus sebelum mengembalikannya ke klien.

## Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru

Wawasan proaktif DevOps Guru mendeteksi kondisi bermasalah yang diketahui pada klaster DB Aurora MySQL sebelum masalah tersebut terjadi. DevOps Guru dapat melakukan hal berikut:

- Mencegah banyak masalah umum pada basis data dengan memeriksa silang konfigurasi basis data terhadap pengaturan umum yang direkomendasikan.
- Memberi tahu Anda tentang masalah kritis dalam armada yang, jika dibiarkan tanpa diperiksa, dapat menyebabkan masalah yang lebih besar di kemudian hari.
- Memberi tahu Anda tentang masalah yang baru ditemukan.

Setiap wawasan proaktif berisi analisis penyebab masalah dan rekomendasi untuk tindakan korektif.

Topik

- Panjang daftar riwayat InnoDB meningkat secara signifikan
- Basis data membuat tabel sementara pada disk

### Panjang daftar riwayat InnoDB meningkat secara signifikan

Mulai dari*`date`*, daftar riwayat Anda untuk perubahan baris meningkat secara signifikan, hingga *`length`* aktif*`db-instance`*. Peningkatan ini memengaruhi performa penonaktifan kueri dan basis data.

Topik

- Versi mesin yang didukung
- Konteks

- Kemungkinan penyebab masalah ini
- Tindakan
- Metrik terkait

Versi mesin yang didukung

Informasi wawasan ini didukung untuk semua versi Aurora MySQL.

Konteks

Sistem transaksi InnoDB mempertahankan kontrol konkurensi multiversi (MVCC). Ketika baris diubah, versi pra-modifikasi dari data yang diubah disimpan sebagai catatan undo dalam log undo. Setiap data undo memiliki referensi ke data redo sebelumnya, yang membentuk daftar tertaut.

Daftar riwayat InnoDB adalah daftar global log undo untuk transaksi yang dilakukan. MySQL menggunakan daftar riwayat untuk membersihkan data dan halaman log saat transaksi tidak lagi memerlukan riwayat. Panjang daftar riwayat adalah jumlah total log undo yang berisi perubahan dalam daftar riwayat. Setiap log berisi satu atau beberapa modifikasi. Jika panjang daftar riwayat InnoDB bertambah terlalu besar, yang menunjukkan sejumlah besar versi baris lama, kueri dan penonaktifan basis data menjadi lebih lambat.

Kemungkinan penyebab masalah ini

Penyebab umum dari daftar riwayat panjang meliputi:

- Transaksi yang berjalan lama, baik baca atau tulis
- Beban tulis yang berat

Tindakan

Kami merekomendasikan tindakan yang berbeda bergantung pada penyebab wawasan Anda.

Topik

- Jangan memulai operasi apa pun yang melibatkan penonaktifan basis data hingga daftar riwayat InnoDB berkurang
- Mengidentifikasi dan mengakhiri transaksi yang berjalan lama
- Mengidentifikasi host teratas dan pengguna teratas dengan menggunakan Wawasan Performa.

Jangan memulai operasi apa pun yang melibatkan penonaktifan basis data hingga daftar riwayat InnoDB berkurang

Karena daftar riwayat InnoDB yang panjang memperlambat penonaktifan basis data, kurangi ukuran daftar sebelum memulai operasi yang melibatkan penonaktifan basis data. Operasi ini meliputi peningkatan basis data versi utama.

Mengidentifikasi dan mengakhiri transaksi yang berjalan lama

Anda dapat menemukan transaksi yang berjalan lama dengan mengueri `information_schema.innodb_trx`.

> **Note**
>
> Pastikan juga untuk mencari transaksi jangka panjang pada replika baca.

Untuk mengidentifikasi dan mengakhiri transaksi yang berjalan lama

1. Di klien SQL Anda, jalankan kueri berikut:

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
      b.USER,
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
ORDER BY trx_started
```

2. Akhiri setiap transaksi yang berjalan lama dengan prosedur mysql.rds_kill yang disimpan.

Mengidentifikasi host teratas dan pengguna teratas dengan menggunakan Wawasan Performa.

Optimalkan transaksi agar sejumlah besar baris yang dimodifikasi segera dilakukan.

Metrik terkait

Metrik berikut terkait dengan wawasan ini:

- `trx_rseg_history_len`— Metrik penghitung ini dapat dilihat di Performance Insights, serta tabel. `INFORMATION_SCHEMA.INNODB_METRICS` Untuk informasi selengkapnya, lihat tabel metrik InnoDB INFORMATION_SCHEMA dalam dokumentasi MySQL.
- `RollbackSegmentHistoryListLength`— CloudWatch Metrik Amazon ini mengukur log pembatalan yang mencatat transaksi yang dilakukan dengan catatan yang ditandai hapus. Catatan ini dijadwalkan untuk diproses oleh operasi purge InnoDB. Metrik `trx_rseg_history_len` memiliki nilai yang sama dengan`RollbackSegmentHistoryListLength`.
- `PurgeBoundary`— Nomor transaksi yang memungkinkan pembersihan InnoDB. Jika CloudWatch metrik ini tidak maju untuk jangka waktu yang lama, ini merupakan indikasi yang baik bahwa pembersihan InnoDB diblokir oleh transaksi yang berjalan lama. Untuk menyelidiki, periksa transaksi aktif pada cluster DB MySQL Aurora Anda. Metrik ini hanya tersedia untuk Aurora MySQL versi 2.11 dan lebih tinggi, dan versi 3.08 dan lebih tinggi.
- `PurgeFinishedPoint`— Nomor transaksi hingga pembersihan InnoDB dilakukan. CloudWatch Metrik ini dapat membantu Anda memeriksa seberapa cepat proses pembersihan InnoDB. Metrik ini hanya tersedia untuk Aurora MySQL versi 2.11 dan lebih tinggi, dan versi 3.08 dan lebih tinggi.
- `TransactionAgeMaximum`— Usia transaksi berjalan aktif tertua. CloudWatch Metrik ini hanya tersedia untuk Aurora MySQL versi 3.08 dan lebih tinggi.
- `TruncateFinishedPoint`— Nomor transaksi hingga pemotongan undo dilakukan. CloudWatch Metrik ini hanya tersedia untuk Aurora MySQL versi 2.11 dan lebih tinggi, dan versi 3.08 dan lebih tinggi.

Untuk informasi selengkapnya tentang CloudWatch metrik, lihatMetrik tingkat instans untuk Amazon Aurora.

## Basis data membuat tabel sementara pada disk

Penggunaan tabel sementara pada disk Anda baru-baru ini meningkat secara signifikan, hingga *`percentage`*. Basis data membuat sekitar *`number`* tabel sementara per detik. Hal ini dapat memengaruhi performa dan meningkatkan operasi disk pada *`db-instance`*.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab masalah ini
- Tindakan
- Metrik terkait


### Versi mesin yang didukung

Informasi wawasan ini didukung untuk semua versi Aurora MySQL.

### Konteks

Terkadang server MySQL perlu membuat tabel sementara internal saat memproses kueri. Aurora MySQL dapat menyimpan tabel sementara internal dalam memori, tempat tabel dapat diproses oleh mesin penyimpanan TempTable atau MEMORY, atau disimpan pada disk oleh InnoDB. Untuk informasi selengkapnya, lihat Penggunaan Tabel Sementara Internal di MySQL di Panduan Referensi MySQL.

### Kemungkinan penyebab masalah ini

Peningkatan tabel sementara pada disk menunjukkan penggunaan kueri yang kompleks. Jika memori yang dikonfigurasi tidak cukup untuk menyimpan tabel sementara dalam memori, Aurora MySQL akan membuat tabel pada disk. Hal ini dapat memengaruhi performa dan meningkatkan operasi disk.

### Tindakan

Kami merekomendasikan tindakan yang berbeda bergantung pada penyebab wawasan Anda.

- Untuk Aurora MySQL versi 3, sebaiknya Anda menggunakan mesin penyimpanan TempTable.
- Optimalkan kueri Anda untuk menampilkan lebih sedikit data dengan memilih kolom yang diperlukan saja.

  Jika Anda mengaktifkan Skema Performa dengan semua `statement` instrumen diaktifkan dan waktunya diatur, Anda dapat mengkueri `SYS.statements_with_temp_tables` untuk mengambil daftar kueri yang menggunakan tabel sementara. Untuk informasi selengkapnya, lihat Prasyarat untuk Menggunakan Skema sys dalam dokumentasi MySQL.

- Pertimbangkan untuk mengindeks kolom yang digunakan dalam operasi pengurutan dan pengelompokan.
- Tulis ulang kueri Anda untuk menghindari kolom `BLOB` dan `TEXT`. Kolom ini selalu menggunakan disk.
- Setel parameter basis data berikut: `tmp_table_size` dan`max_heap_table_size`.

  Nilai default untuk parameter ini adalah 16 MiB. Saat menggunakan mesin penyimpanan MEMORY untuk tabel sementara dalam memori, ukuran maksimumnya ditentukan oleh nilai `tmp_table_size` atau `max_heap_table_size`, mana yang lebih kecil. Jika ukuran maksimum ini tercapai, MySQL secara otomatis mengubah tabel sementara internal dalam memori ke tabel sementara internal pada disk InnoDB. Untuk informasi selengkapnya, lihat Menggunakan mesin penyimpanan TempTable di Amazon RDS for MySQL dan Amazon Aurora MySQL.

  > **Note**
  >
  > Jika tabel MEMORY dibuat secara eksplisit dengan CREATE TABLE, hanya variabel `max_heap_table_size` yang menentukan seberapa besar tabel bisa berkembang. Konversi ke format pada disk juga tidak terjadi.

Metrik terkait

Metrik Wawasan Performa berikut terkait dengan wawasan ini:

- Created_tmp_disk_tables
- Created_tmp_tables

Untuk informasi selengkapnya, lihat Created_tmp_disk_tables dalam dokumentasi MySQL.

# Kueri paralel untuk Amazon Aurora My SQL

Topik ini menjelaskan optimasi kinerja kueri paralel untuk Amazon Aurora My SQL -Compatible Edition. Fitur ini menggunakan jalur pemrosesan khusus untuk kueri tertentu yang sarat data, dengan memanfaatkan arsitektur penyimpanan bersama Aurora. Kueri paralel bekerja paling baik dengan cluster Aurora My SQL DB yang memiliki tabel dengan jutaan baris dan kueri analitik yang membutuhkan waktu beberapa menit atau jam untuk diselesaikan.

Topik


- Ikhtisar query paralel untuk Aurora My SQL
- Membuat cluster DB query paralel di Aurora My SQL
- Menghidupkan dan menonaktifkan kueri paralel di Aurora My SQL
- Mengoptimalkan query paralel di Aurora My SQL
- Memverifikasi pernyataan mana yang menggunakan query paralel untuk Aurora My SQL
- Memantau query paralel untuk Aurora My SQL
- SQLkonstruksi untuk kueri paralel di Aurora My SQL


## Ikhtisar query paralel untuk Aurora My SQL

Aurora My SQL parallel query adalah optimasi yang memparalelkan beberapa I/O dan komputasi yang terlibat dalam pemrosesan kueri intensif data. Pekerjaan yang diparalelkan mencakup pengambilan baris dari penyimpanan, ekstraksi nilai kolom, dan penentuan baris yang cocok dengan ketentuan dalam klausa `WHERE` dan klausa sambungan. Pekerjaan sarat data ini didelegasikan (dalam istilah pengoptimalan basis data, ditekan) ke beberapa simpul di lapisan penyimpanan terdistribusi Aurora. Tanpa query paralel, setiap kueri membawa semua data yang dipindai ke satu node dalam cluster Aurora SQL My (node kepala) dan melakukan semua pemrosesan kueri di sana.

> **Tip**
>
> Mesin SQL database Postgre juga memiliki fitur yang disebut “query paralel.” Fitur tersebut tidak terkait dengan kueri paralel Aurora.

Ketika fitur query paralel diaktifkan, SQL mesin Aurora Saya secara otomatis menentukan kapan kueri dapat memperoleh manfaat, tanpa memerlukan SQL perubahan seperti petunjuk atau

atribut tabel. Pada bagian berikut, Anda dapat menemukan penjelasan tentang kapan kueri paralel diterapkan pada kueri. Anda juga dapat menemukan cara memastikan bahwa kueri paralel diterapkan di tempat yang memberi banyak keuntungan.

> **Note**
>
> Pengoptimalan kueri paralel sangat menguntungkan bagi kueri yang berlangsung lama yang memerlukan waktu beberapa menit atau jam untuk diselesaikan. Aurora My SQL umumnya tidak melakukan optimasi query parallel untuk query murah. Aurora juga umumnya tidak melakukan pengoptimalan kueri paralel jika teknik pengoptimalan lain lebih masuk akal, seperti query caching, buffer pool caching, atau pencarian indeks. Jika Anda mendapati kueri paralel tidak digunakan seperti yang Anda harapkan, lihat Memverifikasi pernyataan mana yang menggunakan query paralel untuk Aurora My SQL.

Topik


- Manfaat
- Arsitektur
- Prasyarat
- Pembatasan
- Biaya I/O dengan kueri paralel


## Manfaat

Dengan query paralel, Anda dapat menjalankan kueri analitik intensif data pada tabel Aurora My. SQL Dalam banyak kasus, Anda bisa mendapatkan peningkatan order-of-magnitude kinerja atas pembagian kerja tradisional untuk pemrosesan kueri.

Manfaat kueri paralel antara lain:

- Meningkatkan performa I/O, karena paralelisasi permintaan baca fisik di beberapa simpul penyimpanan.
- Mengurangi lalu lintas jaringan. Aurora tidak mengirim keseluruhan halaman data dari simpul penyimpanan ke simpul kepala lalu memfilter baris dan kolom yang tidak diperlukan sesudahnya. Sebaliknya, Aurora mengirim urutan ringkas yang hanya berisi nilai kolom yang diperlukan untuk set hasil.

- Mengurangi CPU penggunaan pada node kepala, karena menekan pemrosesan fungsi, pemfilteran baris, dan proyeksi kolom untuk klausa. `WHERE`
- Menurunkan tekanan memori pada pool buffer. Halaman yang diproses oleh kueri paralel tidak ditambahkan ke dalam pool buffer. Pendekatan ini mengurangi peluang pemindaian sarat data mengosongkan data yang sering digunakan dari pool buffer.
- Berpotensi mengurangi duplikasi data dalam pipeline ekstrak, transformasi, load (ETL) Anda, dengan membuatnya praktis untuk melakukan kueri analitik yang berjalan lama pada data yang ada.

## Arsitektur

Fitur query paralel menggunakan prinsip arsitektur utama Aurora MySQL: memisahkan mesin database dari subsistem penyimpanan, dan mengurangi lalu lintas jaringan dengan merampingkan protokol komunikasi. Aurora My SQL menggunakan teknik ini untuk mempercepat operasi intensif penulisan seperti pemrosesan redo log. Kueri paralel menerapkan prinsip yang sama untuk operasi baca.

> **Note**
>
> Arsitektur Aurora My SQL parallel query berbeda dari fitur bernama serupa di sistem database lainnya. Aurora Permintaan SQL paralel saya tidak melibatkan multiprocessing simetris (SMP) sehingga tidak bergantung pada CPU kapasitas server database. Pemrosesan paralel terjadi di lapisan penyimpanan, terlepas dari Aurora My SQL server yang berfungsi sebagai koordinator kueri.

Secara default, tanpa kueri paralel, pemrosesan untuk kueri Aurora mencakup pengiriman data mentah ke satu simpul di dalam klaster Aurora (simpul kepala). Aurora kemudian melakukan semua pemrosesan lebih lanjut untuk kueri tersebut di dalam thread tunggal pada simpul tunggal tersebut. Dengan query paralel, sebagian besar pekerjaan intensif I/O dan CPU -intensif ini didelegasikan ke node di lapisan penyimpanan. Hanya baris ringkas set hasil yang dikirim kembali ke simpul kepala, dengan baris yang sudah difilter, dan nilai kolom sudah diekstraksi dan ditransformasikan. Manfaat kinerja berasal dari pengurangan lalu lintas jaringan, pengurangan CPU penggunaan pada node kepala, dan paralelisasi I/O di seluruh node penyimpanan. Jumlah I/O paralel, pemfilteran, dan proyeksi tidak berhubungan dengan jumlah instans DB dalam klaster Aurora yang menjalankan kueri.

## Prasyarat

Menggunakan semua fitur query paralel memerlukan cluster Aurora My SQL DB yang menjalankan versi 2.09 atau lebih tinggi. Jika Anda sudah memiliki klaster yang ingin Anda gunakan dengan kueri paralel, Anda dapat meningkatkannya ke versi yang kompatibel dan mengaktifkan kueri paralel sesudahnya. Dalam hal ini, pastikan untuk mengikuti prosedur peningkatan dalam Pertimbangan peningkatan untuk kueri paralel karena nama pengaturan konfigurasi dan nilai default berbeda di versi yang lebih baru ini.

Instans DB dalam klaster Anda harus menggunakan kelas instans `db.r*`.

Pastikan bahwa pengoptimalan hash join diaktifkan untuk klaster Anda. Untuk mempelajari caranya, lihat Mengaktifkan hash join untuk klaster kueri paralel.

Untuk menyesuaikan parameter seperti `aurora_parallel_query` dan `aurora_disable_hash_join`, Anda harus memiliki grup parameter kustom yang Anda gunakan untuk klaster Anda. Anda dapat menentukan parameter ini secara terpisah untuk setiap instans DB menggunakan grup parameter DB. Namun, kami menyarankan Anda untuk menetapkannya dalam grup parameter klaster DB. Dengan demikian, semua instans DB di klaster Anda menerima pengaturan yang sama untuk parameter ini.

## Pembatasan

Batasan berikut berlaku untuk fitur kueri paralel:

- Kueri paralel tidak didukung dengan Aurora I/O-Optimized Konfigurasi penyimpanan cluster DB.
- Anda tidak dapat menggunakan kueri paralel dengan kelas instans db.t2 atau db.t3. Batasan ini berlaku bahkan jika Anda meminta kueri paralel menggunakan variabel session `aurora_pq_force`.
- Kueri paralel tidak berlaku pada tabel yang menggunakan format baris `COMPRESSED` atau `REDUNDANT`. Gunakan format baris `COMPACT` atau `DYNAMIC` untuk tabel yang akan Anda gunakan dengan kueri paralel.
- Aurora menggunakan algoritma berbasis biaya untuk menentukan apakah akan menggunakan mekanisme query paralel untuk setiap pernyataan. SQL Menggunakan SQL konstruksi tertentu dalam pernyataan dapat mencegah query paralel atau membuat query paralel tidak mungkin untuk pernyataan itu. Untuk informasi tentang kompatibilitas SQL konstruksi dengan query paralel, lihatSQLkonstruksi untuk kueri paralel di Aurora My SQL.

- Setiap instans DB Aurora dapat menjalankan sesi kueri paralel hanya dalam jumlah tertentu pada satu waktu. Jika kueri memiliki beberapa bagian yang menggunakan kueri paralel, seperti subkueri, sambungan, atau operator `UNION`, fase-fase tersebut berjalan secara berurutan. Pernyataan ini hanya dihitung sebagai satu sesi kueri paralel pada satu waktu. Anda dapat memantau jumlah sesi aktif menggunakan variabel status kueri paralel. Anda dapat memeriksa batas sesi serentak untuk instans DB tertentu dengan melakukan kueri variabel status `Aurora_pq_max_concurrent_requests`.
- Kueri paralel tersedia di semua Wilayah AWS yang didukung Aurora. Untuk sebagian besar Wilayah AWS, minimum Aurora My SQL version yang diperlukan untuk menggunakan query paralel adalah 2.09.
- Kueri paralel dirancang untuk meningkatkan performa kueri sarat data. Ini tidak dirancang untuk kueri ringan.
- Kami menyarankan Anda menggunakan node pembaca untuk SELECT pernyataan, terutama yang intensif data.

### Biaya I/O dengan kueri paralel

Jika SQL klaster Aurora Saya menggunakan kueri paralel, Anda mungkin melihat peningkatan `VolumeReadIOPS` nilai. Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kueri cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.

Biaya I/O kueri paralel untuk kueri Anda diukur pada lapisan penyimpanan, dan akan sama atau lebih besar dengan saat kueri paralel diaktifkan. Anda akan mendapat manfaat berupa peningkatan performa kueri. Ada dua alasan untuk biaya I/O yang berpotensi lebih tinggi dengan kueri paralel:

- Bahkan jika beberapa data dalam tabel berada di pool buffer, kueri paralel mengharuskan semua data dipindai di lapisan penyimpanan, sehingga menimbulkan biaya I/O.
- Menjalankan kueri paralel tidak menyiapkan pool buffer. Akibatnya, menjalankan kueri paralel yang sama secara berurutan menimbulkan biaya I/O penuh.

## Membuat cluster DB query paralel di Aurora My SQL

Untuk membuat SQL klaster Aurora My dengan query paralel, menambahkan instance baru ke dalamnya, atau melakukan operasi administratif lainnya, Anda menggunakan AWS CLI teknik yang sama AWS Management Console dan yang Anda lakukan dengan cluster Aurora My lainnya. SQL

Anda dapat membuat klaster baru untuk bekerja dengan kueri paralel. Anda juga dapat membuat cluster DB untuk bekerja dengan query paralel dengan memulihkan dari snapshot dari cluster Aurora DB yang SQL kompatibel dengan Saya. Jika Anda tidak terbiasa dengan proses untuk membuat SQL cluster Aurora My baru, Anda dapat menemukan informasi latar belakang dan prasyarat di. Membuat klaster DB Amazon Aurora

Ketika Anda memilih versi SQL mesin Aurora My, kami sarankan Anda memilih yang terbaru yang tersedia. Saat ini, semua SQL versi Aurora My yang tersedia mendukung query paralel. Anda memiliki lebih banyak fleksibilitas untuk mengaktifkan dan menonaktifkan query paralel, atau menggunakan query paralel dengan cluster yang ada, jika Anda menggunakan versi terbaru.

Baik Anda membuat cluster baru atau memulihkan dari snapshot, Anda menggunakan teknik yang sama untuk menambahkan instance DB baru yang Anda lakukan dengan cluster Aurora My lainnya. SQL

Anda dapat membuat cluster kueri paralel menggunakan RDS konsol Amazon atau file AWS CLI.

Daftar Isi


- Membuat klaster kueri paralel menggunakan konsol
- Membuat cluster query paralel menggunakan CLI


## Membuat klaster kueri paralel menggunakan konsol

Anda dapat membuat klaster kueri paralel baru dengan konsol seperti dijelaskan berikut ini.

Untuk membuat cluster query paralel dengan AWS Management Console

1. Ikuti AWS Management Console prosedur umum diMembuat klaster DB Amazon Aurora.
2. Untuk tipe Engine, pilih Aurora My. SQL
3. Untuk Konfigurasi tambahan, pilih grup parameter yang Anda buat untuk grup parameter klaster DB. Menggunakan grup parameter khusus seperti itu diperlukan untuk Aurora My SQL 2.09 dan yang lebih tinggi. Di grup parameter klaster DB Anda, tentukan pengaturan parameter `aurora_parallel_query=ON` dan `aurora_disable_hash_join=OFF`. Tindakan tersebut akan mengaktifkan kueri paralel untuk klaster, dan mengaktifkan pengoptimalan hash join yang bekerja bersama kueri paralel.

Untuk memverifikasi bahwa klaster baru dapat menggunakan kueri paralel

1. Buat klaster menggunakan teknik sebelumnya.
2. (Untuk Aurora SQL Versi saya 2 atau 3) Periksa apakah pengaturan `aurora_parallel_query` konfigurasi benar.

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query |
+-------------------------+
|                       1 |
+-------------------------+
```

3. (Untuk Aurora SQL Versi saya 2) Periksa apakah `aurora_disable_hash_join` pengaturannya salah.

```
mysql> select @@aurora_disable_hash_join;
+----------------------------+
| @@aurora_disable_hash_join |
+----------------------------+
|                          0 |
+----------------------------+
```

4. Dengan beberapa tabel besar dan kueri sarat data, periksa rencana kueri untuk mengonfirmasi bahwa beberapa kueri Anda menggunakan pengoptimalan kueri paralel. Untuk melakukannya, ikuti prosedur dalam Memverifikasi pernyataan mana yang menggunakan query paralel untuk Aurora My SQL.

## Membuat cluster query paralel menggunakan CLI

Anda dapat membuat cluster query paralel baru dengan CLI seperti yang dijelaskan berikut.

Untuk membuat cluster query paralel dengan AWS CLI

1. (Opsional) Periksa Aurora SQL Versi saya mana yang kompatibel dengan cluster kueri paralel. Untuk melakukannya, gunakan perintah `describe-db-engine-versions` dan periksa nilai bidang `SupportsParallelQuery`. Sebagai contoh, lihat Memeriksa kompatibilitas Aurora My SQL version untuk query paralel.

2. (Opsional) Buat grup parameter klaster DB kustom dengan pengaturan `aurora_parallel_query=ON` dan `aurora_disable_hash_join=OFF`. Gunakan perintah seperti berikut ini.

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-
mysql8.0 --db-cluster-parameter-group-name pq-enabled-80-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-
reboot
```

   Jika Anda melakukan langkah ini, tentukan opsi `--db-cluster-parameter-group-name` *`my_cluster_parameter_group`* dalam pernyataan `create-db-cluster` selanjutnya. Ganti nama grup parameter Anda. Jika menghapus langkah ini, Anda membuat grup parameter dan mengaitkannya dengan klaster di lain waktu, seperti dijelaskan di Menghidupkan dan menonaktifkan kueri paralel di Aurora My SQL.

3. Ikuti AWS CLI prosedur umum diMembuat klaster DB Amazon Aurora.

4. Tentukan rangkaian opsi berikut:

   - Untuk opsi `--engine`, gunakan `aurora-mysql`. Nilai-nilai ini menghasilkan cluster query paralel yang kompatibel dengan My SQL 5.7 atau 8.0.
   - Untuk pilihan `--db-cluster-parameter-group-name`, tentukan nama grup parameter klaster DB yang Anda buat dan tentukan nilai parameter `aurora_parallel_query=ON`. Jika Anda menghilangkan pilihan ini, Anda dapat membuat klaster dengan kelompok parameter default, kemudian memodifikasinya untuk menggunakan grup parameter kustom tersebut.
   - Untuk `--engine-version` opsi, gunakan SQL versi Aurora My yang kompatibel dengan query paralel. Gunakan prosedur dari Mengoptimalkan query paralel di Aurora My SQL untuk mendapatkan daftar versi jika diperlukan.

     Contoh kode berikut menunjukkan caranya. Gantikan nilai Anda sendiri untuk masing-masing variabel lingkungan seperti*`$CLUSTER_ID`*. Contoh ini juga menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi

dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
  --engine aurora-mysql --engine-version 8.0.mysql_aurora.3.04.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. Verifikasi bahwa klaster yang Anda buat atau pulihkan memiliki fitur kueri paralel yang tersedia.

   Periksa apakah pengaturan konfigurasi `aurora_parallel_query` ada. Jika pengaturan ini memiliki nilai 1, kueri paralel siap Anda gunakan. Jika pengaturan ini memiliki nilai 0, atur menjadi 1 sebelum Anda dapat menggunakan kueri paralel. Berapa pun nilainya, klaster ini mampu melakukan kueri paralel.

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

Untuk mengembalikan snapshot ke cluster query paralel dengan AWS CLI

1. Periksa SQL versi Aurora My mana yang kompatibel dengan cluster kueri paralel. Untuk melakukannya, gunakan perintah `describe-db-engine-versions` dan periksa nilai bidang `SupportsParallelQuery`. Sebagai contoh, lihat Memeriksa kompatibilitas Aurora My SQL version untuk query paralel. Tentukan versi yang akan digunakan untuk klaster yang dipulihkan.
2. Temukan snapshot cluster yang SQL kompatibel dengan Aurora My.
3. Ikuti AWS CLI prosedur umum diMemulihkan dari snapshot klaster DB.

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier mynewdbcluster \
  --snapshot-identifier mydbclustersnapshot \
  --engine aurora-mysql
```

4. Verifikasi bahwa klaster yang Anda buat atau pulihkan memiliki fitur kueri paralel yang tersedia. Gunakan prosedur verifikasi yang sama seperti di Membuat cluster query paralel menggunakan CLI.

## Menghidupkan dan menonaktifkan kueri paralel di Aurora My SQL

Saat query paralel diaktifkan, Aurora My SQL menentukan apakah akan menggunakannya saat runtime untuk setiap kueri. Dalam kasus gabungan, serikat pekerja, subkueri, dan sebagainya, Aurora My menentukan SQL apakah akan menggunakan query paralel saat runtime untuk setiap blok kueri. Untuk detailnya, lihat Memverifikasi pernyataan mana yang menggunakan query paralel untuk Aurora My SQL dan SQLkonstruksi untuk kueri paralel di Aurora My SQL.

Anda dapat mengaktifkan dan menonaktifkan kueri paralel secara dinamis pada tingkat global dan sesi untuk instans DB menggunakan opsi aurora_parallel_query. Anda dapat mengubah pengaturan `aurora_parallel_query` di grup klaster DB Anda untuk mengaktifkan atau menonaktifkan kueri paralel secara default.

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

Untuk mengganti parameter `aurora_parallel_query` pada tingkat sesi, gunakan metode standar untuk mengubah pengaturan konfigurasi klien. Misalnya, Anda dapat melakukannya melalui baris `mysql` perintah atau di dalam ODBC aplikasi JDBC atau. Perintah pada standar SQL Klien saya adalah`set session aurora_parallel_query = {'ON'/'OFF'}`. Anda juga dapat menambahkan parameter tingkat sesi ke JDBC konfigurasi atau dalam kode aplikasi Anda untuk mengaktifkan atau mematikan query paralel secara dinamis.

Anda dapat secara permanen mengubah pengaturan untuk parameter `aurora_parallel_query`, baik untuk instans DB tertentu atau untuk seluruh klaster Anda. Jika Anda menetapkan nilai parameter dalam grup parameter DB, nilai tersebut hanya berlaku untuk instans DB tertentu dalam klaster Anda. Jika Anda menetapkan nilai parameter dalam grup parameter klaster DB, semua instans DB di klaster tersebut menerima pengaturan yang sama. Untuk mengganti parameter `aurora_parallel_query`, gunakan teknik untuk bekerja dengan grup parameter, seperti dijelaskan dalam . Ikuti langkah-langkah ini:

1. Buat grup parameter klaster kustom (disarankan) atau grup parameter DB kustom.
2. Dalam grup parameter ini, perbarui `parallel_query` dengan nilai yang Anda inginkan.
3. Bergantung pada apakah Anda membuat grup parameter klaster DB atau grup parameter DB, pasang grup parameter ke klaster Aurora Anda atau ke instans DB spesifik tempat Anda berencana menggunakan fitur kueri paralel.

> **Tip**
>
> Karena `aurora_parallel_query` adalah parameter dinamis, parameter ini tidak memerlukan mulai ulang klaster setelah pengaturan ini diubah. Namun, koneksi apa pun yang menggunakan kueri paralel sebelum beralih opsi akan terus melakukannya sampai koneksi ditutup, atau instans di-boot ulang.

Anda dapat memodifikasi parameter query paralel dengan menggunakan API operasi M odifyDBCluster ParameterGroup atau M odifyDBParameter Group atau AWS Management Console.

Anda dapat mengaktifkan gabungan hash untuk cluster kueri paralel, mengaktifkan dan menonaktifkan kueri paralel menggunakan RDS konsol Amazon atau AWS CLI, dan mengganti pengoptimal kueri paralel.

Daftar Isi


- Mengaktifkan hash join untuk klaster kueri paralel
- Mengaktifkan dan menonaktifkan kueri paralel menggunakan konsol
- Menghidupkan dan mematikan query paralel menggunakan CLI
- Menimpa pengoptimal kueri paralel


## Mengaktifkan hash join untuk klaster kueri paralel

Kueri paralel biasanya digunakan untuk jenis kueri sarat sumber daya yang diuntungkan dari pengoptimalan hash join. Oleh karena itu, sebaiknya pastikan hash join diaktifkan untuk klaster yang Anda rencanakan akan menggunakan kueri paralel. Untuk informasi tentang cara menggunakan hash join secara efektif, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

## Mengaktifkan dan menonaktifkan kueri paralel menggunakan konsol

Anda dapat mengaktifkan atau menonaktifkan kueri paralel pada tingkat instans DB atau tingkat klaster DB dengan menggunakan grup parameter.

Untuk mengaktifkan atau mematikan query paralel untuk cluster DB dengan AWS Management Console

1. Buat grup parameter kustom, seperti yang dijelaskan di .
2. Perbarui aurora_parallel_query ke 1 (diaktifkan) atau 0 (dinonaktifkan). Untuk klaster dengan fitur kueri paralel yang tersedia, aurora_parallel_query dinonaktifkan secara default.
3. Jika Anda menggunakan grup parameter klaster kustom, lampirkan grup tersebut ke klaster Aurora DB tempat Anda berencana menggunakan fitur kueri paralel. Jika Anda menggunakan grup parameter DB kustom, lampirkan grup tersebut ke satu atau lebih instans DB dalam klaster. Kami menyarankan menggunakan grup parameter klaster. Melakukan hal tersebut dapat memastikan semua instans DB di klaster memiliki pengaturan yang sama untuk kueri paralel dan fitur terkait seperti hash join.

## Menghidupkan dan mematikan query paralel menggunakan CLI

Anda dapat memodifikasi parameter kueri paralel dengan menggunakan perintah `modify-db-cluster-parameter-group` atau `modify-db-parameter-group`. Pilih perintah yang sesuai, tergantung pada apakah Anda menentukan nilai `aurora_parallel_query` melalui grup parameter klaster DB atau grup parameter DB.

Untuk mengaktifkan atau mematikan query paralel untuk cluster DB dengan CLI

- Ubah parameter kueri paralel menggunakan perintah `modify-db-cluster-parameter-group`. Gunakan perintah seperti berikut. Ganti dengan nama yang sesuai untuk grup parameter kustom Anda. Ganti `ON` atau `OFF` untuk bagian `ParameterValue` dari opsi `--parameters`.

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}
```

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

Anda juga dapat mengaktifkan atau mematikan query paralel di tingkat sesi, misalnya melalui baris `mysql` perintah atau dalam ODBC aplikasi JDBC atau. Untuk melakukannya, gunakan metode standar untuk mengubah pengaturan konfigurasi klien. Misalnya, perintah pada standar SQL Klien saya adalah `set session aurora_parallel_query = {'ON'/'OFF'}` untuk Aurora My. SQL

Anda juga dapat menambahkan parameter tingkat sesi ke JDBC konfigurasi atau dalam kode aplikasi Anda untuk mengaktifkan atau mematikan query paralel secara dinamis.

### Menimpa pengoptimal kueri paralel

Anda dapat menggunakan variabel sesi `aurora_pq_force` untuk menimpa pengoptimal kueri paralel dan meminta kueri paralel untuk setiap kueri. Kami menyarankan Anda melakukan tindakan ini hanya untuk tujuan pengujian. Contoh berikut ini menunjukkan cara menggunakan `aurora_pq_force` dalam sebuah sesi.

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

Untuk mematikan penimpaan, lakukan hal berikut:

```
set SESSION aurora_pq_force = OFF;
```

## Mengoptimalkan query paralel di Aurora My SQL

Untuk mengoptimalkan cluster DB Anda untuk query paralel, pertimbangkan cluster DB mana yang akan mendapat manfaat dari query paralel dan apakah akan meningkatkan untuk query paralel. Kemudian, atur beban kerja Anda dan buat objek skema untuk query paralel.

Daftar Isi


- Merencanakan klaster kueri paralel
  - Memeriksa kompatibilitas Aurora My SQL version untuk query paralel
- Pertimbangan peningkatan untuk kueri paralel
  - Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL

  - Upgrade ke Aurora SQL My 2.09 dan lebih tinggi
- Penyempurnaan performa untuk kueri paralel
- Membuat objek skema untuk memanfaatkan kueri paralel

## Merencanakan klaster kueri paralel

Perencanaan untuk klaster DB dengan kueri paralel yang diaktifkan mengharuskan beberapa pilihan. Ini termasuk melakukan langkah-langkah penyiapan (baik membuat atau memulihkan cluster Aurora SQL My lengkap) dan memutuskan seberapa luas untuk mengaktifkan kueri paralel di seluruh cluster DB Anda.

Pertimbangkan hal berikut sebagai bagian dari perencanaan:

- Jika Anda menggunakan Aurora My SQL yang kompatibel dengan My SQL 5.7, Anda harus membuat cluster yang disediakan. Kemudian Anda mengaktifkan kueri paralel menggunakan parameter `aurora_parallel_query`.

  Jika Anda memiliki SQL cluster Aurora My yang sudah ada, Anda tidak perlu membuat cluster baru untuk menggunakan query paralel. Anda dapat mengaitkan klaster Anda, atau instans DB tertentu dalam klaster, dengan grup parameter yang mengaktifkan parameter `aurora_parallel_query`. Dengan melakukannya, Anda dapat mengurangi waktu dan upaya untuk mengatur data yang relevan untuk digunakan dengan kueri paralel.
- Rencanakan setiap tabel besar yang perlu Anda atur ulang agar Anda dapat menggunakan kueri paralel saat mengaksesnya. Anda mungkin perlu membuat versi baru dari beberapa tabel besar di mana kueri paralel akan bermanfaat. Misalnya, Anda mungkin perlu menghapus indeks pencarian teks penuh. Untuk detailnya, lihat Membuat objek skema untuk memanfaatkan kueri paralel.

### Memeriksa kompatibilitas Aurora My SQL version untuk query paralel

Untuk memeriksa Aurora SQL Versi saya mana yang kompatibel dengan cluster kueri paralel, gunakan `describe-db-engine-versions` AWS CLI perintah dan periksa nilai bidang. `SupportsParallelQuery` Contoh kode berikut menunjukkan cara memeriksa kombinasi mana yang tersedia untuk cluster query paralel di AWS Wilayah tertentu. Pastikan untuk menentukan string parameter `--query` secara lengkap pada satu baris.

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

Perintah sebelumnya menghasilkan output yang serupa dengan yang berikut ini. Outputnya mungkin bervariasi tergantung pada SQL versi Aurora My yang tersedia di Wilayah yang ditentukan AWS .

```
5.7.mysql_aurora.2.11.1
5.7.mysql_aurora.2.11.2
5.7.mysql_aurora.2.11.3
5.7.mysql_aurora.2.11.4
5.7.mysql_aurora.2.11.5
5.7.mysql_aurora.2.11.6
5.7.mysql_aurora.2.12.0
5.7.mysql_aurora.2.12.1
5.7.mysql_aurora.2.12.2
5.7.mysql_aurora.2.12.3
5.7.mysql_aurora.2.12.4
8.0.mysql_aurora.3.04.0
8.0.mysql_aurora.3.04.1
8.0.mysql_aurora.3.04.2
8.0.mysql_aurora.3.04.3
8.0.mysql_aurora.3.05.2
8.0.mysql_aurora.3.06.0
8.0.mysql_aurora.3.06.1
8.0.mysql_aurora.3.07.0
8.0.mysql_aurora.3.07.1
```

Setelah mulai menggunakan kueri paralel dengan suatu klaster, Anda dapat memantau performa dan menghilangkan batasan pada penggunaan kueri paralel. Untuk petunjuk langkah demi langkah, lihat Penyempurnaan performa untuk kueri paralel.

## Pertimbangan peningkatan untuk kueri paralel

Tergantung versi asal dan tujuan saat Anda meningkatkan versi klaster kueri paralel, Anda mungkin menemukan penyempurnaan dalam jenis kueri yang dapat dioptimalkan oleh kueri paralel. Anda mungkin juga menemukan bahwa Anda tidak perlu menentukan parameter mode mesin khusus untuk kueri paralel. Bagian berikutnya menjelaskan pertimbangan saat Anda meningkatkan klaster dengan kueri paralel yang diaktifkan.

Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL

Beberapa SQL pernyataan, klausa, dan tipe data memiliki dukungan query paralel yang baru atau lebih baik dimulai di Aurora My versi 3. SQL Saat Anda melakukan peningkatan dari rilis sebelum versi 3, periksa apakah kueri tambahan dapat memperoleh manfaat dari pengoptimalan kueri paralel.

Untuk informasi tentang penyempurnaan kueri paralel ini, lihat Jenis data kolom, Tabel yang dipartisi, dan Fungsi agregat, klausa GROUP BY, dan klausa HAVING.

Jika Anda memutakhirkan cluster kueri paralel dari Aurora SQL My 2.08 atau yang lebih rendah, pelajari juga tentang perubahan cara mengaktifkan kueri paralel. Untuk mempelajarinya, baca Upgrade ke Aurora SQL My 2.09 dan lebih tinggi.

Di Aurora My SQL versi 3, optimasi bergabung hash diaktifkan secara default. Opsi konfigurasi `aurora_disable_hash_join` dari versi sebelumnya tidak digunakan.

Upgrade ke Aurora SQL My 2.09 dan lebih tinggi

Di Aurora My SQL versi 2.09 dan yang lebih tinggi, query paralel berfungsi untuk cluster yang disediakan dan tidak memerlukan parameter mode engine. `parallelquery` Dengan demikian, Anda tidak perlu membuat klaster baru atau memulihkan dari snapshot yang sudah ada untuk menggunakan kueri paralel dengan versi ini. Anda dapat menggunakan prosedur peningkatan yang dijelaskan dalam Memutakhirkan versi minor atau level patch cluster Aurora SQL My DB untuk meningkatkan klaster Anda ke versi tersebut. Anda dapat meningkatkan klaster yang lebih lama terlepas dari apakah itu berupa klaster kueri paralel atau klaster yang tersedia. Untuk mengurangi jumlah pilihan dalam Versi mesin Anda dapat memilih Tampilkan versi yang mendukung fitur kueri paralel untuk memfilter entri di menu tersebut. Kemudian pilih Aurora My SQL 2.09 atau lebih tinggi.

Setelah Anda memutakhirkan klaster kueri paralel sebelumnya ke Aurora My SQL 2.09 atau lebih tinggi, Anda mengaktifkan kueri paralel di klaster yang ditingkatkan. Kueri paralel dimatikan secara default dalam versi ini, dan prosedur untuk mengaktifkannya berbeda. Pengoptimalan hash join juga dinonaktifkan secara default dan harus diaktifkan secara terpisah. Oleh karena itu, pastikan Anda mengaktifkan pengaturan ini lagi setelah peningkatan. Untuk petunjuk tentang melakukannya, lihat Menghidupkan dan menonaktifkan kueri paralel di Aurora My SQL dan Mengaktifkan hash join untuk klaster kueri paralel.

Secara khusus, Anda mengaktifkan kueri paralel dengan menggunakan parameter konfigurasi `aurora_parallel_query=ON` dan `aurora_disable_hash_join=OFF`, bukan `aurora_pq_supported` dan `aurora_pq`. `aurora_pq`Parameter `aurora_pq_supported` dan tidak digunakan lagi di Aurora versi Saya yang lebih baru. SQL

Dalam klaster yang ditingkatkan, atribut `EngineMode` memiliki nilai `provisioned`, bukan `parallelquery`. Untuk memeriksa apakah query paralel tersedia untuk versi mesin tertentu, sekarang Anda memeriksa nilai `SupportsParallelQuery` bidang dalam output `describe-db-engine-versions` AWS CLI perintah. Dalam SQL versi Aurora My sebelumnya, Anda memeriksa keberadaan `parallelquery` dalam daftar. `SupportedEngineModes`

Setelah Anda meningkatkan ke Aurora My SQL versi 2.09 atau lebih tinggi, Anda dapat memanfaatkan fitur-fitur berikut. Fitur-fitur ini tidak tersedia untuk cluster kueri paralel yang menjalankan versi Aurora SQL My yang lebih lama.

- Wawasan Performa. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Pelacakan Mundur. Untuk informasi selengkapnya, lihat Melakukan backtracking klaster DB Aurora.
- Menghentikan dan memulai klaster. Untuk informasi selengkapnya, lihat Menghentikan dan memulai klaster DB Amazon Aurora.

## Penyempurnaan performa untuk kueri paralel

Untuk mengelola performa beban kerja dengan kueri paralel, pastikan kueri paralel digunakan untuk kueri di mana pengoptimalan ini paling membantu.

Untuk melakukannya, Anda dapat melakukan hal berikut:

- Pastikan tabel terbesar Anda kompatibel dengan kueri paralel. Anda mungkin mengubah properti tabel atau membuat ulang beberapa tabel sehingga kueri untuk tabel tersebut dapat memanfaatkan pengoptimalan kueri paralel. Untuk mempelajari caranya, lihat Membuat objek skema untuk memanfaatkan kueri paralel.
- Pantau kueri mana yang menggunakan kueri paralel. Untuk mempelajari caranya, lihat Memantau query paralel untuk Aurora My SQL.
- Verifikasi bahwa kueri paralel digunakan untuk kueri sarat data dan berlangsung lama, dan dengan tingkat keserentakan yang tepat untuk beban kerja Anda. Untuk mempelajari caranya, lihat Memverifikasi pernyataan mana yang menggunakan query paralel untuk Aurora My SQL.
- Sempurnakan SQL kode Anda untuk mengaktifkan kueri paralel agar diterapkan pada kueri yang Anda harapkan. Untuk mempelajari caranya, lihat SQLkonstruksi untuk kueri paralel di Aurora My SQL.

## Membuat objek skema untuk memanfaatkan kueri paralel

Sebelum Anda membuat atau memodifikasi tabel yang akan Anda gunakan untuk kueri paralel, pastikan untuk memahami persyaratan yang dijelaskan dalam Prasyarat dan Pembatasan.

Karena kueri paralel memerlukan tabel untuk menggunakan pengaturan `ROW_FORMAT=Compact` atau `ROW_FORMAT=Dynamic`, periksa pengaturan konfigurasi Aurora Anda untuk setiap perubahan pada opsi konfigurasi `INNODB_FILE_FORMAT`. Keluarkan pernyataan `SHOW TABLE STATUS` untuk mengonfirmasi format baris untuk semua tabel dalam basis data.

Sebelum mengubah skema Anda untuk mengaktifkan kueri paralel agar dapat bekerja dengan lebih banyak tabel, pastikan untuk melakukan pengujian. Pengujian Anda harus mengonfirmasi apakah kueri paralel menghasilkan peningkatan performa bersih untuk tabel tersebut. Selain itu, pastikan bahwa persyaratan skema untuk kueri paralel kompatibel dengan sasaran Anda.

Sebagai contoh, sebelum beralih dari `ROW_FORMAT=Compressed` ke `ROW_FORMAT=Compact` atau `ROW_FORMAT=Dynamic`, uji performa beban kerja untuk tabel asal dan baru. Selain itu, pertimbangkan potensi efek lainnya seperti peningkatan volume data.

## Memverifikasi pernyataan mana yang menggunakan query paralel untuk Aurora My SQL

Dalam operasi biasa, Anda tidak perlu melakukan tindakan khusus apa pun untuk memanfaatkan kueri paralel. Setelah kueri memenuhi persyaratan penting untuk kueri paralel, pengoptimal kueri secara otomatis memutuskan apakah akan menggunakan kueri paralel untuk setiap kueri tertentu.

Jika menjalankan percobaan di lingkungan pengembangan atau pengujian, Anda mungkin menemukan bahwa kueri paralel tidak digunakan karena tabel Anda terlalu kecil dalam jumlah baris atau volume data keseluruhan. Data untuk tabel mungkin juga sepenuhnya berada di dalam buffer pool, terutama untuk tabel yang baru Anda buat untuk melakukan percobaan.

Saat Anda memantau atau menyesuaikan performa klaster, pastikan untuk memutuskan apakah kueri paralel digunakan dalam konteks yang sesuai. Anda dapat menyesuaikan skema database, pengaturan, SQL kueri, atau bahkan topologi cluster dan pengaturan koneksi aplikasi untuk memanfaatkan fitur ini.

Untuk memeriksa apakah kueri menggunakan query paralel, periksa rencana kueri (juga dikenal sebagai “jelaskan rencana”) dengan menjalankan EXPLAINpernyataan. Untuk contoh bagaimana SQL pernyataan, klausa, dan ekspresi mempengaruhi `EXPLAIN` output untuk query paralel, lihat. SQLkonstruksi untuk kueri paralel di Aurora My SQL

Contoh berikut menunjukkan perbedaan antara rencana kueri tradisional dan rencana kueri paralel. Rencana penjelasan ini berasal dari Query 3 dari benchmark TPC -H. Banyak kueri sampel di seluruh

bagian ini menggunakan tabel dari kumpulan data TPC -H. Anda bisa mendapatkan definisi tabel, kueri, dan dbgen program yang menghasilkan data sampel dari situs web TPC-h.

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

Secara default, kueri tersebut mungkin memiliki rencana seperti berikut. Jika Anda tidak melihat hash join digunakan dalam rencana kueri, pastikan pengoptimalan diaktifkan terlebih dahulu.

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

Untuk Aurora My SQL versi 3, Anda mengaktifkan hash join di tingkat sesi dengan mengeluarkan pernyataan berikut.

```
SET optimizer_switch='block_nested_loop=on';
```

Untuk Aurora My SQL versi 2.09 dan yang lebih tinggi, Anda mengatur parameter `aurora_disable_hash_join` DB atau parameter cluster DB ke `0` (off). Jika `aurora_disable_hash_join` dinonaktifkan, nilai `optimizer_switch` berubah menjadi `hash_join=on`.

Setelah Anda mengaktifkan hash join, coba jalankan pernyataan `EXPLAIN` lagi. Untuk informasi tentang cara menggunakan hash join secara efektif, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

Dengan hash join diaktifkan tetapi kueri paralel dinonaktifkan, kueri mungkin memiliki rencana seperti berikut, yang menggunakan hash join, tetapi tidak menggunakan kueri paralel.

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
```

Setelah kueri paralel diaktifkan, dua langkah dalam rencana kueri paralel, seperti ditunjukkan pada kolom `Extra` dalam output `EXPLAIN`. Pemrosesan intensif I/O dan CPU intensif untuk langkah-langkah tersebut didorong ke lapisan penyimpanan.

```
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                    |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

```
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                      |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------
+
```

Untuk informasi tentang cara menafsirkan EXPLAIN output untuk query paralel dan bagian SQL pernyataan yang dapat diterapkan oleh query paralel, lihatSQLkonstruksi untuk kueri paralel di Aurora My SQL.

Contoh output berikut menunjukkan hasil dari menjalankan kueri sebelumnya pada instans db.r4.2xlarge dengan IT buffer pool dingin. Kueri berjalan jauh lebih cepat saat menggunakan kueri paralel.

> **Note**
>
> Karena waktu ditentukan oleh banyak faktor lingkungan, hasil Anda mungkin berbeda. Selalu lakukan uji performa Anda sendiri untuk mengonfirmasi temuan dengan lingkungan dan beban kerja Anda sendiri, dan sebagainya.

```
-- Without parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
```

```
|   92511430 | 514726.4896 | 1995-03-06  |             0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |             0 |
+------------+-------------+-------------+---------------+
10 rows in set (1 min 49.91 sec)
```

Banyak kueri sampel di seluruh bagian ini menggunakan tabel dari kumpulan data TPC -H ini, khususnya PART tabel, yang memiliki 20 juta baris dan definisi berikut.

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

Bereksperimenlah dengan beban kerja Anda untuk mengetahui apakah SQL pernyataan individual dapat memanfaatkan query paralel. Lalu gunakan teknik pemantauan berikut untuk membantu memverifikasi frekuensi penggunaan kueri paralel dalam beban kerja nyata dari waktu ke waktu. Untuk beban kerja nyata, faktor tambahan seperti batas keserentakan berlaku.

## Memantau query paralel untuk Aurora My SQL

Jika SQL klaster Aurora Saya menggunakan kueri paralel, Anda mungkin melihat peningkatan `VolumeReadIOPS` nilai. Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kuerinya cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.

Selain CloudWatch metrik Amazon yang dijelaskan dalamMelihat metrik di konsol Amazon RDS, Aurora menyediakan variabel status global lainnya. Anda dapat menggunakan variabel status global ini untuk membantu memantau eksekusi kueri paralel. Variabel tersebut dapat memberi Anda wawasan tentang alasan pengoptimal dapat menggunakan atau tidak menggunakan kueri paralel

dalam situasi tertentu. Untuk mengakses variabel ini, Anda dapat menggunakan perintah SHOW GLOBAL STATUS. Anda juga dapat menemukan variabel ini tercantum sebagai berikut.

Sesi query paralel belum tentu one-to-one pemetaan dengan query yang dilakukan oleh database. Contohnya, anggaplah bahwa rencana kueri Anda memiliki dua langkah yang menggunakan kueri paralel. Dalam hal ini, kueri tersebut mencakup dua sesi paralel dan penghitung untuk permintaan yang telah diupayakan dan permintaan yang berhasil ditambah sebanyak dua.

Saat Anda bereksperimen dengan kueri paralel dengan mengeluarkan pernyataan EXPLAIN, bersiaplah untuk melihat kenaikan di penghitung yang ditetapkan sebagai "tidak dipilih" meskipun kueri tersebut sebenarnya tidak berjalan. Saat bekerja dengan kueri paralel dalam produksi, Anda dapat memeriksa apakah penghitung yang "tidak dipilih" meningkat lebih cepat dari yang Anda perkirakan. Pada titik ini, Anda dapat menyesuaikan sehingga kueri paralel berjalan untuk kueri yang Anda harapkan. Untuk melakukannya, Anda dapat mengubah pengaturan klaster, campuran kueri, instans DB tempat kueri paralel diaktifkan, dan sebagainya.

Penghitung ini dilacak di tingkat instans DB. Saat terhubung ke titik akhir yang berbeda, Anda mungkin melihat metrik yang berbeda karena setiap instans DB menjalankan serangkaian kueri paralelnya sendiri. Anda mungkin juga melihat metrik yang berbeda saat titik akhir pembaca terhubung ke instans DB yang berbeda untuk setiap sesi.

| Nama | Penjelasan |
| --- | --- |
| Aurora_pq_bytes_returned | Jumlah byte untuk struktur data tuple yang ditransmisikan ke simpul head selama kueri paralel. Bagi dengan 16.384 untuk membandingkan dengan Aurora_pq_pages_pushed_down . |
| Aurora_pq_max_concurrent_requests | Jumlah maksimum sesi kueri paralel yang dapat berjalan secara konkuren di instans Aurora DB ini. Ini adalah nomor tetap yang tergantung pada kelas instans AWS DB. |
| Aurora_pq_pages_pushed_down | Jumlah halaman data (masing-masing dengan ukuran tetap 16 KiB) tempat kueri paralel menghindari transmisi jaringan ke simpul head. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_attempted` | Jumlah sesi kueri paralel yang diminta. Nilai ini mungkin mewakili lebih dari satu sesi per kueri, tergantung pada SQL konstruksi seperti subkueri dan gabungan. |
| `Aurora_pq_request_executed` | Jumlah sesi kueri paralel yang berhasil berjalan. |
| `Aurora_pq_request_failed` | Jumlah sesi kueri paralel yang menampilk an kesalahan kepada klien. Dalam beberapa kasus, permintaan untuk kueri paralel mungkin gagal, misalnya karena masalah dalam lapisan penyimpanan. Dalam kasus ini, bagian kueri yang gagal dicoba kembali menggunakan mekanisme kueri nonparalel. Jika kueri yang dicoba kembali juga gagal, kesalahan ditampilk an ke klien dan penghitung ini ditambah. |
| `Aurora_pq_request_in_progress` | Jumlah sesi kueri paralel yang sedang berlangsung. Angka ini berlaku untuk instans Aurora DB tertentu yang Anda hubungkan, bukan seluruh klaster Aurora DB. Untuk melihat apakah instans DB mendekati batas konkurens inya, bandingkan nilai ini dengan `Aurora_pq _max_concurrent_requests` . |
| `Aurora_pq_request_not_chosen` | Berapa kali kueri paralel tidak dipilih untuk memenuhi suatu kueri. Nilai ini adalah jumlah dari beberapa penghitung granular lainnya. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-ben ar dilakukan. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_below_min_rows` | Berapa kali kueri paralel tidak dipilih karena jumlah baris dalam tabel. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |
| `Aurora_pq_request_not_chosen_column_bit` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena jenis data yang tidak didukung dalam daftar kolom yang diproyeksikan. |
| `Aurora_pq_request_not_chosen_column_geometry` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `GEOMETRY`. Untuk informasi tentang Aurora SQL versi Saya yang menghapus batasan ini, lihat. Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL |
| `Aurora_pq_request_not_chosen_column_lob` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `LOB`, atau kolom `VARCHAR` yang disimpan secara eksternal karena panjang yang dinyatakan. Untuk informasi tentang Aurora SQL versi Saya yang menghapus batasan ini, lihat. Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL |
| `Aurora_pq_request_not_chosen_column_virtual` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel berisi kolom virtual. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_custom_charset` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan set karakter kustom. |
| `Aurora_pq_request_not_chosen_fast_ddl` | Jumlah permintaan query paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel saat ini sedang diubah oleh pernyataan cepat DDL`ALTER`. |
| `Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool` | Berapa kali kueri paralel tidak dipilih, meskipun kurang dari 95 persen data tabel berada di dalam pool buffer, karena data tabel yang tidak di-buffer tidak cukup untuk membuat kueri paralel layak dijalankan. |
| `Aurora_pq_request_not_chosen_full_text_index` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki indeks teks penuh. |
| `Aurora_pq_request_not_chosen_high_buffer_pool_pct` | Berapa kali kueri paralel tidak dipilih karena persentase tinggi data tabel (saat ini, lebih dari 95 persen) sudah berada di dalam pool buffer. Dalam kasus ini, pengoptimal menentukan bahwa membaca data dari pool buffer akan lebih efisien. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |
| `Aurora_pq_request_not_chosen_index_hint` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mencakup petunjuk indeks. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_innodb_table_format` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel menggunakan format baris InnoDB yang tidak didukung. Kueri paralel Aurora hanya berlaku untuk format baris `COMPACT`, `REDUNDANT` , dan `DYNAMIC`. |
| `Aurora_pq_request_not_chosen_long_trx` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri dimulai di dalam transaksi yang berjalan lama. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |
| `Aurora_pq_request_not_chosen_no_where_clause` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri tidak mencakup klausa `WHERE` apa pun. |
| `Aurora_pq_request_not_chosen_range_scan` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan pemindaian rentang pada indeks. |
| `Aurora_pq_request_not_chosen_row_length_too_long` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena total panjang gabungan semua kolom terlalu panjang. |
| `Aurora_pq_request_not_chosen_small_table` | Berapa kali kueri paralel tidak dipilih karena ukuran keseluruhan tabel, sebagaimana ditentukan oleh jumlah baris dan rata-rata panjang baris. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_temporary_table` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mengacu pada tabel sementara yang menggunakan jenis tabel `MyISAM` atau `memory` yang tidak didukung. |
| `Aurora_pq_request_not_chosen_tx_isolation` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan tingkat isolasi transaksi yang tidak didukung. Pada instans pembaca DB, kueri paralel hanya berlaku untuk tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED` . |
| `Aurora_pq_request_not_chosen_update_delete_stmts` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri adalah bagian dari pernyataan `UPDATE` atau `DELETE`. |
| `Aurora_pq_request_not_chosen_unsupported_access` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klausa `WHERE` tidak memenuhi kriteria untuk kueri paralel. Hasil ini dapat muncul jika kueri tidak memerlukan pemindaian sarat data, atau jika kueri adalah pernyataan `DELETE` atau `UPDATE`. |
| `Aurora_pq_request_not_chosen_unsupported_storage_type` | Jumlah permintaan query paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klaster Aurora SQL My DB tidak menggunakan konfigurasi penyimpan an klaster Aurora yang didukung. Parameter ini tersedia di Aurora My SQL versi 3.04 dan lebih tinggi. Untuk informasi selengkapnya, lihat Pembatasan. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_throttled` | Berapa kali kueri paralel tidak dipilih karena jumlah maksimum kueri paralel konkuren sudah berjalan pada instans Aurora DB tertentu. |

## SQLkonstruksi untuk kueri paralel di Aurora My SQL

Di bagian berikut, Anda dapat menemukan detail lebih lanjut tentang mengapa SQL pernyataan tertentu menggunakan atau tidak menggunakan query paralel. Bagian ini juga merinci bagaimana SQL fitur Aurora My berinteraksi dengan query paralel. Informasi ini dapat membantu Anda mendiagnosis masalah performa untuk klaster yang menggunakan kueri paralel atau memahami cara kueri paralel berlaku untuk beban kerja tertentu Anda.

Keputusan untuk menggunakan kueri paralel bergantung pada banyak faktor yang terjadi pada saat pernyataan itu berjalan. Dengan demikian, kueri paralel dapat digunakan untuk kueri tertentu selalu, tidak pernah, atau hanya dalam kondisi tertentu.

> **Tip**
>
> Ketika Anda melihat contoh-contoh ini diHTML, Anda dapat menggunakan widget Salin di sudut kanan atas setiap daftar kode untuk menyalin kode untuk mencoba SQL sendiri. Dengan menggunakan widget Salin, Anda tidak perlu menyalin karakter tambahan di sekitar prompt `mysql>` dan baris lanjutan `->`.

Topik

- EXPLAINpernyataan
- Klausul WHERE
- Bahasa definisi data (DDL)
- Jenis data kolom
- Tabel yang dipartisi
- Fungsi agregat, klausa GROUP BY, dan klausa HAVING
- Panggilan fungsi dalam WHERE klausa
- Klausul LIMIT

- Operator perbandingan
- Gabungan
- Subkueri
- UNION
- Tampilan
- Pernyataan bahasa manipulasi data (DML)
- Transaksi dan penguncian
- Indeks pohon B
- Indeks pencarian teks lengkap (FTS)
- Kolom virtual
- Mekanisme caching default
- Petunjuk pengoptimal
- Meja ISAM sementara saya

### EXPLAINpernyataan

Seperti yang ditunjukkan dalam contoh di seluruh bagian ini, pernyataan `EXPLAIN` menunjukkan apakah setiap tahap kueri saat ini memenuhi syarat untuk kueri paralel. Pernyataan itu juga menunjukkan aspek mana dari kueri yang dapat diturunkan ke lapisan penyimpanan. Berikut ini adalah item terpenting dalam rencana kueri:

- Nilai selain `NULL` untuk kolom `key` menunjukkan bahwa kueri dapat dilakukan secara efisien menggunakan pencarian indeks, dan kueri paralel tidak memungkinkan.
- Nilai kecil untuk kolom `rows` (nilai bukan dalam jutaan) menunjukkan bahwa kueri tersebut tidak mengakses cukup data untuk membuat kueri paralel yang bermanfaat. Ini berarti kueri paralel tidak memungkinkan.
- Kolom `Extra` menunjukkan jika kueri paralel yang diharapkan akan digunakan. Output ini terlihat seperti contoh berikut.

  ```
  Using parallel query (A columns, B filters, C exprs; D extra)
  ```

  Angka `columns` menunjukkan jumlah kolom yang dimaksud dalam blok kueri.

Angka `filters` menunjukkan jumlah predikat `WHERE` yang menunjukkan perbandingan sederhana antara nilai kolom dan konstanta. Perbandingannya dapat berupa kesetaraan, ketidaksetaraan, atau rentang. Aurora dapat memaralelkan jenis predikat ini dengan sangat efektif.

Angka `exprs` menunjukkan jumlah ekspresi seperti panggilan fungsi, operator, atau ekspresi lainnya yang juga dapat diparalelkan, meskipun tidak sama efektifnya dengan ketentuan filter.

Angka `extra` menunjukkan jumlah ekspresi yang tidak dapat diturunkan dan dilakukan oleh simpul kepala.

Contohnya, pertimbangkan output `EXPLAIN` berikut ini.

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
```

Informasi dari kolom `Extra` menunjukkan bahwa lima kolom diekstraksi dari setiap baris untuk mengevaluasi ketentuan kueri dan menyusun set hasil. Satu predikat `WHERE` mencakup sebuah filter, yaitu, kolom yang langsung diuji dalam klausa `WHERE`. Dua klausa `WHERE` memerlukan evaluasi ekspresi yang lebih rumit, dalam hal ini mencakup panggilan fungsi. Bidang `0 extra` mengonfirmasi bahwa semua operasi dalam klausa `WHERE` diturunkan ke lapisan penyimpanan sebagai bagian dari pemrosesan kueri paralel.

Dalam kasus di mana kueri paralel tidak dipilih, Anda biasanya dapat menyimpulkan alasan dari output kolom `EXPLAIN` lain. Contohnya, nilai `rows` mungkin terlalu kecil, atau kolom `possible_keys` mungkin menunjukkan bahwa kueri dapat menggunakan pencarian indeks alih-alih pemindaian sarat data. Contoh berikut menunjukkan sebuah kueri di mana pengoptimal dapat

memperkirakan bahwa kueri tersebut hanya akan memindai sejumlah kecil baris. Hal itu dilakukan berdasarkan karakteristik kunci primer. Dalam hal ini, kueri paralel tidak diperlukan.

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
+----+-------------+-------+------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+------+---------------+---------+---------+------+------
+--------------------------+
```

Output yang menunjukkan apakah kueri paralel akan digunakan mempertimbangkan semua faktor yang tersedia saat pernyataan `EXPLAIN` dijalankan. Pengoptimal mungkin mengambil pilihan lain saat kueri benar-benar dijalankan, jika situasi berubah pada saat itu. Contohnya, `EXPLAIN` dapat melaporkan bahwa suatu pernyataan akan menggunakan kueri paralel. Namun saat kueri benar-benar dijalankan nantinya, kueri tersebut mungkin tidak menggunakan kueri paralel berdasarkan kondisi pada saat itu. Kondisi tersebut dapat mencakup beberapa kueri paralel lainnya yang berjalan secara serentak. Kondisi lainnya juga dapat mencakup baris yang dihapus dari tabel, indeks baru yang dibuat, terlalu banyak waktu yang terlewati dalam transaksi terbuka, dan sebagainya.

## Klausul WHERE

Agar menggunakan pengoptimalan kueri paralel, suatu kueri harus mencakup klausa `WHERE`.

Pengoptimalan kueri paralel mempercepat berbagai jenis ekspresi yang digunakan dalam klausa `WHERE`:

- Perbandingan sederhana antara nilai kolom dengan konstanta, yang disebut filter. Perbandingan ini sangat diuntungkan ketika diturunkan ke lapisan penyimpanan. Jumlah ekspresi filter dalam sebuah kueri dilaporkan dalam output `EXPLAIN`.
- Jenis ekspresi lain dalam klausa `WHERE` juga diturunkan ke lapisan penyimpanan jika memungkinkan. Jumlah ekspresi tersebut dalam sebuah kueri dilaporkan dalam output `EXPLAIN`. Ekspresi ini dapat berupa panggilan fungsi, operator `LIKE`, ekspresi `CASE`, dan sebagainya.
- Fungsi dan operator tertentu saat ini tidak diturunkan oleh kueri paralel. Jumlah ekspresi tersebut dalam sebuah kueri dilaporkan sebagai penghitung `extra` dalam output `EXPLAIN`. Sisa kueri tersebut masih dapat menggunakan kueri paralel.

- Meskipun ekspresi dalam daftar pilihan tidak diturunkan, kueri yang berisi fungsi tersebut masih dapat diuntungkan dari pengurangan lalu lintas jaringan untuk hasil perantara dari kueri paralel. Contohnya, kueri yang memanggil fungsi agregat dalam daftar pilihan dapat diuntungkan dari kueri paralel, meskipun fungsi agregasi tidak diturunkan.

Contohnya, kueri berikut ini melakukan pemindaian tabel lengkap dan memproses semua nilai untuk kolom `P_BRAND`. Akan tetapi, kueri tersebut tidak menggunakan kueri paralel karena tidak mencakup klausa `WHERE` apa pun.

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+----------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                            |
+----+-------------+-------+------+---------------+------+---------+------+----------
+----------------------------------+
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+----------------------------------+
```

Sebaliknya, kueri berikut mencakup predikat `WHERE` yang memfilter hasil, sehingga kueri paralel dapat diterapkan:

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+-------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                          |
+----+...+----------
+-------------------------------------------------------------------------------------------
+
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------------------
+
```

Jika pengoptimal memperkirakan bahwa baris yang dikembalikan untuk suatu blok kueri jumlahnya kecil, kueri paralel tidak digunakan untuk blok kueri tersebut. Contoh berikut menunjukkan kasus di mana operator lebih-besar-daripada di kolom kunci primer berlaku untuk jutaan baris, yang menyebabkan kueri paralel digunakan. Pengujian kurang-dari yang berkebalikan diperkirakan berlaku hanya pada beberapa baris dan tidak menggunakan kueri paralel.

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+--------------------------+
| id |...| rows | Extra                    |
+----+...+------+--------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+--------------------------+
```

## Bahasa definisi data (DDL)

Di Aurora My SQL versi 2, query paralel hanya tersedia untuk tabel yang tidak ada operasi bahasa definisi data cepat (DDL) yang tertunda. Di Aurora My SQL versi 3, Anda dapat menggunakan query paralel pada tabel pada saat yang sama sebagai operasi instanDDL.

Instan DDL di Aurora SQL Versi saya 3 menggantikan fitur cepat DDL di Aurora My versi 2. SQL Untuk informasi tentang instanDDL, lihatDDL instan (Aurora MySQL versi 3).

## Jenis data kolom

Di Aurora My SQL versi 3, query paralel dapat bekerja dengan tabel yang berisi kolom dengan tipe data`TEXT`,, `BLOB` `JSON`, dan. `GEOMETRY` Ini juga dapat bekerja dengan kolom `VARCHAR` dan `CHAR` dengan panjang maksimum yang dinyatakan melebihi 768 byte. Jika kueri Anda mengacu pada kolom yang berisi jenis objek besar seperti itu, tugas tambahan untuk mengambilnya akan memunculkan overhead ke dalam pemrosesan kueri. Dalam hal ini, periksa apakah kueri

dapat menghilangkan referensi ke kolom-kolom tersebut. Jika tidak, jalankan tolok ukur untuk mengonfirmasi apakah kueri tersebut lebih cepat dengan kueri paralel diaktifkan atau dinonaktifkan.

Di Aurora My SQL versi 2, query paralel memiliki batasan ini untuk tipe objek besar:

- Jenis data `TEXT`, `BLOB`, `JSON`, dan `GEOMETRY` tidak didukung dengan kueri paralel. Kueri yang mengacu pada kolom apa pun dari jenis ini tidak dapat menggunakan kueri paralel.
- Kolom dengan panjang variabel (jenis data `VARCHAR` dan `CHAR`) kompatibel dengan kueri paralel hingga panjang maksimum yang dinyatakan 768 byte. Kueri yang mengacu pada kolom apa pun dari jenis yang dinyatakan dengan panjang maksimum yang lebih panjang tidak dapat menggunakan kueri paralel. Untuk kolom yang menggunakan set karakter multibyte, batas byte mempertimbangkan jumlah maksimum byte dalam set karakter. Contohnya, untuk set karakter `utf8mb4` (yang memiliki panjang karakter maksimum 4 byte), kolom `VARCHAR(192)` kompatibel dengan kueri paralel tetapi kolom `VARCHAR(193)` tidak.

## Tabel yang dipartisi

Anda dapat menggunakan tabel yang dipartisi dengan query paralel di Aurora My versi 3. SQL Karena tabel yang dipartisi direpresentasikan secara internal sebagai beberapa tabel yang lebih kecil, kueri yang menggunakan kueri paralel pada tabel yang tidak dipartisi mungkin tidak menggunakan kueri paralel pada tabel yang dipartisi identik. Aurora My SQL mempertimbangkan apakah setiap partisi cukup besar untuk memenuhi syarat untuk optimasi query paralel, alih-alih mengevaluasi ukuran seluruh tabel. Periksa apakah variabel status `Aurora_pq_request_not_chosen_small_table` bertambah jika kueri pada tabel yang dipartisi tidak menggunakan kueri paralel padahal Anda mengharapkannya.

Misalnya, pertimbangkan satu tabel yang dipartisi dengan `PARTITION BY HASH (`*`column`*`) PARTITIONS 2` dan tabel lain dipartisi dengan `PARTITION BY HASH (`*`column`*`) PARTITIONS 10`. Dalam tabel dengan dua partisi, partisinya lima kali lebih besar daripada tabel dengan sepuluh partisi. Dengan demikian, kueri paralel lebih memungkinkan untuk digunakan untuk mengueri tabel dengan lebih sedikit partisi. Dalam contoh berikut, tabel `PART_BIG_PARTITIONS` memiliki dua partisi dan `PART_SMALL_PARTITIONS` memiliki sepuluh partisi. Dengan data yang identik, kueri paralel lebih memungkinkan untuk digunakan untuk tabel dengan lebih sedikit partisi besar.

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
```

```
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+---------------------+-----------
+-------------------------------------------------------------------------------------------
+
| id | select_type | table               | partitions | Extra
                                                                                          |
+----+-------------+---------------------+-----------
+-------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
+----+-------------+---------------------+-----------
+-------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+----------------------+------------------------------
+------------------------------+
| id | select_type | table                | partitions                   | Extra
                  |
+----+-------------+----------------------+------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+----------------------+------------------------------
+------------------------------+
```

## Fungsi agregat, klausa GROUP BY, dan klausa HAVING

Kueri yang mencakup fungsi agregat sering kali merupakan kandidat yang baik untuk kueri paralel, karena kueri tersebut memindai baris dalam jumlah besar dalam tabel besar.

Di Aurora My SQL 3, query paralel dapat mengoptimalkan panggilan fungsi agregat dalam daftar pilih dan klausa. `HAVING`

Sebelum Aurora My SQL 3, panggilan fungsi agregat dalam daftar pilih atau `HAVING` klausa tidak ditekan ke lapisan penyimpanan. Akan tetapi, kueri paralel tetap dapat meningkatkan performa kueri dengan fungsi agregat tersebut. Hal itu dilakukan pertama-tama dengan mengekstraksi nilai kolom dari halaman data mentah secara paralel pada lapisan penyimpanan. Kueri paralel tersebut

kemudian mengirim kembali nilai itu ke simpul kepala dalam format urutan yang padat, bukan sebagai keseluruhan halaman data. Seperti biasa, kueri memerlukan setidaknya satu predikat `WHERE` agar kueri paralel dapat diaktifkan.

Contoh sederhana berikut mengilustrasikan jenis kueri agregat yang dapat diuntungkan dari kueri paralel. Hal tersebut dilakukan dengan mengembalikan hasil menengah dalam bentuk ringkas ke simpul kepala, memfilter baris yang tidak cocok dari hasil menengah, atau keduanya.

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+-------------------------------------------------------------------------+
| id |...| Extra                                                                   |
+----+...+-------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+-------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

## Panggilan fungsi dalam WHERE klausa

Aurora dapat menerapkan pengoptimalan kueri paralel pada panggilan ke sebagian besar fungsi default dalam klausa `WHERE`. Memparalelkan panggilan fungsi ini menurunkan beberapa CPU pekerjaan dari node kepala. Mengevaluasi fungsi predikat secara paralel selama tahap kueri paling awal akan membantu Aurora meminimalkan jumlah data yang dikirim dan diproses selama tahap berikutnya.

Saat ini, paralelisasi tidak berlaku untuk panggilan fungsi dalam daftar pilihan. Fungsi-fungsi tersebut dievaluasi oleh simpul kepala, bahkan jika panggilan fungsi yang identik muncul dalam

klausa WHERE. Nilai asli dari kolom yang relevan disertakan dalam urutan yang dikirim dari simpul penyimpanan kembali ke simpul kepala. Simpul kepala melakukan transformasi apa pun seperti UPPER, CONCATENATE, dan seterusnya untuk menghasilkan nilai akhir untuk set hasil.

Dalam contoh berikut, kueri paralel memaralelkan panggilan dengan LOWER karena muncul di klausa WHERE. Kueri paralel tidak memengaruhi panggilan ke SUBSTR dan UPPER karena muncul dalam daftar pilihan.

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

Pertimbangan yang sama berlaku untuk ekspresi lain, seperti ekspresi CASE atau operator LIKE. Misalnya, contoh berikut menunjukkan bahwa kueri paralel mengevaluasi ekspresi CASE dan operator LIKE di klausa WHERE.

```
mysql> explain select p_mfgr, p_retailprice from part
    -> where p_retailprice > case p_mfgr
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
    -> group by p_retailprice;
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
```

```
+----+...
+-----------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------
+
```

## Klausul LIMIT

Saat ini, kueri paralel tidak digunakan untuk blok kueri yang mencakup klausa `LIMIT`. Kueri paralel masih dapat digunakan untuk fase kueri sebelumnya dengan `GROUP by`, `ORDER BY`, atau gabungan.

## Operator perbandingan

Pengoptimal memperkirakan jumlah baris yang harus dipindai untuk mengevaluasi operator perbandingan, dan menentukan apakah akan menggunakan kueri paralel berdasarkan estimasi tersebut.

Contoh pertama berikut ini menunjukkan bahwa perbandingan setara terhadap kolom kunci primer dapat dilakukan secara efisien tanpa kueri paralel. Contoh kedua berikut menunjukkan bahwa perbandingan yang serupa terhadap kolom yang tidak diindeks memerlukan pemindaian jutaan baris, sehingga dapat diuntungkan dari kueri paralel.

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
```

```
+----+...+----------
+-------------------------------------------------------------------------+
```

Pertimbangan yang sama berlaku untuk pengujian tidak sama dan untuk perbandingan rentang seperti kurang dari, lebih besar dari, atau sama dengan, atau `BETWEEN`. Pengoptimal memperkirakan jumlah baris yang akan dipindai, dan menentukan apakah kueri paralel bermanfaat berdasarkan volume keseluruhan I/O.

### Gabungan

Kueri gabungan dengan tabel besar biasanya mencakup operasi sarat data yang diuntungkan dari pengoptimalan kueri paralel. Perbandingan nilai kolom di antara beberapa tabel (yaitu, predikat gabungan itu sendiri) saat ini tidak diparalelkan. Namun, kueri paralel dapat menurunkan beberapa pemrosesan internal untuk fase penggabungan lainnya, seperti membangun filter Bloom selama hash join. Kueri paralel dapat diterapkan pada kueri gabungan tanpa klausa `WHERE`. Oleh karena itu, kueri gabungan adalah pengecualian untuk aturan bahwa klausa `WHERE` diperlukan untuk menggunakan kueri paralel.

Setiap fase pemrosesan gabungan dievaluasi untuk memeriksa apakah fase tersebut memenuhi syarat untuk kueri paralel. Jika lebih dari satu fase dapat menggunakan kueri paralel, fase-fase ini dilakukan secara berurutan. Dengan demikian, setiap kueri gabungan dihitung sebagai satu sesi kueri paralel dalam hal batas keserentakan.

Contohnya, saat kueri gabungan mencakup predikat `WHERE` untuk memfilter baris dari salah satu tabel yang digabungkan, opsi pemfilteran tersebut dapat menggunakan kueri paralel. Sebagai contoh lain, anggaplah kueri gabungan menggunakan mekanisme hash join, contohnya untuk menggabungkan tabel besar dengan tabel kecil. Dalam hal ini, pemindaian tabel untuk menghasilkan struktur data filter Bloom mungkin dapat menggunakan kueri paralel.

> **Note**
>
> Kueri paralel biasanya digunakan untuk jenis kueri sarat sumber daya yang diuntungkan dari pengoptimalan hash join. Metode untuk mengaktifkan optimasi bergabung hash tergantung pada versi Aurora MySQL. Untuk detail untuk setiap versi, lihat Mengaktifkan hash join untuk klaster kueri paralel. Untuk informasi tentang cara menggunakan hash join secara efektif, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
```

```
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------
+
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

              |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

              |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------
+
```

Untuk kueri gabungan yang menggunakan mekanisme nested loop, blok nested loop terluar mungkin menggunakan kueri paralel. Penggunaan kueri paralel bergantung pada faktor yang sama seperti biasanya, seperti adanya ketentuan filter tambahan dalam klausa `WHERE`.

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                  |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                       |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
```

## Subkueri

Blok kueri luar dan blok subkueri dalam masing-masing dapat menggunakan kueri paralel, atau tidak. Untuk setiap blok, menggunakan kueri paralel atau tidak, didasarkan pada karakteristik umum tabel, klausa `WHERE`, dan sebagainya. Contohnya, kueri berikut menggunakan kueri paralel untuk blok subkueri tetapi tidak untuk blok luar.

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type |...| rows     | Extra
                       |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                       |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

Saat ini, subkueri yang berkorelasi tidak dapat menggunakan pengoptimalan kueri paralel.

## UNION

Setiap blok kueri dalam kueri `UNION` dapat menggunakan kueri paralel atau tidak, berdasarkan karakteristik umum tabel, klausul `WHERE`, dan sebagainya, untuk setiap bagian `UNION`.

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+----------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type    |...| rows     | Extra
                          |
+----+----------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
```

```
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                        |
+----+--------------+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> Setiap klausa `UNION` dalam kueri dijalankan secara berurutan. Bahkan jika kueri mencakup beberapa tahap yang semuanya menggunakan kueri paralel, kueri tersebut hanya menjalankan satu kueri paralel dalam satu waktu. Oleh karena itu, bahkan kueri multitahap yang kompleks hanya dihitung sebagai 1 terhadap batas kueri paralel yang serentak.

## Tampilan

Pengoptimal menulis ulang kueri apa pun menggunakan tampilan sebagai kueri yang lebih panjang menggunakan tabel yang mendasarinya. Dengan demikian, kueri paralel bekerja dengan cara yang sama baik referensi tabelnya berupa tampilan atau tabel nyata. Semua pertimbangan yang sama tentang apakah akan menggunakan kueri paralel untuk suatu kueri, dan bagian mana yang diturunkan, berlaku untuk kueri terakhir yang ditulis ulang.

Contohnya, rencana kueri berikut menunjukkan definisi tampilan yang biasanya tidak menggunakan kueri paralel. Ketika tampilan ditanyakan dengan `WHERE` klausa tambahan, Aurora My SQL menggunakan query paralel.

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

## Pernyataan bahasa manipulasi data (DML)

Pernyataan `INSERT` dapat menggunakan kueri paralel untuk fase pemrosesan `SELECT`, jika bagian `SELECT` sesuai dengan ketentuan lainnya untuk kueri paralel.

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> Biasanya, setelah pernyataan `INSERT`, data untuk baris yang baru dimasukkan ada di dalam buffer pool. Oleh karena itu, suatu tabel mungkin tidak memenuhi syarat untuk kueri paralel segera setelah memasukkan baris dalam jumlah besar. Lalu, setelah data dikosongkan dari buffer pool selama operasi normal, kueri terhadap tabel tersebut dapat mulai menggunakan kueri paralel.

Pernyataan `CREATE TABLE AS SELECT` tidak menggunakan kueri paralel, meskipun bagian `SELECT` dari pernyataan tersebut akan memenuhi syarat untuk kueri paralel. DDLAspek pernyataan ini membuatnya tidak kompatibel dengan pemrosesan query paralel. Sebaliknya, dalam pernyataan `INSERT ... SELECT`, bagian `SELECT` dapat menggunakan kueri paralel.

Kueri paralel tidak pernah digunakan untuk pernyataan `DELETE` atau `UPDATE`, terlepas dari ukuran tabel dan predikat dalam klausa `WHERE`.

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
```

```
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

## Transaksi dan penguncian

Anda dapat menggunakan semua tingkat isolasi pada instans utama Aurora.

Pada instans pembaca Aurora DB, kueri paralel berlaku untuk pernyataan yang dilakukan dalam tingkat isolasi `REPEATABLE READ`. Aurora SQL Versi saya 2.09 atau lebih tinggi juga dapat menggunakan tingkat `READ COMMITTED` isolasi pada instans DB pembaca. `REPEATABLE READ`adalah tingkat isolasi default untuk instans DB pembaca Aurora. Untuk menggunakan tingkat isolasi `READ COMMITTED` pada instans reader DB membutuhkan pengaturan opsi konfigurasi `aurora_read_replica_read_committed` pada tingkat sesi. Tingkat `READ COMMITTED` isolasi untuk instance pembaca sesuai dengan perilaku SQL standar. Namun, isolasi ini lebih longgar pada instans pembaca dibandingkan ketika kueri menggunakan tingkat isolasi `READ COMMITTED` pada instans penulis.

Untuk informasi lebih lanjut tentang tingkat isolasi Aurora, terutama perbedaan dalam hal `READ COMMITTED` antara instans penulis dan pembaca, lihat Tingkat isolasi Aurora MySQL.

Setelah transaksi besar selesai, statistik tabel mungkin menjadi usang. Statistik yang usang tersebut mungkin memerlukan pernyataan `ANALYZE TABLE` sebelum Aurora dapat secara akurat memperkirakan jumlah baris. DMLPernyataan skala besar mungkin juga membawa sebagian besar data tabel ke dalam kumpulan buffer. Memiliki data ini di dalam buffer pool dapat menyebabkan kueri paralel menjadi lebih jarang dipilih untuk tabel tersebut sampai data dikosongkan dari pool.

Jika sesi Anda berada dalam transaksi yang berlangsung lama (secara default, 10 menit), kueri lebih lanjut di dalam sesi tersebut tidak menggunakan kueri paralel. Waktu habis juga dapat terjadi dalam satu kueri yang berlangsung lama. Jenis waktu habis ini dapat terjadi jika kueri berjalan lebih lama dari interval maksimum (saat ini 10 menit) sebelum pemrosesan kueri paralel dimulai.

Anda dapat mengurangi kemungkinan memulai transaksi yang berlangsung lama secara tidak sengaja dengan mengatur `autocommit=1` dalam sesi `mysql` di mana Anda melakukan kueri ad hoc (satu waktu). Bahkan pernyataan `SELECT` terhadap suatu tabel akan memulai transaksi dengan membuat tampilan baca. Tampilan baca merupakan rangkaian data yang konsisten untuk kueri berikutnya yang tetap ada hingga transaksi dilakukan. Waspadai pembatasan ini juga saat menggunakan JDBC atau ODBC aplikasi dengan Aurora, karena aplikasi tersebut mungkin berjalan dengan pengaturan `autocommit` dimatikan.

Contoh berikut ini menunjukkan bagaimana, dengan pengaturan `autocommit` yang dinonaktifkan, menjalankan suatu kueri terhadap tabel akan menciptakan tampilan baca yang secara implisit memulai transaksi. Kueri yang dijalankan segera sesudahnya masih dapat menggunakan kueri paralel. Namun, setelah jeda beberapa menit, kueri tidak lagi memenuhi syarat untuk kueri paralel. Mengakhiri transaksi dengan `COMMIT` atau `ROLLBACK` akan memulihkan eligilibilitas kueri paralel.

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+

mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+

mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
```

```
+----+...+---------
+-------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------+
```

Untuk melihat berapa kali kueri tidak memenuhi syarat untuk kueri paralel karena kueri tersebut berada dalam transaksi yang berlangsung lama, periksa variabel status `Aurora_pq_request_not_chosen_long_trx`.

```
mysql> show global status like '%pq%trx%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+-------------------------------+-------+
```

Setiap pernyataan `SELECT` yang memperoleh kunci, seperti sintaks `SELECT FOR UPDATE` atau `SELECT LOCK IN SHARE MODE`, tidak dapat menggunakan kueri paralel.

Kueri paralel dapat berfungsi untuk tabel yang dikunci oleh pernyataan `LOCK TABLES`.

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-------------------------------------------------------------------------+
| id |...| rows      | Extra
          |
+----+...+-----------
+-------------------------------------------------------------------------+
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-------------------------------------------------------------------------+

mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
```

```
+----+...+-----------+-------------+
```

## Indeks pohon B

Statistik yang dikumpulkan oleh pernyataan `ANALYZE TABLE` membantu pengoptimal untuk memutuskan waktu untuk menggunakan kueri paralel atau pencarian indeks, berdasarkan karakteristik data untuk setiap kolom. Pertahankan statistik terkini dengan menjalankan `ANALYZE TABLE` setelah DML operasi yang membuat perubahan besar pada data dalam tabel.

Jika pencarian indeks dapat melakukan kueri secara efisien tanpa pemindaian sarat data, Aurora mungkin menggunakan pencarian indeks. Melakukan hal ini akan menghindari pengeluaran tambahan dari pemrosesan kueri paralel. Terdapat juga batas keserentakan pada jumlah kueri paralel yang dapat berjalan secara bersamaan di klaster Aurora DB mana pun. Pastikan Anda menggunakan praktik terbaik untuk mengindeks tabel, sehingga kueri yang paling sering dan paling sering muncul bersamaan menggunakan pencarian indeks.

## Indeks pencarian teks lengkap (FTS)

Saat ini, kueri paralel tidak digunakan untuk tabel yang berisi indeks pencarian teks penuh, terlepas dari apakah kueri tersebut mengacu pada kolom yang diindeks tersebut atau menggunakan operator `MATCH`.

## Kolom virtual

Saat ini, kueri paralel tidak digunakan untuk tabel yang berisi kolom virtual, terlepas dari apakah kueri tersebut merujuk pada kolom virtual mana pun.

## Mekanisme caching default

Aurora mencakup mekanisme caching default, yaitu buffer pool dan cache kueri. Pengoptimal Aurora memilih antara mekanisme caching ini dan kueri paralel bergantung pada mana yang paling efektif untuk kueri tertentu.

Saat kueri paralel memfilter baris dan mengubah serta mengekstraksi nilai kolom, data dikirim kembali ke simpul kepala sebagai urutan dan bukan sebagai halaman data. Oleh karena itu, menjalankan kueri paralel tidak akan menambahkan halaman apa pun ke dalam buffer pool, atau mengosongkan halaman yang sudah ada di dalam buffer pool.

Aurora memeriksa jumlah halaman data tabel yang ada dalam buffer pool, dan bagian mana dari data tabel tersebut yang direpresentasikan oleh angka tersebut. Aurora menggunakan informasi

tersebut untuk menentukan apakah lebih efisien untuk menggunakan kueri paralel (dan mem-bypass data dalam buffer pool). Sebagai alternatif, Aurora mungkin menggunakan jalur pemrosesan kueri nonparalel, yang menggunakan cache data dalam buffer pool. Halaman mana yang dibuat cache dan bagaimana kueri sarat data dapat memengaruhi caching dan pengosongan bergantung pada pengaturan konfigurasi yang terkait dengan buffer pool. Oleh karena itu, akan sulit untuk memprediksi apakah kueri tertentu menggunakan kueri paralel, karena pilihannya bergantung pada data yang selalu berubah dalam buffer pool.

Selain itu, Aurora menerapkan batas keserentakan pada kueri paralel. Karena tidak setiap kueri menggunakan kueri paralel, tabel yang diakses oleh beberapa kueri secara serentak biasanya memiliki bagian yang substansial dari datanya dalam buffer pool. Oleh karena itu, Aurora sering kali tidak memilih tabel ini untuk kueri paralel.

Ketika Anda menjalankan urutan kueri nonparalel pada tabel yang sama, kueri pertama mungkin lambat karena data tidak ada dalam buffer pool. Lalu kueri kedua dan selanjutnya jauh lebih cepat karena buffer pool sekarang sudah "dipanaskan". Kueri paralel biasanya menunjukkan performa yang konsisten dari kueri pertama terhadap tabel. Saat melakukan uji performa, buat tolok ukur kueri nonparalel dengan buffer pool yang dingin dan hangat. Dalam beberapa kasus, hasil dari buffer pool hangat dapat menghasilkan perbandingan yang baik dengan waktu kueri paralel. Dalam kasus ini, pertimbangkan faktor seperti frekuensi kueri terhadap tabel tersebut. Pertimbangkan juga apakah bermanfaat untuk menyimpan data untuk tabel tersebut di dalam buffer pool.

Cache kueri menghindari menjalankan ulang kueri saat kueri yang identik dikirimkan dan data tabel yang mendasarinya tidak berubah. Kueri yang dioptimalkan dengan fitur kueri paralel dapat masuk ke cache kueri, yang secara efektif menjadikannya instan saat dijalankan lagi.

> **Note**
>
> Saat melakukan perbandingan performa, cache kueri dapat menghasilkan jumlah pengaturan waktu yang rendah secara buatan. Oleh karena itu, dalam situasi serupa tolok ukur, Anda dapat menggunakan petunjuk `sql_no_cache`. Petunjuk ini mencegah hasil tersebut tersaji dari cache kueri, meskipun kueri yang sama telah dijalankan sebelumnya. Petunjuk segera muncul setelah pernyataan `SELECT` dalam kueri. Banyak contoh kueri paralel dalam topik ini mencakup petunjuk ini, untuk menjadikan waktu kueri sebanding antara versi kueri dengan kueri paralel yang diaktifkan dan dinonaktifkan.
> Pastikan Anda menghapus petunjuk ini dari sumber Anda saat beralih ke penggunaan kueri paralel untuk produksi.

## Petunjuk pengoptimal

Cara lain untuk mengontrol pengoptimal adalah dengan menggunakan petunjuk pengoptimal, yang dapat ditentukan dalam pernyataan individual. Misalnya, Anda dapat mengaktifkan pengoptimalan untuk satu tabel dalam sebuah pernyataan, kemudian mematikan pengoptimalan untuk tabel yang berbeda. Untuk informasi selengkapnya tentang petunjuk ini, lihat Petunjuk Pengoptimal di Manual Referensi Saya SQL.

Anda dapat menggunakan SQL petunjuk dengan kueri Aurora SQL My untuk menyempurnakan kinerja. Anda juga dapat menggunakan petunjuk agar rencana eksekusi untuk kueri penting tidak berubah karena kondisi yang tidak dapat diprediksi.

Kami telah memperluas fitur SQL petunjuk untuk membantu Anda mengontrol pilihan pengoptimal untuk paket kueri Anda. Petunjuk ini berlaku untuk kueri yang menggunakan pengoptimalan kueri paralel. Untuk informasi selengkapnya, lihat Aurora Petunjuk saya SQL.

## Meja ISAM sementara saya

Pengoptimalan kueri paralel hanya berlaku untuk tabel InnoDB. Karena Aurora My SQL menggunakan My ISAM behind the scenes untuk tabel sementara, fase kueri internal yang melibatkan tabel temporer tidak pernah menggunakan query paralel. Fase kueri ini ditunjukkan oleh `Using temporary` dalam output `EXPLAIN`.

# Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL

Anda dapat menggunakan fitur Audit Lanjutan berkinerja tinggi di Amazon Aurora SQL My untuk mengaudit aktivitas database. Untuk melakukannya, Anda mengaktifkan pengumpulan log audit dengan mengatur beberapa parameter klaster DB. Ketika Audit Lanjutan diaktifkan, Anda dapat menggunakannya untuk mencatat log kombinasi peristiwa yang didukung.

Anda dapat melihat atau mengunduh log audit untuk meninjau informasi audit untuk instans DB satu per satu. Untuk melakukannya, Anda dapat menggunakan prosedur dalam Memantau log Amazon Aurora Amazon.

> **Tip**
>
> Untuk klaster DB Aurora yang berisi beberapa instans DB, Anda mungkin merasa lebih praktis untuk memeriksa log audit untuk semua instans di klaster. Untuk melakukannya, Anda dapat menggunakan CloudWatch Log. Anda dapat mengaktifkan setelan di tingkat klaster untuk mempublikasikan data log SQL audit Aurora My ke grup log. CloudWatch Kemudian Anda dapat melihat, memfilter, dan mencari log audit melalui CloudWatch antarmuka. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .

## Mengaktifkan Audit Lanjutan

Gunakan parameter yang dijelaskan di bagian ini untuk mengaktifkan dan mengonfigurasi Audit Lanjutan untuk klaster DB Anda.

Gunakan parameter `server_audit_logging` untuk mengaktifkan atau menonaktifkan Audit Lanjutan.

Gunakan parameter `server_audit_events` untuk menentukan peristiwa apa yang akan dicatat lognya.

Gunakan `server_audit_incl_users` dan `server_audit_excl_users` untuk menentukan pengguna yang akan diaudit. Secara default, semua pengguna diaudit. Untuk detail tentang cara kerja parameter ini ketika salah satu atau keduanya dibiarkan kosong, atau nama pengguna yang sama ditentukan di keduanya, lihat server_audit_incl_users dan server_audit_excl_users.

Konfigurasikan Audit Lanjutan dengan mengatur parameter ini di grup parameter yang digunakan oleh klaster DB Anda. Anda dapat menggunakan prosedur yang ditunjukkan Memodifikasi parameter dalam grup parameter DB di untuk memodifikasi parameter cluster DB menggunakan AWS Management Console. Anda dapat menggunakan modify-db-cluster-parameter-group AWS CLI perintah atau RDS API operasi M odifyDBCluster ParameterGroup Amazon untuk memodifikasi parameter cluster DB secara terprogram.

Memodifikasi parameter ini tidak memerlukan pengaktifan ulang klaster DB saat grup parameter sudah dikaitkan dengan klaster Anda. Saat Anda mengaitkan grup parameter dengan klaster untuk pertama kalinya, pengaktifan ulang klaster diperlukan.

Topik

- server_audit_logging
- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

### server_audit_logging

Mengaktifkan atau menonaktifkan Audit Lanjutan. Parameter ini default keOFF; atur ke ON untuk mengaktifkan Audit Lanjutan.

Tidak ada data audit yang muncul di log kecuali jika Anda juga menentukan satu atau beberapa jenis peristiwa yang akan diaudit menggunakan parameter `server_audit_events`.

Untuk mengonfirmasi bahwa data audit dicatat lognya untuk instans DB, periksa apakah beberapa file log untuk instans tersebut memiliki nama form `audit/audit.log.`*`other_identifying_information`*. Untuk melihat nama file log, ikuti prosedur dalam Melihat dan mencantumkan file log basis data.

### server_audit_events

Berisi daftar peristiwa yang dipisahkan koma yang akan dicatat lognya. Peristiwa harus ditentukan dalam huruf besar semua, dan tidak boleh ada spasi antara elemen daftar, misalnya: `CONNECT,QUERY_DDL`. Parameter ini diatur secara default ke string kosong.

Anda dapat mencatat log untuk kombinasi peristiwa berikut:

- CONNECT— Log koneksi yang berhasil dan gagal dan juga pemutusan. Peristiwa ini mencakup informasi pengguna.
- QUERY— Mencatat semua kueri dalam teks biasa, termasuk kueri yang gagal karena kesalahan sintaks atau izin.

> **Tip**
>
> Dengan jenis peristiwa ini diaktifkan, data audit akan mencakup informasi tentang informasi pemantauan berkelanjutan dan pemeriksaan kondisi yang dilakukan Aurora secara otomatis. Jika Anda hanya tertarik pada jenis operasi tertentu, Anda dapat menggunakan jenis peristiwa yang lebih spesifik. Anda juga dapat menggunakan CloudWatch antarmuka untuk mencari di log untuk peristiwa yang terkait dengan database, tabel, atau pengguna tertentu.

- QUERY_ DCL — Mirip dengan QUERY acara, tetapi hanya mengembalikan kueri bahasa kontrol data (DCL) (GRANT,REVOKE, dan sebagainya).
- QUERY_ DDL — Mirip dengan QUERY acara, tetapi hanya mengembalikan kueri bahasa definisi data (DDL) (CREATE,ALTER, dan sebagainya).
- QUERY_ DML — Mirip dengan QUERY acara, tetapi hanya mengembalikan kueri bahasa manipulasi data (DML) (INSERT,UPDATE, dan sebagainya, dan jugaSELECT).
- TABLE— Log tabel yang dipengaruhi oleh eksekusi kueri.

> **Note**
>
> Tidak ada filter di Aurora yang mengecualikan kueri tertentu dari log audit. Untuk mengecualikan `SELECT` kueri, Anda harus mengecualikan semua DML pernyataan. Jika pengguna tertentu melaporkan `SELECT` kueri internal ini di log audit, maka Anda dapat mengecualikan pengguna tersebut dengan menyetel parameter cluster DB server_audit_excl_users. Namun, jika pengguna tersebut juga digunakan dalam aktivitas lain dan tidak dapat dihilangkan, maka tidak ada opsi lain untuk mengecualikan `SELECT` kueri.

### server_audit_incl_users

Berisi daftar nama pengguna yang dipisahkan koma untuk pengguna yang aktivitasnya dicatat dalam log. Tidak boleh ada spasi antara elemen daftar, misalnya: `user_3,user_4`. Parameter ini diatur

secara default ke string kosong. Panjang maksimumnya adalah 1024 karakter. Nama pengguna yang ditentukan harus cocok dengan nilai yang sesuai di kolom `User` dalam tabel `mysql.user`. Untuk informasi selengkapnya tentang nama pengguna, lihat Nama Pengguna Akun dan Kata Sandi dalam SQL dokumentasi Saya.

Jika `server_audit_incl_users` dan `server_audit_excl_users` keduanya kosong (default), semua pengguna diaudit.

Jika Anda menambahkan pengguna ke `server_audit_incl_users` dan mengosongkan `server_audit_excl_users`, hanya pengguna tersebut yang diaudit.

Jika Anda menambahkan pengguna ke `server_audit_excl_users` dan mengosongkan `server_audit_incl_users`, semua pengguna diaudit, kecuali yang tercantum dalam `server_audit_excl_users`.

Jika Anda menambahkan pengguna yang sama ke `server_audit_excl_users` dan `server_audit_incl_users`, pengguna tersebut diaudit. Ketika pengguna yang sama tercantum di kedua pengaturan tersebut, `server_audit_incl_users` akan diberi prioritas yang lebih tinggi.

Peristiwa koneksi dan pemutusan koneksi tidak terpengaruh oleh variabel ini; peristiwa ini akan selalu dicatat lognya jika ditentukan. Pengguna dicatat lognya meskipun pengguna tersebut juga ditentukan dalam parameter `server_audit_excl_users` karena `server_audit_incl_users` memiliki prioritas yang lebih tinggi.

### server_audit_excl_users

Berisi daftar nama pengguna yang dipisahkan koma untuk pengguna yang aktivitasnya tidak dicatat dalam log. Tidak boleh ada spasi antara elemen daftar, misalnya: `rdsadmin,user_1,user_2`. Parameter ini diatur secara default ke string kosong. Panjang maksimumnya adalah 1024 karakter. Nama pengguna yang ditentukan harus cocok dengan nilai yang sesuai di kolom `User` dalam tabel `mysql.user`. Untuk informasi selengkapnya tentang nama pengguna, lihat Nama Pengguna Akun dan Kata Sandi dalam SQL dokumentasi Saya.

Jika `server_audit_incl_users` dan `server_audit_excl_users` keduanya kosong (default), semua pengguna diaudit.

Jika Anda menambahkan pengguna ke `server_audit_excl_users` dan mengosongkan `server_audit_incl_users`, maka hanya pengguna yang Anda cantumkan di `server_audit_excl_users` yang tidak diaudit, dan semua pengguna lainnya diaudit.

Jika Anda menambahkan pengguna yang sama ke `server_audit_excl_users` dan `server_audit_incl_users`, pengguna tersebut diaudit. Ketika pengguna yang sama tercantum di kedua pengaturan tersebut, `server_audit_incl_users` akan diberi prioritas yang lebih tinggi.

Peristiwa koneksi dan pemutusan koneksi tidak terpengaruh oleh variabel ini; peristiwa ini akan selalu dicatat lognya jika ditentukan. Pengguna dicatat lognya jika pengguna tersebut juga ditentukan dalam parameter `server_audit_incl_users` karena pengaturan tersebut memiliki prioritas yang lebih tinggi daripada `server_audit_excl_users`.

## Melihat log audit

Anda dapat melihat dan mengunduh log audit menggunakan konsol. Di halaman Basis data, pilih instans DB untuk menampilkan detailnya, kemudian gulir ke bagian Log. Log audit yang dihasilkan oleh fitur Audit Lanjutan memiliki nama form `audit/audit.log.`*`other_identifying_information`*.

Untuk mengunduh file log, pilih file tersebut di bagian Log lalu pilih Unduh.

Anda juga bisa mendapatkan daftar file log dengan menggunakan describe-db-log-files AWS CLI perintah. Anda dapat mengunduh isi file log dengan menggunakan download-db-log-file-porsi AWS CLI perintah. Untuk informasi selengkapnya, silakan lihat Melihat dan mencantumkan file log basis data dan Mengunduh file log basis data.

## Detail log audit

File log direpresentasikan sebagai file variabel (CSV) yang dipisahkan koma dalam format UTF -8. Kueri juga di-wrapping dengan tanda kutip tunggal (').

Log audit disimpan secara terpisah di penyimpanan lokal setiap instans Aurora My SQL DB. Setiap instance mendistribusikan penulisan di empat file log sekaligus. Ukuran maksimum file log adalah 100 MB. Ketika batas yang tidak dapat dikonfigurasi ini tercapai, Aurora memutar file dan menghasilkan file baru.

> **Tip**
>
> Entri file log tidak berurutan. Untuk mengurutkan entri, gunakan nilai stempel waktu. Untuk melihat peristiwa terbaru, Anda mungkin harus meninjau semua file log. Untuk fleksibilitas lebih dalam menyortir dan mencari data log, aktifkan pengaturan untuk mengunggah log audit CloudWatch dan melihatnya menggunakan CloudWatch antarmuka.

Untuk melihat data audit dengan lebih banyak jenis bidang dan dengan output dalam JSON format, Anda juga dapat menggunakan fitur Aliran Aktivitas Database. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

File log audit meliputi informasi yang dipisahkan koma berikut dalam baris, dalam urutan yang ditentukan:

| Bidang | Deskripsi |
|---|---|
| timestamp | Stempel waktu Unix untuk peristiwa yang dicatat lognya dengan tingkat presisi mikrodetik. |
| serverhost | Nama instans tempat peristiwa dicatat. |
| nama pengguna | Nama pengguna yang terhubung dengan pengguna. |
| host | Host yang terhubung ke pengguna. |
| connectionid | Nomor ID koneksi untuk operasi yang dicatat. |
| queryid | Nomor ID kueri, yang dapat digunakan untuk menemukan peristiwa tabel hubungan dan kueri terkait. Untuk peristiwa `TABLE`, beberapa baris ditambahkan. |
| operation | Jenis tindakan yang dicatat. Kemungkinan nilainya adalah: `CONNECT`, `QUERY`, `READ`, `WRITE`, `CREATE`, `ALTER`, `RENAME`, dan `DROP`. |
| database | Basis data aktif, yang diatur oleh perintah `USE`. |
| object | Untuk peristiwa `QUERY`, nilai ini menunjukkan kueri yang dilakukan basis data. Untuk peristiwa `TABLE`, nilai ini menunjukkan nama tabel. |
| retcode | Kode yang dihasilkan dari operasi yang dicatat lognya. |

# Replikasi dengan Amazon Aurora MySQL

Fitur-fitur replikasi Aurora MySQL merupakan kunci untuk ketersediaan dan performa yang tinggi dari klaster Anda. Aurora mempermudah pembuatan atau perubahan ukuran klaster dengan maksimal 15 Replika Aurora.

Semua replika beroperasi dari data acuan yang sama. Jika beberapa instans basis data menjadi offline, yang lain akan tetap tersedia untuk melanjutkan pemrosesan kueri atau mengambil alih sebagai penulis jika diperlukan. Aurora akan secara otomatis menyebarkan koneksi hanya-baca Anda ke beberapa instans basis data, sehingga membantu klaster Aurora mendukung beban kerja sarat kueri.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara kerja replikasi Aurora MySQL dan cara menyesuaikan pengaturan replikasi untuk ketersediaan dan performa terbaik.

Topik


- Menggunakan Replika Aurora
- Opsi replikasi untuk Amazon Aurora MySQL
- Pertimbangan performa untuk replikasi Amazon Aurora MySQL
- Restart nol-downtime () untuk ZDR Amazon Aurora My SQL
- Mengkonfigurasi filter replikasi dengan Aurora My SQL
- Memantau replikasi Amazon Aurora MySQL
- Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS
- Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner)
- Menggunakan replikasi GTID berbasis


## Menggunakan Replika Aurora

Replika Aurora adalah titik akhir independen dalam klaster basis data Aurora, yang paling berguna untuk menskalakan operasi baca dan meningkatkan ketersediaan. Hingga 15 Replika Aurora dapat didistribusikan ke seluruh Zona Ketersediaan yang dicakup oleh sebuah klaster DB di satu Wilayah AWS. Meskipun volume klaster DB terdiri dari beberapa salinan data untuk klaster DB, data dalam volume klaster direpresentasikan sebagai volume logis tunggal untuk instans primer dan Replika Aurora dalam klaster DB tersebut. Untuk informasi selengkapnya tentang Replika Aurora, lihat Replika Aurora.

Replika Aurora berfungsi dengan baik untuk penskalaan baca karena ditujukan sepenuhnya untuk operasi baca pada volume klaster Anda. Operasi tulis dikelola oleh instans primer. Karena volume klaster dibagikan di antara semua instans dalam klaster DB Aurora MySQL Anda, tidak diperlukan pekerjaan tambahan untuk mereplikasi salinan data untuk setiap Replika Aurora. Sebaliknya, replika baca MySQL harus mengulang, pada thread tunggal, semua operasi tulis dari instans DB sumber daya ke penyimpanan data lokalnya. Batasan ini dapat memengaruhi kemampuan replika baca MySQL untuk mendukung volume lalu lintas baca yang besar.

Dengan Aurora MySQL, saat Replika Aurora dihapus, titik akhir instansnya akan segera dihapus, dan Replika Aurora akan dihapus dari titik akhir pembaca. Jika ada pernyataan yang berjalan di Replika Aurora yang dihapus, ada masa tenggang selama tiga menit. Pernyataan yang ada dapat diselesaikan sepenuhnya selama masa tenggang ini. Setelah masa tenggang berakhir, Replika Aurora akan dinonaktifkan dan dihapus.

**Important**

Replika Aurora untuk Aurora MySQL selalu menggunakan tingkat isolasi transaksi default `REPEATABLE READ` untuk operasi pada tabel InnoDB. Anda dapat menggunakan perintah `SET TRANSACTION ISOLATION LEVEL` untuk mengubah tingkat transaksi hanya untuk instans primer klaster DB Aurora MySQL. Pembatasan ini bertujuan untuk menghindari kunci tingkat pengguna pada Replika Aurora, dan memungkinkan penskalaan Replika Aurora untuk mendukung ribuan koneksi pengguna aktif sambil tetap menjaga lag replika seminimal mungkin.

**Note**

Pernyataan DDL yang berjalan pada instans primer dapat menginterupsi koneksi basis data pada Replika Aurora terkait. Jika koneksi Replika Aurora secara aktif menggunakan objek basis data, seperti tabel, dan objek tersebut dimodifikasi pada instans primer menggunakan pernyataan DDL, koneksi Replika Aurora dapat terinterupsi.

**Note**

Wilayah Tiongkok (Ningxia) tidak mendukung replika baca lintas Wilayah.

# Opsi replikasi untuk Amazon Aurora MySQL

Anda dapat mengatur replikasi dengan opsi-opsi berikut:

- Dua cluster DB MySQL Aurora berbeda Wilayah AWS, dengan membuat replika baca lintas wilayah dari cluster DB MySQL Aurora.

  Untuk informasi selengkapnya, lihat Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS.
- Dua cluster Aurora MySQL DB yang sama, dengan menggunakan replikasi log biner MySQL ( Wilayah AWS binlog).

  Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).
- Instans DB RDS for MySQL sebagai sumber dan klaster DB Aurora MySQL, dengan membuat replika baca Aurora dari instans DB RDS for MySQL.

  Anda dapat menggunakan pendekatan ini untuk memindahkan perubahan data yang ada dan yang sedang berlangsung ke Aurora MySQL selama migrasi ke Aurora. Untuk informasi selengkapnya, lihat Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora.

  Anda juga dapat menggunakan pendekatan ini untuk meningkatkan skalabilitas kueri baca untuk data Anda. Caranya adalah dengan mengueri data menggunakan satu atau beberapa instans DB dalam klaster Aurora MySQL hanya baca. Untuk informasi selengkapnya, lihat Penskalaan bacaan untuk SQL database Saya dengan Amazon Aurora.
- Cluster DB MySQL Aurora dalam satu dan Wilayah AWS hingga lima cluster Aurora MySQL DB read-only Aurora di Wilayah yang berbeda, dengan membuat database global Aurora.

  Anda dapat menggunakan basis data global Aurora untuk mendukung aplikasi dengan jejak global. Klaster DB Aurora MySQL primer memiliki instans Penulis dan maksimal 15 Replika Aurora. Klaster DB Aurora MySQL sekunder hanya baca masing-masing dapat terdiri dari maksimal 16 Replika Aurora. Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

> **Note**
>
> Me-boot ulang instans primer klaster DB Amazon Aurora juga akan secara otomatis mem-boot ulang Replika Aurora untuk klaster DB tersebut. Hal ini bertujuan untuk menetapkan kembali titik masuk yang menjamin konsistensi baca/tulis di seluruh klaster DB.

## Pertimbangan performa untuk replikasi Amazon Aurora MySQL

Fitur berikut dapat membantu Anda menyesuaikan performa replikasi Aurora MySQL.

Fitur kompresi log replika secara otomatis mengurangi bandwidth jaringan untuk pesan replikasi. Karena setiap pesan ditransmisikan ke semua Replika Aurora, manfaat yang diberikan akan lebih besar untuk klaster yang lebih besar. Fitur ini memerlukan beberapa overhead CPU pada simpul penulis untuk melakukan kompresi. Fitur ini selalu diaktifkan di Aurora MySQL versi 2 dan versi 3.

Fitur pemfilteran binlog secara otomatis mengurangi bandwidth jaringan untuk pesan replikasi. Karena Replika Aurora tidak menggunakan informasi binlog yang disertakan dalam pesan replikasi, data tersebut dihilangkan dari pesan yang dikirim ke simpul-simpul tersebut.

Di Aurora MySQL versi 2, Anda dapat mengontrol fitur ini dengan mengubah parameter `aurora_enable_repl_bin_log_filtering`. Parameter ini aktif secara default. Karena optimisasi ini dimaksudkan agar bersifat transparan, Anda dapat menonaktifkan pengaturan ini hanya selama diagnosis atau pemecahan masalah yang terkait dengan replikasi. Misalnya, Anda dapat melakukannya untuk mencocokkan dengan perilaku klaster Aurora MySQL lama yang tidak menyediakan fitur ini.

Pemfilteran Binlog selalu diaktifkan di Aurora MySQL versi 3.

## Restart nol-downtime () untuk ZDR Amazon Aurora My SQL

Fitur zero-downtime restart (ZDR) dapat mempertahankan beberapa atau semua koneksi aktif ke instans DB selama jenis restart tertentu. ZDRberlaku untuk memulai ulang yang dilakukan Aurora secara otomatis untuk menyelesaikan kondisi kesalahan, misalnya ketika replika mulai tertinggal terlalu jauh di belakang sumbernya.

> **Important**
>
> ZDRMekanisme ini beroperasi atas dasar upaya terbaik. SQLVersi Aurora My, kelas instans, kondisi kesalahan, SQL operasi yang kompatibel, dan faktor lain yang menentukan di mana ZDR berlaku dapat berubah sewaktu-waktu.

ZDRuntuk Aurora My SQL 2.x membutuhkan versi 2.10 dan lebih tinggi. ZDRtersedia di semua versi minor Aurora My SQL 3.x. Di Aurora My SQL versi 2 dan 3, ZDR mekanisme diaktifkan secara default dan Aurora tidak menggunakan parameter. `aurora_enable_zdr`

Di halaman Peristiwa, Aurora melaporkan aktivitas yang terkait dengan zero-downtime restart. Aurora merekam suatu peristiwa ketika mencoba memulai ulang menggunakan mekanisme. ZDR Peristiwa ini menyatakan alasan Aurora melakukan pengaktifan ulang. Kemudian, Aurora mencatat peristiwa lain ketika pengaktifan ulang selesai. Peristiwa akhir ini melaporkan berapa lama prosesnya, dan berapa banyak koneksi yang dipertahankan atau diputus selama pengaktifan ulang. Anda dapat melihat log kesalahan basis data untuk melihat detail selengkapnya tentang apa yang terjadi selama pengaktifan ulang.

Meskipun koneksi tetap utuh setelah ZDR operasi yang berhasil, beberapa variabel dan fitur diinisialisasi ulang. Jenis informasi berikut ini tidak dipertahankan selama pengaktifan ulang yang disebabkan oleh zero-downtime restart:

- Variabel global. Aurora memulihkan variabel sesi, tetapi tidak memulihkan variabel global setelah pengaktifan ulang.
- Variabel status. Secara khusus, nilai uptime yang dilaporkan oleh status mesin akan direset.
- `LAST_INSERT_ID`.
- Status `auto_increment` dalam memori untuk tabel. Status inkremen otomatis dalam memori diinisialisasi ulang. Untuk informasi selengkapnya tentang nilai kenaikan otomatis, lihat Panduan SQLReferensi Saya.
- Informasi diagnostik dari tabel `INFORMATION_SCHEMA` dan `PERFORMANCE_SCHEMA`. Informasi diagnostik ini juga muncul dalam output perintah seperti `SHOW PROFILE` dan `SHOW PROFILES`.

Tabel berikut menunjukkan versi, peran instans, dan keadaan lain yang menentukan apakah Aurora dapat menggunakan ZDR mekanisme saat memulai ulang instans DB di cluster Anda.

| Aurora Versi saya SQL | ZDRberlaku untuk penulis? | ZDRberlaku untuk pembaca? | ZDRselalu diaktifkan? | Catatan |
|---|---|---|---|---|
| 2.x, lebih rendah dari 2.10.0 | Tidak | Tidak | N/A | ZDRtidak tersedia untuk versi ini. |
| 2.10.0—2.11.0 | Ya | Ya | Ya | Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi.<br><br>Aurora membatalkan koneksi yang menggunakanTLS/SSL, tabel sementara, kunci meja, atau kunci pengguna. |
| 2.11.1 dan lebih tinggi | Ya | Ya | Ya | Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi.<br><br>Aurora membatalkan koneksi apa pun yang menggunakan tabel sementara, kunci meja, atau kunci pengguna. |
| 3.01—3.03 | Ya | Ya | Ya | Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi.<br><br>Aurora membatalkan koneksi yang menggunakanTLS/SSL, tabel sementara, kunci meja, atau kunci pengguna. |
| 3.04 dan lebih tinggi | Ya | Ya | Ya | Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi. |

| Aurora Versi saya SQL | ZDRberlaku untuk penulis? | ZDRberlaku untuk pembaca? | ZDRselalu diaktifkan? | Catatan |
|---|---|---|---|---|
| | | | | Aurora membatalkan koneksi apa pun yang menggunakan tabel sementara, kunci meja, atau kunci pengguna. |

## Mengkonfigurasi filter replikasi dengan Aurora My SQL

Anda dapat menggunakan filter replikasi untuk menentukan basis data dan tabel mana yang direplikasi dengan replika baca. Filter replikasi dapat menyertakan basis data dan tabel dalam replikasi atau mengecualikannya dari replikasi.

Berikut ini adalah beberapa kasus penggunaan untuk filter replikasi:

- Untuk mengurangi ukuran replika baca. Dengan filter replikasi, Anda dapat mengecualikan basis data dan tabel yang tidak diperlukan pada replika baca.
- Untuk mengecualikan basis data dan tabel dari replika baca untuk alasan keamanan.
- Untuk mereplikasi basis data yang berbeda dan tabel untuk kasus penggunaan tertentu di replika baca yang berbeda. Misalnya, Anda mungkin menggunakan replika baca khusus untuk analitik atau penyerpihan.
- Untuk cluster DB yang telah membaca replika di berbagai Wilayah AWS, untuk mereplikasi database atau tabel yang berbeda dalam berbagai Wilayah AWS.
- Untuk menentukan database dan tabel mana yang direplikasi dengan cluster Aurora My SQL DB yang dikonfigurasi sebagai replika dalam topologi replikasi masuk. Untuk informasi selengkapnya tentang konfigurasi ini, silakan lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Topik

- Mengatur parameter penyaringan replikasi untuk Aurora My SQL
- Batasan penyaringan replikasi untuk Aurora My SQL
- Contoh penyaringan replikasi untuk Aurora My SQL
- Melihat filter replikasi untuk replika baca

## Mengatur parameter penyaringan replikasi untuk Aurora My SQL

Untuk mengonfigurasi filter replikasi, atur parameter berikut:

- `binlog-do-db` – Mereplikasi perubahan ke log biner yang ditentukan. Ketika Anda mengatur parameter ini untuk klaster sumber binlog, hanya log biner yang ditentukan dalam parameter yang direplikasi.
- `binlog-ignore-db` – Jangan mereplikasi ke log biner yang ditentukan. Ketika parameter `binlog-do-db` diatur untuk klaster sumber binlog, parameter ini tidak dievaluasi.
- `replicate-do-db` – Mereplikasi perubahan ke basis data yang ditentukan. Ketika Anda mengatur parameter ini untuk klaster replika binlog, hanya basis data yang ditentukan dalam parameter yang direplikasi.
- `replicate-ignore-db` – Jangan mereplikasi perubahan ke basis data yang ditentukan. Ketika parameter `replicate-do-db` diatur untuk klaster replika binlog, parameter ini tidak dievaluasi.
- `replicate-do-table` – Mereplikasi perubahan ke tabel yang ditentukan. Ketika Anda menetapkan parameter ini untuk replika baca, hanya tabel yang ditentukan dalam parameter yang direplikasi. Selain itu, ketika parameter `replicate-ignore-db` atau `replicate-do-db` diatur, pastikan untuk menyertakan basis data yang mencakup tabel tertentu dalam replikasi dengan klaster replika binlog.
- `replicate-ignore-table` – Jangan mereplikasi perubahan ke tabel yang ditentukan. Ketika parameter `replicate-do-table` diatur untuk klaster replika binlog, parameter ini tidak dievaluasi.
- `replicate-wild-do-table` – Mereplikasi tabel berdasarkan basis data dan pola nama tabel yang ditentukan. Karakter wildcard `%` dan `_` didukung. Ketika parameter `replicate-ignore-db` atau `replicate-do-db` diatur, pastikan untuk menyertakan basis data yang mencakup tabel tertentu dalam replikasi dengan klaster replika binlog.
- `replicate-wild-ignore-table` – Jangan mereplikasi tabel berdasarkan basis data dan pola nama tabel yang ditentukan. Karakter wildcard `%` dan `_` didukung. Ketika parameter `replicate-wild-do-table` atau `replicate-do-table` diatur untuk klaster replika binlog, parameter ini tidak dievaluasi.

Parameter dievaluasi sesuai dengan urutannya dalam daftar. Untuk informasi selengkapnya tentang cara kerja parameter ini, lihat SQL Dokumentasi saya:

- Untuk informasi umum, lihat Opsi dan Variabel Server Replika.

- Untuk informasi tentang cara parameter pemfilteran replikasi basis data dievaluasi, lihat Evaluation of Database-Level Replication and Binary Logging Options.
- Untuk informasi tentang cara parameter filter replikasi basis data dievaluasi, lihat Evaluasi Opsi Replikasi Tingkat Tabel.

Secara default, masing-masing parameter ini memiliki nilai kosong. Pada setiap klaster binlog, Anda dapat menggunakan parameter ini untuk mengatur, mengubah, dan menghapus filter replikasi. Ketika Anda menetapkan salah satu parameter ini, pisahkan masing-masing filter dari yang lain dengan koma.

Anda dapat menggunakan karakter wildcard `%` dan `_` dalam parameter `replicate-wild-do-table` dan `replicate-wild-ignore-table`. Parameter wildcard `%` mencocokkan jumlah karakter berapa pun, dan wildcard `_` hanya mencocokkan satu karakter.

Format pencatatan log biner instans DB sumber penting untuk replikasi karena menentukan catatan perubahan data. Pengaturan parameter `binlog_format` menentukan apakah replikasi berbasis baris atau berbasis pernyataan. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aurora untuk database Single-AZ.

> **Note**
>
> Semua pernyataan bahasa definisi data (DDL) direplikasi sebagai pernyataan, terlepas dari `binlog_format` pengaturan pada instance DB sumber.

## Batasan penyaringan replikasi untuk Aurora My SQL

Batasan berikut berlaku untuk penyaringan replikasi untuk Aurora My: SQL

- Filter replikasi hanya didukung untuk Aurora SQL My versi 3.
- Setiap parameter filter replikasi memiliki batas 2.000 karakter.
- Koma tidak didukung dalam filter replikasi.
- Filter replikasi tidak mendukung transaksi XA.

  Untuk informasi selengkapnya, lihat Pembatasan Transaksi XA di SQL dokumentasi Saya.

## Contoh penyaringan replikasi untuk Aurora My SQL

Untuk mengonfigurasi filter replikasi untuk replika baca, modifikasi parameter filter replikasi dalam grup parameter klaster DB yang terkait dengan replika baca.

> **Note**
>
> Anda tidak dapat memodifikasi grup parameter klaster DB default. Jika replika baca menggunakan grup parameter default, buat grup parameter baru dan kaitkan dengan replika baca tersebut. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat .

Anda dapat mengatur parameter dalam grup parameter cluster DB menggunakan AWS Management Console, AWS CLI, atau RDSAPI. Untuk informasi tentang mengatur parameter, lihat Memodifikasi parameter dalam grup parameter DB di . Ketika Anda mengatur parameter dalam grup parameter klaster DB, semua klaster DB yang terkait dengan grup parameter tersebut akan menggunakan pengaturan parameter. Jika Anda mengatur parameter filter replikasi dalam grup parameter klaster DB, pastikan bahwa grup parameter dikaitkan hanya dengan klaster replika baca. Biarkan parameter filter replikasi kosong untuk instans DB sumber.

Contoh berikut mengatur parameter menggunakan AWS CLI. Contoh-contoh ini diatur `ApplyMethod` `immediate` agar perubahan parameter terjadi segera setelah CLI perintah selesai. Jika Anda ingin menerapkan perubahan tertunda setelah replika baca di-boot ulang, atur `ApplyMethod` ke `pending-reboot`.

Contoh berikut mengatur filter replikasi:

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example Menyertakan basis data dalam replikasi

Contoh berikut menyertakan basis data `mydb1` dan `mydb2` dalam replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example Menyertakan tabel dalam replikasi

Contoh berikut menyertakan tabel `table1` dan `table2` dalam `mydb1` basis data dalam replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example Menyertakan tabel dalam replikasi menggunakan karakter wildcard

Contoh berikut menyertakan tabel dengan nama berawalan `order` dan `return` dalam basis data `mydb` dalam replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example Mengecualikan basis data dari replikasi

Contoh berikut mengecualikan basis data `mydb5` dan `mydb6` dari replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example Mengecualikan tabel dari replikasi

Contoh berikut mengecualikan tabel `table1` dalam basis data `mydb5` dan `table2` dalam basis data `mydb6` dari replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
```

```
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example Mengecualikan tabel dari replikasi menggunakan karakter wildcard

Contoh berikut mengecualikan tabel dengan nama berawalan `order` dan `return` dalam basis data `mydb7` dari replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

## Melihat filter replikasi untuk replika baca

Anda dapat melihat filter replikasi untuk replika baca dengan cara berikut:

- Memeriksa pengaturan parameter filter replikasi dalam grup parameter yang terkait dengan replika baca.

  Untuk petunjuk, silakan lihat Melihat nilai parameter untuk grup parameter DB di Aurora.
- Di SQL klien Saya, sambungkan ke replika baca dan jalankan `SHOW REPLICA STATUS` pernyataan.

Dalam output, bidang berikut menunjukkan filter replikasi untuk replika baca:

- `Binlog_Do_DB`
- `Binlog_Ignore_DB`
- `Replicate_Do_DB`
- `Replicate_Ignore_DB`
- `Replicate_Do_Table`
- `Replicate_Ignore_Table`
- `Replicate_Wild_Do_Table`
- `Replicate_Wild_Ignore_Table`

Untuk informasi selengkapnya tentang bidang ini, lihat Memeriksa Status Replikasi di SQL Dokumentasi saya.

## Memantau replikasi Amazon Aurora MySQL

Penskalaan baca dan ketersediaan yang tinggi tergantung dari waktu lag minimal. Anda dapat memantau seberapa jauh Aurora Replica tertinggal dari instance utama cluster DB MySQL Aurora Anda dengan memantau metrik Amazon. CloudWatch `AuroraReplicaLag` Metrik `AuroraReplicaLag` dicatat dalam setiap Replika Aurora.

Instans primer DB juga merekam metrik `AuroraReplicaLagMaximum` dan `AuroraReplicaLagMinimum` Amazon CloudWatch. Metrik `AuroraReplicaLagMaximum` akan mencatat jumlah maksimum lag antara instans DB primer dan setiap Replika Aurora dalam klaster DB. Metrik `AuroraReplicaLagMinimum` akan mencatat jumlah minimum lag antara instans DB primer dan setiap Replika Aurora dalam klaster DB.

Jika Anda membutuhkan nilai terbaru untuk lag Aurora Replica, Anda dapat memeriksa metrik `AuroraReplicaLag` di Amazon. CloudWatch Lag Replika Aurora juga dicatat pada setiap Replika Aurora dari klaster DB Aurora MySQL Anda dalam tabel `information_schema.replica_host_status`. Untuk informasi selengkapnya tentang tabel ini, lihat information_schema.replica_host_status.

Untuk informasi selengkapnya tentang pemantauan instans dan CloudWatch metrik RDS, lihat. Memantau metrik di klaster Amazon Aurora

# Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS

Anda dapat membuat kluster DB MySQL Amazon Aurora sebagai replika baca di cluster DB sumber yang berbeda. Wilayah AWS Mengambil pendekatan ini dapat meningkatkan kemampuan pemulihan bencana Anda, memungkinkan Anda menskalakan operasi baca menjadi Wilayah AWS yang lebih dekat dengan pengguna Anda, dan membuatnya lebih mudah untuk bermigrasi dari satu Wilayah AWS ke yang lain.

Anda dapat membuat replika baca klaster DB terenkripsi atau tidak terenkripsi. Replika baca harus dienkripsi jika klaster DB sumber dienkripsi.

Untuk setiap klaster DB sumber, Anda dapat memiliki hingga lima klaster DB lintas Wilayah yang berupa replika baca.

> **Note**
>
> Sebagai alternatif untuk replika baca lintas Wilayah, Anda dapat menskalakan operasi baca dengan waktu lag minimal dengan menggunakan basis data global Aurora. Database global Aurora memiliki cluster DB Aurora primer dalam satu Wilayah AWS dan hingga 10 cluster DB read-only sekunder di Wilayah yang berbeda. Setiap klaster DB sekunder dapat menyertakan hingga 16 (bukan 15) Replika Aurora. Replikasi dari klaster DB primer ke semua sekunder ditangani oleh lapisan penyimpanan Aurora, bukan oleh mesin basis data, sehingga waktu lag untuk mereplikasi perubahan menjadi minimal—biasanya, kurang dari 1 detik. Mengecualikan mesin basis data dari proses replikasi berarti bahwa mesin basis data dikhususkan untuk memproses beban kerja. Ini juga berarti bahwa Anda tidak perlu mengonfigurasi atau mengelola replikasi binlog (binlog biner) Aurora MySQL. Untuk mempelajari selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

Saat Anda membuat replika baca cluster MySQL DB Aurora di tempat lain Wilayah AWS, Anda harus mengetahui hal berikut:

- Klaster DB sumber Anda dan klaster DB replika baca lintas Wilayah Anda dapat memiliki hingga 15 Replika Aurora, di samping instans primer untuk klaster DB tersebut. Dengan menggunakan fungsi ini, Anda dapat menskalakan operasi baca untuk sumber Wilayah AWS dan target Wilayah AWS replikasi Anda.
- Dalam sebuah skenario lintas Wilayah, terdapat lebih banyak waktu lag antara klaster DB sumber dan replika baca karena saluran jaringan yang lebih panjang antar- Wilayah AWS.

- Data yang ditransfer untuk replikasi lintas Wilayah akan menimbulkan biaya transfer data Amazon RDS. Tindakan replikasi lintas Wilayah berikut menghasilkan biaya untuk data yang ditransfer dari Wilayah AWS sumber:
  - Saat Anda membuat replika baca, Amazon RDS mengambil snapshot dari cluster sumber dan mentransfer snapshot ke Wilayah AWS yang menyimpan replika baca.
  - Untuk setiap modifikasi data yang dibuat dalam database sumber, Amazon RDS mentransfer data dari wilayah sumber ke Wilayah AWS yang menyimpan replika baca.

  Untuk informasi selengkapnya tentang harga transfer data Amazon RDS, lihat Harga Amazon Aurora.
- Anda dapat menjalankan beberapa tindakan pembuatan atau penghapusan konkuren untuk replika baca yang mengacu pada klaster DB sumber yang sama. Namun, Anda harus tetap berada dalam batasan lima replika baca untuk setiap klaster DB sumber.
- Agar replikasi dapat beroperasi secara efektif, setiap replika baca harus memiliki jumlah sumber daya komputasi dan penyimpanan yang sama dengan klaster DB sumber. Jika Anda menskalakan klaster DB sumber, Anda juga harus menskalakan replika baca.

Topik


- Sebelum Anda memulai
- Membuat cluster DB replika baca lintas wilayah untuk Aurora My SQL
- Mempromosikan replika baca ke cluster DB untuk Aurora My SQL
- Memecahkan masalah Replika lintas wilayah untuk Amazon Aurora My SQL


## Sebelum Anda memulai

Sebelum Anda dapat membuat klaster DB Aurora MySQL yang merupakan replika baca lintas Wilayah, Anda harus mengaktifkan pencatatan log biner pada klaster DB Aurora MySQL sumber Anda. Replikasi lintas wilayah untuk Aurora MySQL menggunakan replikasi biner MySQL untuk mengulang perubahan pada klaster DB replika baca lintas Wilayah.

Untuk mengaktifkan pencatatan log biner pada sebuah klaster DB Aurora MySQL, perbarui parameter `binlog_format` untuk klaster DB sumber Anda. Parameter `binlog_format` adalah parameter tingkat klaster yang berada dalam grup parameter klaster default. Jika klaster DB Anda menggunakan grup parameter klaster DB default, buat sebuah grup parameter klaster DB baru untuk memodifikasi pengaturan `binlog_format`. Kami sarankan Anda mengatur `binlog_format` ke

`MIXED`. Namun, Anda juga dapat mengatur `binlog_format` ke `ROW` atau `STATEMENT` jika Anda memerlukan format binlog tertentu. Boot ulang klaster DB Aurora Anda agar perubahan dapat diterapkan.

Untuk informasi selengkapnya tentang menggunakan pencatatan log biner dengan Aurora MySQL, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner). Untuk informasi selengkapnya tentang memodifikasi parameter konfigurasi Aurora MySQL, lihat Parameter klaster DB dan instans DB Amazon Aurora dan .

## Membuat cluster DB replika baca lintas wilayah untuk Aurora My SQL

Anda dapat membuat klaster Aurora DB yang merupakan replika baca lintas wilayah dengan menggunakan, AWS Command Line Interface (AWS CLI) AWS Management Console, atau Amazon. RDS API Anda dapat membuat replika baca klaster lintas Wilayah dari klaster DB yang terenkripsi atau tidak terenkripsi.

Saat Anda membuat replika baca Lintas wilayah untuk Aurora My SQL dengan menggunakan, AWS Management Console Amazon RDS membuat cluster DB di target Wilayah AWS, dan kemudian secara otomatis membuat instance DB yang merupakan instance utama untuk cluster DB tersebut.

Saat Anda membuat replika baca lintas wilayah menggunakan AWS CLI or RDSAPI, pertama-tama Anda membuat cluster DB di target Wilayah AWS dan menunggu hingga menjadi aktif. Setelah aktif, Anda kemudian dapat membuat sebuah instans DB yang merupakan instans primer untuk klaster DB tersebut.

Replikasi dimulai saat instans primer dari klaster DB replika baca menjadi tersedia.

Gunakan prosedur berikut untuk membuat replika baca lintas wilayah dari cluster Aurora My DB. SQL Prosedur ini berfungsi untuk pembuatan replika baca dari klaster DB terenkripsi atau tidak terenkripsi.

Konsol

Untuk membuat cluster Aurora My SQL DB yang merupakan replika baca lintas wilayah dengan AWS Management Console

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS yang menghosting cluster DB sumber Anda.

3. Di panel navigasi, pilih Basis Data.
4. Pilih klaster DB yang ingin Anda buat replika baca lintas Wilayahnya.
5. Untuk Tindakan, pilih Buat replika baca lintas Wilayah.
6. Pada halaman Buat replika baca lintas wilayah, pilih pengaturan opsi untuk klaster DB replika baca lintas Wilayah Anda, seperti yang dijelaskan dalam tabel berikut.

| Opsi | Deskripsi |
| --- | --- |
| Wilayah tujuan | Pilih Wilayah AWS untuk meng-host cluster DB replika baca Lintas wilayah yang baru. |
| Grup subnet DB tujuan | Pilih grup subnet DB yang akan digunakan untuk klaster DB replika baca lintas Wilayah. |
| Dapat diakses publik | Pilih Ya untuk memberikan alamat IP publik ke klaster DB replika baca lintas Wilayah; jika tidak, pilih Tidak. |
| Enkripsi | Pilih Aktifkan Enkripsi untuk mengaktifkan enkripsi saat diam untuk klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora. |
| AWS KMS key | Hanya tersedia jika Enkripsi diatur ke Aktifkan Enkripsi. Pilih yang akan digunakan AWS KMS key untuk mengenkripsi cluster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora. |
| Kelas instans DB | Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk instans primer dalam DB klaster. Untuk informasi selengkapnya tentang opsi kelas instans DB, lihat Kelas instans Amazon Aurora DB. |
| Deployment Multi-AZ | Pilih Ya untuk membuat replika baca cluster DB baru di Availability Zone lain dalam target Wilayah AWS dukungan failover. Untuk informasi selengkapnya |

| Opsi | Deskripsi |
| --- | --- |
| | tentang beberapa Availability Zone, lihat Wilayah dan Zona Ketersediaan. |
| Sumber replika baca | Pilih klaster DB sumber yang akan dibuatkan replika baca lintas Wilayahnya. |
| Pengidentifikasi instans DB | Ketikkan sebuah nama untuk instans primer dalam klaster DB replika baca lintas Wilayah Anda. Pengidentifikasi ini digunakan dalam alamat titik akhir untuk instans primer klaster DB baru.<br><br>Pengidentifikasi instans DB memiliki batasan berikut:<br><br>• Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung.<br>• Karakter pertamanya harus berupa huruf.<br>• Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.<br>• Itu harus unik untuk semua instans DB untuk masing-masing Akun AWS, untuk masing-masing Wilayah AWS.<br><br>Karena klaster DB replika baca lintas Wilayah dibuat dari sebuah snapshot klaster DB sumber, nama pengguna master dan kata sandi master untuk replika baca harus sama dengan nama pengguna master dan kata sandi master untuk klaster DB sumber. |

| Opsi | Deskripsi |
| --- | --- |
| Pengidentifikasi klaster DB | Ketikkan sebuah nama untuk klaster DB replika baca lintas Wilayah yang unik untuk akun Anda dalam Wilayah AWS target untuk replika Anda. Pengident ifikasi ini digunakan dalam alamat titik akhir klaster untuk klaster DB Anda. Untuk informasi tentang titik akhir klaster, lihat Koneksi titik akhir Amazon Aurora.<br><br>Pengidentifikasi klaster DB memiliki batasan berikut:<br><br>• Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung.<br>• Karakter pertamanya harus berupa huruf.<br>• Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.<br>• Itu harus unik untuk semua cluster DB untuk masing-masing Akun AWS, untuk masing-masing Wilayah AWS. |
| Prioritas | Pilih sebuah prioritas failover untuk instans primer dari klaster DB baru. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora. |
| Port basis data | Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Aurora DB cluster default ke default My SQL port, 3306. Firewall di beberapa perusahaan memblokir koneksi ke port ini. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru. |

| Opsi | Deskripsi |
| --- | --- |
| Pemantauan yang ditingkatkan | Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan pengumpulan metrik secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |
| Peran Pemantauan | Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Pilih IAM peran yang Anda buat untuk mengizinkan Amazon RDS berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default untuk RDS membuat peran untuk nama Andards-monitoring-role . Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |
| Granularitas | Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda. |
| Peningkatan versi minor otomatis | Pengaturan ini tidak berlaku untuk klaster Aurora My SQL DB.<br><br>Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat. Pembaruan mesin database untuk Amazon Aurora My SQL |

7. Pilih Buat untuk membuat replika baca lintas Wilayah Anda untuk Aurora.

AWS CLI

Untuk membuat cluster Aurora My SQL DB yang merupakan replika baca lintas wilayah dengan CLI

1. Panggil AWS CLI create-db-clusterperintah di Wilayah AWS tempat Anda ingin membuat cluster DB replika baca. Sertakan `--replication-source-identifier` opsi dan tentukan Amazon Resource Name (ARN) dari cluster DB sumber untuk membuat replika baca.

   Untuk replikasi lintas Wilayah yang mengenkripsi klaster DB yang diidentifikasi berdasarkan `--replication-source-identifier`, tentukan opsi `--kms-key-id` dan opsi `--storage-encrypted`.

   > **Note**
   >
   > Anda dapat mengatur replikasi lintas Wilayah dari sebuah klaster DB yang tidak terenkripsi ke replika baca terenkripsi dengan menentukan `--storage-encrypted` dan menyediakan sebuah nilai untuk `--kms-key-id`.

   Anda tidak dapat menentukan parameter `--master-username` dan `--master-user-password`. Nilai tersebut diambil dari klaster DB sumber.

   Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB tidak terenkripsi di Wilayah us-west-2. Perintah ini dipanggil di Wilayah us-east-1. Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Untuk Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Contoh kode berikut ini menciptakan sebuah replika baca di Region us-east-1 dari sebuah snapshot klaster DB terenkripsi di Region us-west-2. Perintah ini dipanggil di Wilayah us-east-1.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
  --storage-encrypted
```

Untuk Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

`--source-region`Opsi ini diperlukan untuk replikasi lintas wilayah antara Wilayah AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat), di mana cluster DB yang diidentifikasi oleh dienkripsi. `--replication-source-identifier` Untuk`--source-region`, tentukan Wilayah AWS cluster DB sumber.

Jika `--source-region` tidak ditentukan, tentukan nilai `--pre-signed-url`. Presigned URL adalah URL yang berisi permintaan ditandatangani Signature Version 4 untuk `create-db-cluster` perintah yang dipanggil di sumber Wilayah AWS. Untuk mempelajari lebih lanjut tentang `pre-signed-url` opsi, lihat create-db-clusterdi Referensi AWS CLI Perintah.

2. Periksa apakah cluster DB telah tersedia untuk digunakan dengan menggunakan AWS CLI describe-db-clustersperintah, seperti yang ditunjukkan pada contoh berikut.

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

Saat hasil **`describe-db-clusters`** menunjukkan sebuah status dari `available`, ciptakan instans primer untuk klaster DB sehingga replikasi dapat dimulai. Untuk melakukannya, gunakan AWS CLI create-db-instanceperintah seperti yang ditunjukkan pada contoh berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r5.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora-mysql
```

Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r5.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora-mysql
```

Ketika instans DBdibuat dan tersedia, replikasi akan dimulai. Anda dapat menentukan apakah instans DB tersedia dengan memanggil AWS CLI describe-db-instancesperintah.

RDS API

Untuk membuat cluster Aurora My SQL DB yang merupakan replika baca lintas wilayah dengan API

1. Panggil reateDBCluster operasi RDS API C di Wilayah AWS tempat Anda ingin membuat cluster DB replika baca. Sertakan `ReplicationSourceIdentifier` parameter dan tentukan Amazon Resource Name (ARN) dari cluster DB sumber untuk membuat replika baca.

   Untuk replikasi lintas Wilayah yang mengenkripsi klaster DB yang diidentifikasi berdasarkan `ReplicationSourceIdentifier`, tentukan parameter `KmsKeyId` dan atur parameter `StorageEncrypted` ke `true`.

   > **Note**
   >
   > Anda dapat mengatur replikasi lintas Wilayah dari sebuah klaster DB yang tidak terenkripsi ke replika baca terenkripsi dengan menentukan `StorageEncrypted` sebagai **`true`** dan menyediakan sebuah nilai untuk `KmsKeyId`. Dalam hal ini, Anda tidak perlu menentukan `PreSignedUrl`.

   Anda tidak perlu menyertakan parameter `MasterUsername` dan `MasterUserPassword` karena nilai-nilai tersebut diambil dari klaster DB sumber.

   Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB tidak terenkripsi di Wilayah us-west-2. Tindakan ini dipanggil di Wilayah us-east-1.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
```

```
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB terenkripsi di Wilayah us-west-2. Tindakan ini dipanggil di Wilayah us-east-1.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

Untuk replikasi lintas wilayah antara Wilayah AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat), di mana cluster DB yang diidentifikasi oleh `ReplicationSourceIdentifier` dienkripsi, juga tentukan parameternya. `PreSignedUrl` Presigned URL harus merupakan permintaan yang valid untuk `CreateDBCluster` API operasi yang dapat dilakukan di sumber Wilayah AWS yang berisi cluster DB terenkripsi untuk direplikasi. Pengidentifikasi KMS kunci digunakan untuk mengenkripsi replika baca, dan harus menjadi KMS kunci yang valid untuk tujuan. Wilayah AWS Untuk secara otomatis daripada secara manual menghasilkan presignedURL, gunakan AWS CLI create-db-clusterperintah dengan `--source-region` opsi sebagai gantinya.

2. Periksa apakah cluster DB telah tersedia untuk digunakan dengan menggunakan escribeDBClusters operasi RDS API D, seperti yang ditunjukkan pada contoh berikut.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

Saat hasil `DescribeDBClusters` menunjukkan status dari `available`, buat instans primer untuk klaster DB sehingga replikasi dapat dimulai. Untuk melakukannya, gunakan reateDBInstance tindakan RDS API C seperti yang ditunjukkan pada contoh berikut.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r5.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
```

```
&X-Amz-Date=20160201T003808Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

Ketika instans DB dibuat dan tersedia, replikasi akan dimulai. Anda dapat menentukan apakah instans DB tersedia dengan memanggil escribeDBInstances perintah AWS CLI D.

Melihat Amazon Aurora Replika Lintas wilayah Saya SQL

Anda dapat melihat hubungan replikasi lintas wilayah untuk klaster Amazon Aurora SQL My DB Anda dengan memanggil describe-db-clustersAWS CLI perintah atau operasi D. escribeDBClusters RDS API Dalam respons, lihat bidang `ReadReplicaIdentifiers` untuk menemukan pengidentifikasi klaster DB milik setiap klaster DB replika baca lintas Wilayah. Lihat `ReplicationSourceIdentifier` elemen untuk cluster DB sumber yang merupakan sumber replikasi. ARN

## Mempromosikan replika baca ke cluster DB untuk Aurora My SQL

Anda dapat mempromosikan replika SQL baca Aurora My ke cluster DB mandiri. Saat Anda mempromosikan replika SQL baca Aurora My, instance DB-nya di-boot ulang sebelum tersedia.

Biasanya, Anda mempromosikan replika SQL baca Aurora My ke cluster DB mandiri sebagai skema pemulihan data jika cluster DB sumber gagal.

Untuk melakukannya, pertama-tama buat replika baca lalu pantau klaster DB sumber untuk mengetahui kegagalan. Jika terjadi kegagalan, lakukan hal berikut:

1. Promosikan replika baca.
2. Arahkan lalu lintas basis data ke klaster DB yang dipromosikan.
3. Buat replika baca pengganti dengan klaster DB yang dipromosikan sebagai sumbernya.

Saat Anda mempromosikan replika baca, replika baca tersebut menjadi klaster DB Aurora mandiri. Proses promosi dapat memakan waktu beberapa menit atau lebih lama, tergantung dari ukuran replika baca. Setelah Anda mempromosikan replika baca menjadi klaster DB baru, replika tersebut akan sama seperti klaster DB lainnya. Misalnya, Anda dapat membuat replika baca darinya dan melakukan operasi point-in-time pemulihan. Anda juga dapat membuat Replika Aurora untuk klaster DB.

Karena klaster DB yang dipromosikan bukan lagi merupakan replika baca, Anda tidak dapat menggunakannya sebagai target replikasi.

Langkah-langkah berikut menunjukkan proses umum untuk mempromosikan replika baca menjadi klaster DB:

1. Hentikan penulisan transaksi apa pun ke klaster DB sumber replika baca, lalu tunggu semua pembaruan yang akan dilakukan ke replika baca. Pembaruan basis data terjadi pada replika baca setelah pembaruan terjadi pada klaster DB sumber, dan lag replikasi ini dapat bervariasi secara signifikan. Gunakan metrik `ReplicaLag` untuk menentukan saat semua pembaruan sudah dilakukan pada replika baca. Metrik `ReplicaLag` mencatat jumlah waktu lag instans DB replika di belakang instans DB sumber. Saat metrik `ReplicaLag` mencapai `0`, replika baca telah menyamai instans DB sumber.
2. Promosikan replika baca dengan menggunakan opsi Promosikan di RDS konsol Amazon, AWS CLI perintah promote-read-replica-db-cluster, atau operasi PromoteReadReplicaDBClusterAmazon RDSAPI.

   Anda memilih instans Aurora My SQL DB untuk mempromosikan replika baca. Setelah replika baca dipromosikan, cluster Aurora SQL My DB dipromosikan ke cluster DB mandiri. Instans DB dengan prioritas failover tertinggi akan dipromosikan menjadi instans DB primer untuk klaster DB. Instans DB lainnya menjadi Replika Aurora.

   > **Note**
   >
   > Proses promosi memakan waktu beberapa menit. Saat Anda mempromosikan replika baca, replikasi dihentikan dan instans DB di-boot ulang. Saat boot ulang selesai, replika baca tersedia sebagai klaster DB baru.

Konsol

Untuk mempromosikan replika SQL baca Aurora My ke cluster DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pada konsol, pilih Instans.

   Panel Instans akan muncul.

3. Dalam panel Instans, pilih replika baca yang ingin Anda promosikan.

   Replika baca muncul sebagai instance Aurora SQL My DB.
4. Untuk Tindakan, pilih Promosikan replika baca.
5. Pada halaman konfirmasi, pilih Promosikan replika baca.

AWS CLI

Untuk mempromosikan replika baca ke cluster DB, gunakan perintah AWS CLI promote-read-replica-db-cluster.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

Untuk Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

RDS API

Untuk mempromosikan replika baca ke cluster DB, hubungi PromoteReadReplicaDBCluster.

## Memecahkan masalah Replika lintas wilayah untuk Amazon Aurora My SQL

Di bagian berikut ini, Anda dapat menemukan daftar pesan kesalahan umum yang mungkin Anda temukan saat membuat replika baca lintas Wilayah Amazon Aurora, dan cara menyelesaikan kesalahan yang ditentukan.

Cluster sumber [DB clusterARN] tidak mengaktifkan binlog

Untuk mengatasi masalah ini, aktifkan pencatatan log biner pada klaster DB sumber. Untuk informasi selengkapnya, lihat Sebelum Anda memulai.

Cluster sumber [DB clusterARN] tidak memiliki grup parameter cluster yang disinkronkan pada penulis

Anda akan menerima kesalahan ini jika Anda telah memperbarui parameter klaster DB `binlog_format`, tetapi belum mem-boot ulang instans primer untuk klaster DB. Boot ulang instans primer (yaitu penulis) untuk klaster DB lalu coba lagi.

Cluster sumber [DB clusterARN] sudah memiliki replika baca di wilayah ini

Anda dapat memiliki hingga lima klaster DB lintas Wilayah yang berupa replika baca untuk setiap klaster DB sumber di Wilayah AWS mana pun. Jika Anda sudah memiliki jumlah maksimum replika baca pada klaster DB di Wilayah AWS tertentu, Anda harus menghapus yang sudah ada sebelum Anda dapat membuat klaster DB lintas Wilayah baru di Wilayah tersebut.

Cluster DB [DB clusterARN] memerlukan upgrade mesin database untuk dukungan replikasi lintas wilayah

Untuk mengatasi masalah ini, tingkatkan versi mesin basis data pada semua instans dalam klaster DB sumber ke versi mesin basis data terbaru, lalu coba buat lagi DB replika baca lintas Wilayah.

## Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner)

Karena Amazon Aurora My SQL kompatibel dengan MySQL, Anda dapat mengatur replikasi antara SQL database Saya dan cluster Amazon Aurora My DB. SQL Jenis replikasi ini menggunakan replikasi log SQL biner Saya, juga disebut sebagai replikasi binlog. Jika Anda menggunakan replikasi log biner dengan Aurora, sebaiknya database SQL Saya menjalankan SQL My version 5.5 atau yang lebih baru. Anda dapat mengatur replikasi di mana cluster Aurora SQL My DB Anda adalah sumber replikasi atau replika. Anda dapat mereplikasi dengan instans Amazon RDS My SQL DB, SQL database Saya di luar AmazonRDS, atau cluster Aurora SQL My DB lainnya.

> **Note**
>
> Anda tidak dapat menggunakan replikasi binlog ke atau dari jenis klaster Aurora DB tertentu. Secara khusus, replikasi binlog tidak tersedia untuk Aurora Serverless v1 klaster. Jika pernyataan `SHOW MASTER STATUS` and `SHOW SLAVE STATUS` (Aurora My SQL version 2) atau (`SHOW REPLICA STATUS`Aurora My SQL version 3) tidak mengembalikan output, periksa apakah cluster yang Anda gunakan mendukung replikasi binlog.

Anda juga dapat mereplikasi dengan instance RDS for My SQL DB atau cluster Aurora SQL My DB di cluster lain Wilayah AWS. Saat Anda melakukan replikasi di seluruh Wilayah AWS, pastikan bahwa cluster DB dan instans DB Anda dapat diakses publik. Jika cluster Aurora My SQL DB berada di subnet pribadi di AndaVPC, gunakan peering di antara VPC Wilayah AWS Untuk informasi selengkapnya, lihat Cluster DB dalam VPC diakses oleh sebuah EC2 instance di tempat yang berbeda VPC.

Jika Anda ingin mengonfigurasi replikasi antara cluster Aurora SQL My DB dan cluster Aurora SQL My DB di cluster lain Wilayah AWS, Anda dapat membuat cluster Aurora My SQL DB sebagai replika baca di tempat yang berbeda Wilayah AWS dari cluster DB sumber. Untuk informasi selengkapnya, lihat Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS.

Dengan Aurora My SQL versi 2 dan 3, Anda dapat mereplikasi antara Aurora My SQL dan sumber eksternal atau target yang menggunakan pengidentifikasi transaksi global () untuk replikasi. GTIDs Pastikan bahwa parameter GTID -related di cluster Aurora SQL My DB memiliki pengaturan yang kompatibel dengan GTID status database eksternal. Untuk mempelajari cara melakukannya, lihat Menggunakan replikasi GTID berbasis. Di Aurora My SQL versi 3.01 dan yang lebih tinggi, Anda dapat memilih GTIDs cara menetapkan transaksi yang direplikasi dari sumber yang tidak digunakan. GTIDs Untuk informasi tentang prosedur tersimpan yang mengontrol pengaturan tersebut, lihat mysql.rds_assign_gtids_to_anonymous_transactions (Aurora Versi saya 3) SQL.

> **Warning**
>
> Saat Anda mereplikasi antara Aurora SQL My dan SQL My, pastikan Anda hanya menggunakan tabel InnoDB. Jika Anda memiliki ISAM tabel Saya yang ingin Anda replikasi, Anda dapat mengonversinya ke InnoDB sebelum mengatur replikasi dengan perintah berikut.
>
> ```
> alter table <schema>.<table_name> engine=innodb, algorithm=copy;
> ```

Di bagian berikut, siapkan replikasi, hentikan replikasi, skala pembacaan untuk database Anda, optimalkan replikasi binlog, dan siapkan binlog yang disempurnakan.

Topik

- Menyiapkan replikasi log biner untuk Aurora My SQL
- Menghentikan replikasi log biner untuk Aurora My SQL
- Penskalaan bacaan untuk SQL database Saya dengan Amazon Aurora

- Mengoptimalkan replikasi log biner untuk Aurora MySQL
- Menyiapkan binlog yang disempurnakan untuk Aurora MySQL

## Menyiapkan replikasi log biner untuk Aurora My SQL

Menyiapkan SQL Replikasi Saya dengan Aurora SQL My melibatkan langkah-langkah berikut, yang dibahas secara rinci:

Daftar Isi


- 1. Aktifkan pencatatan log biner pada sumber replikasi
- 2. Pertahankan log biner pada sumber replikasi hingga tidak diperlukan lagi
- 3. Buat salinan atau dump sumber replikasi Anda
- 4. Muat dump ke target replika Anda (jika diperlukan)
- 5. Buat pengguna replikasi pada sumber replikasi Anda
- 6. Aktifkan replikasi pada target replika Anda
  - Mengatur lokasi untuk menghentikan replikasi ke replika baca
- 7. Pantau replika Anda
- Menyinkronkan kata sandi antara sumber dan target replikasi


### 1. Aktifkan pencatatan log biner pada sumber replikasi

Temukan petunjuk tentang cara mengaktifkan pencatatan log biner pada sumber replikasi untuk mesin basis data Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk mengaktifkan logging biner pada cluster Aurora My DB SQL<br><br>Atur parameter klaster DB `binlog_format` ke `ROW`, `STATEMENT` , atau `MIXED`. `MIXED` direkomendasikan kecuali jika Anda membutuhkan format binlog tertentu. (Nilai default-nya adalah `OFF`.) |

| Mesin basis data | Petunjuk |
| --- | --- |
| | Untuk mengubah parameter `binlog_format` , buat grup parameter klaster DB kustom dan kaitkan grup parameter kustom tersebut dengan klaster DB Anda. Anda tidak dapat mengubah parameter dalam grup parameter klaster DB default.<br><br>Jika Anda mengubah parameter `binlog_format` dari `OFF` ke nilai lainnya, boot ulang klaster DB Aurora Anda agar perubahan dapat diterapkan.<br><br>Untuk informasi selengkapnya, silakan lihat Parameter klaster DB dan instans DB Amazon Aurora dan . |
| RDSuntuk saya SQL | Untuk mengaktifkan logging biner pada instans Amazon RDS DB<br><br>Anda tidak dapat mengaktifkan pencatatan biner secara langsung untuk instans Amazon RDS DB, tetapi Anda dapat mengaktifkannya dengan melakukan salah satu hal berikut:<br><br>• Aktifkan cadangan otomatis untuk instans DB. Anda dapat mengaktifkan cadangan otomatis saat Anda membuat instans DB, atau Anda dapat mengaktifkan cadangan dengan memodifikasi instans DB yang sudah ada. Untuk informasi selengkapnya, lihat Membuat instans DB di Panduan RDS Pengguna Amazon.<br>• Buat replika baca untuk instans DB. Untuk informasi selengkapnya, lihat Bekerja dengan replika baca di Panduan RDS Pengguna Amazon. |

| Mesin basis data | Petunjuk |
| --- | --- |
| Saya SQL (eksterna l) | Untuk mengatur replikasi terenkripsi<br><br>Untuk mereplikasi data dengan aman dengan Aurora My SQL versi 2, Anda dapat menggunakan replikasi terenkripsi.<br><br>**Note**<br>Jika Anda tidak perlu menggunakan replikasi terenkripsi, Anda dapat melewati langkah-langkah ini.<br><br>Berikut ini adalah prasyarat untuk menggunakan replikasi terenkripsi:<br><br>• Secure Sockets Layer (SSL) harus diaktifkan pada database SQL sumber saya eksternal.<br>• Kunci klien dan sertifikat klien harus disiapkan untuk cluster Aurora My SQL DB.<br><br>Selama replikasi terenkripsi, cluster Aurora My SQL DB bertindak sebagai klien ke server database Saya. SQL Sertifikat dan kunci untuk Aurora SQL Klien saya ada dalam file dalam format.pem.<br><br>1. Pastikan bahwa Anda siap untuk replikasi terenkripsi:<br><br>• Jika Anda belum SSL mengaktifkan database SQL sumber saya eksternal dan tidak memiliki kunci klien dan sertifikat klien yang disiapkan, aktifkan SSL server SQL database Saya dan buat kunci klien dan sertifikat klien yang diperlukan.<br>• Jika SSL diaktifkan pada sumber eksternal, berikan kunci klien dan sertifika t untuk klaster Aurora My SQL DB. Jika Anda tidak memilikinya, buat kunci dan sertifikat baru untuk cluster Aurora My SQL DB. Untuk menandatangani sertifikat klien, Anda harus memiliki kunci otoritas sertifikat yang Anda gunakan untuk mengkonfigurasi SSL pada database SQL sumber saya eksternal. |

| Mesin basis data | Petunjuk |
| --- | --- |

Untuk informasi selengkapnya, lihat Membuat SSL sertifikat dan kunci menggunakan openssl di dokumentasi SayaSQL.

Anda memerlukan sertifikat otoritas sertifikat, kunci klien, dan sertifikat klien.

2. Connect ke cluster Aurora My SQL DB sebagai pengguna master menggunakan. SSL

   Untuk informasi tentang menghubungkan ke cluster Aurora My SQL DB denganSSL, lihat. Koneksi TLS ke cluster DB MySQL Aurora

3. Jalankan prosedur `mysql.rds_import_binlog_ssl_material` tersimpan untuk mengimpor SSL informasi ke cluster Aurora My SQL DB.

   Untuk `ssl_material_value` parameter, masukkan informasi dari file format.pem untuk cluster Aurora My SQL DB di payload yang benar. JSON

   Contoh berikut mengimpor SSL informasi ke dalam cluster Aurora SQL My DB. Dalam file berformat .pem, kode konten biasanya lebih panjang dari kode konten yang ditunjukkan dalam contoh.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA
TE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
```

| Mesin basis data | Petunjuk |
| --- | --- |

```
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE
 KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc
jqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF
JuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ
tjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

Untuk informasi selengkapnya, silakan lihat mysql.rds_import_binlog_ssl_material dan Koneksi TLS ke cluster DB MySQL Aurora.

> **Note**
>
> Setelah menjalankan prosedur ini, rahasia disimpan dalam file. Untuk menghapus file nanti, Anda dapat menjalankan prosedur tersimpan mysql.rds_remove_binlog_ssl_material.

Untuk mengaktifkan logging biner pada SQL database saya eksternal

1. Dari shell perintah, hentikan layanan mysql.

```
sudo service mysqld stop
```

2. Edit file `my.cnf` (file ini biasanya ada di bawah `/etc`).

| Mesin basis data | Petunjuk |
| --- | --- |

```
sudo vi /etc/my.cnf
```

Tambahkan opsi `log_bin` dan `server_id` ke bagian `[mysqld]`. Opsi `log_bin` menyediakan sebuah pengidentifikasi nama file untuk file log biner. Opsi `server_id` menyediakan pengidentifikasi unik untuk server dalam hubungan sumber-replika.

Jika replikasi terenkripsi tidak diperlukan, pastikan SQL database Saya eksternal dimulai dengan binlog diaktifkan dan SSL dimatikan.

Berikut ini adalah entri yang relevan dalam file `/etc/my.cnf` untuk data yang tidak terenkripsi.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

Jika replikasi terenkripsi diperlukan, pastikan SQL database Saya eksternal dimulai SSL dan binlog diaktifkan.

Entri dalam `/etc/my.cnf` file termasuk lokasi file.pem untuk server database SayaSQL.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

| Mesin basis data | Petunjuk |
| --- | --- |

Selain itu, `sql_mode` opsi untuk instans My SQL DB Anda harus disetel ke 0, atau tidak boleh disertakan dalam file.cnf saya.

Saat terhubung ke SQL database saya eksternal, catat posisi log biner SQL database saya eksternal.

```
mysql> SHOW MASTER STATUS;
```

Output Anda harus seperti yang berikut ini:

```
+------------------+----------+--------------+------------------
+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000031 |      107 |              |                  |
               |
+------------------+----------+--------------+------------------
+-------------------+
1 row in set (0.00 sec)
```

Untuk informasi selengkapnya, silakan lihat Setting the replication source configuration di SQL Dokumentasi saya.

3. Mulai layanan mysql.

```
sudo service mysqld start
```

2. Pertahankan log biner pada sumber replikasi hingga tidak diperlukan lagi

Saat Anda menggunakan replikasi log SQL biner saya, Amazon RDS tidak mengelola proses replikasi. Akibatnya, Anda harus memastikan bahwa file binlog pada sumber replikasi Anda

dipertahankan hingga perubahan sudah diterapkan ke replika. Dengan mempertahankannya, Anda akan dapat memulihkan basis data sumber Anda jika terjadi kegagalan.

Gunakan petunjuk berikut untuk mempertahankan log biner untuk mesin basis data Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk menyimpan log biner pada cluster Aurora My DB SQL<br><br>Anda tidak memiliki akses ke file binlog untuk cluster Aurora SQL My DB. Akibatnya , Anda harus memilih kerangka waktu untuk menyimpan file binlog pada sumber replikasi Anda cukup lama untuk memastikan bahwa perubahan telah diterapkan ke replika Anda sebelum file binlog dihapus oleh Amazon. RDS Anda dapat menyimpan file binlog di cluster Aurora SQL My DB hingga 90 hari.<br><br>Jika Anda menyiapkan replikasi dengan SQL database Saya atau RDS untuk instance My SQL DB sebagai replika, dan database tempat Anda membuat replika sangat besar, pilih kerangka waktu yang besar untuk menyimpan file binlog hingga salinan awal database ke replika selesai dan lag replika telah mencapai 0.<br><br>Untuk mengatur rentang waktu retensi log biner, gunakan prosedur mysql.rds _set_configuration dan tentukan parameter konfigurasi `'binlog retention hours'` bersama dengan jumlah jam untuk mempertahankan file binlog di klaster DB. Nilai maksimum untuk Aurora My SQL versi 2.11.0 dan lebih tinggi dan versi 3 adalah 2160 (90 hari).<br><br>Contoh berikut menetapkan periode retensi untuk file binlog menjadi 6 hari:<br><br>`CALL mysql.rds_set_configuration('binlog retention hours', 144);`<br><br>Setelah replikasi dimulai, Anda dapat memverifikasi bahwa perubahan telah diterapka n ke replika Anda dengan menjalankan perintah (`SHOW SLAVE STATUS`Aurora My SQL version 2) atau (`SHOW REPLICA STATUS`Aurora My SQL version 3) pada replika Anda dan memeriksa bidangnya. `Seconds behind master` Jika bidang `Seconds behind master` adalah 0, maka tidak ada lag replika. Jika tidak ada lag |

| Mesin basis data | Petunjuk |
| --- | --- |
| | replika, kurangi durasi waktu retensi file binlog dengan mengatur parameter konfigurasi `binlog retention hours` ke rentang waktu yang lebih kecil.<br><br>Jika pengaturan ini tidak ditentukan, default untuk Aurora My SQL adalah 24 (1 hari).<br><br>Jika Anda menentukan nilai untuk `'binlog retention hours'` itu lebih tinggi dari nilai maksimum, maka Aurora My SQL menggunakan maksimum. |
| RDSuntuk saya SQL | Untuk mempertahankan log biner pada instans Amazon RDS DB<br><br>Anda dapat menyimpan file log biner pada instans Amazon RDS DB dengan menyetel jam retensi binlog seperti halnya dengan cluster Aurora SQL My DB, yang dijelaskan di baris sebelumnya.<br><br>Anda juga dapat menyimpan file binlog pada instans Amazon RDS DB dengan membuat replika baca untuk instans DB. Replika baca ini bersifat sementara dan hanya untuk mempertahankan file binlog. Setelah replika baca dibuat, panggil prosedur mysql.rds_stop_replication pada replika baca. Saat replikasi dihentikan, Amazon RDS tidak menghapus file binlog apa pun di sumber replikasi. Setelah Anda mengatur replikasi dengan replika permanen, Anda dapat menghapus replika baca saat lag replika (bidang `Seconds behind master`) antara sumber replikasi dan replika permanen Anda mencapai 0. |
| Saya SQL (eksternal) | Untuk mempertahankan log biner pada SQL database saya eksternal<br><br>Karena file binlog di SQL database Saya eksternal tidak dikelola oleh AmazonRDS, file tersebut disimpan hingga Anda menghapusnya.<br><br>Setelah replikasi dimulai, Anda dapat memverifikasi bahwa perubahan telah diterapkan ke replika Anda dengan menjalankan perintah (`SHOW SLAVE STATUS`Aurora My SQL version 2) atau (`SHOW REPLICA STATUS`Aurora My SQL version 3) pada replika Anda dan memeriksa bidangnya. `Seconds behind master` Jika bidang `Seconds behind master` adalah 0, maka tidak ada lag replika. Saat tidak ada lag replika, Anda dapat menghapus file binlog lama. |

3. Buat salinan atau dump sumber replikasi Anda

Anda menggunakan snapshot, klon, atau dump sumber replikasi Anda untuk memuat salinan dasar data Anda ke replika Anda. Kemudian Anda mulai mereplikasi dari titik itu.

Gunakan petunjuk berikut untuk membuat salinan atau dump sumber replikasi untuk mesin database Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk membuat salinan cluster Aurora My DB SQL<br><br>Pilih salah satu metode berikut:<br><br>• Pulihkan dari snapshot cluster DB:<br>1. Buat snapshot klaster DB untuk klaster DB Amazon Aurora Anda. Untuk informasi selengkapnya, lihat Membuat snapshot klaster DB.<br>2. Buat klaster DB Aurora baru dengan memulihkan dari snapshot klaster DB yang baru saja Anda buat.<br><br>Pastikan untuk mempertahankan grup parameter DB yang sama untuk klaster DB yang Anda pulihkan sebagai klaster DB asli Anda. Tindakan ini akan memastikan bahwa salinan klaster DB Anda memiliki pencatatan log biner aktif. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.<br>• Kloning cluster DB Anda. Untuk informasi selengkapnya, lihat Mengkloning volume untuk klaster DB Amazon Aurora.<br><br>Untuk menentukan nama dan posisi file binlog<br><br>Pilih salah satu metode berikut:<br><br>• Dalam AWS Management Console:<br>1. Pilih Database, lalu pilih instance utama (penulis) untuk cluster Aurora DB baru Anda untuk menampilkan detailnya.<br>2. Gulir ke Peristiwa Terbaru. Akan muncul pesan peristiwa yang menyertakan nama dan posisi file binlog. Pesan peristiwa memiliki format seperti berikut ini. |

| Mesin basis data | Petunjuk |
| --- | --- |

```
Binlog position from crash recovery is binlog-file-name  binlog-po
sition
```

3. Simpan nilai nama dan posisi file binlog saat Anda memulai replikasi.

- Panggil AWS CLI perintah describe-events, seperti pada contoh berikut.

```
aws rds describe-events

{
    "Events": [
        {
            "EventCategories": [],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:
db:sample-restored-instance",
            "Date": "2016-10-28T19:43:46.862Z",
            "Message": "Binlog position from crash recovery is mysql-
bin-changelog.000003 4278",
            "SourceIdentifier": "sample-restored-instance"
        }
    ]
}
```

- Periksa log SQL kesalahan saya untuk posisi file SQL binlog Saya terakhir.

Untuk membuat dump cluster Aurora SQL My DB

Jika target replika Anda adalah SQL database Saya eksternal atau RDS untuk instance My SQL DB, maka Anda harus membuat file dump dari cluster Aurora DB Anda.

Pastikan untuk menjalankan `mysqldump` perintah terhadap salinan cluster DB sumber Anda yang Anda buat. Ini untuk menghindari pertimbangan penguncian saat mengambil tempat pembuangan sampah. Jika dump diambil pada cluster DB sumber secara langsung, akan perlu untuk mengunci tabel sumber untuk mencegah penulisan bersamaan kepada mereka saat dump sedang berlangsung.

| Mesin basis data | Petunjuk |
| --- | --- |
| | 1. Connect ke cluster DB Anda menggunakan SQL klien Saya.<br>2. Keluarkan `mysqldump` perintah. Sebagai contoh:<br>`PROMPT> mysqldump --databases database_name --single-transaction --order-by-primary -r backup.sql -u local_user s -p`<br>3. Setelah Anda membuat file dump, Anda dapat menghapus salinan cluster DB. |
| RDSuntuk saya SQL | Untuk membuat snapshot dari instans Amazon RDS DB<br><br>Buat replika baca instans Amazon RDS DB Anda. Untuk informasi selengkapnya, lihat Membuat replika baca dalam Panduan Pengguna Amazon Relational Database Service.<br><br>1. Hubungkan ke replika baca Anda dan hentikan replikasi dengan menjalankan prosedur mysql.rds_stop_replication.<br>2. Sementara replika baca Dihentikan, Connect ke replika baca dan jalankan perintah (`SHOW SLAVE STATUS`Aurora My SQL version 2) atau (`SHOW REPLICA STATUS`Aurora My version 3). SQL Ambil nama file log biner saat ini dari bidang `Relay_Master_Log_File` dan posisi file log dari bidang `Exec_Master_Log_Pos` . Simpan nilai-nilai ini saat Anda memulai replikasi.<br>3. Jika replika baca tetap Dihentikan, buat snapshot DB replika baca. Untuk informasi selengkapnya, lihat Membuat snapshot DB dalam Panduan Pengguna Amazon Relational Database Service.<br>4. Hapus replika baca. |

| Mesin basis data | Petunjuk |
| --- | --- |
| Saya SQL (eksternal) | Untuk membuat dump database eksternal Saya SQL<br>1. Sebelum Anda membuat dump, Anda harus memastikan bahwa lokasi binlog untuk dump sama barunya dengan data dalam instans sumber Anda. Untuk melakukan hal ini, Anda harus terlebih dahulu menghentikan operasi tulis apa pun ke instans dengan perintah berikut:<br>`mysql> FLUSH TABLES WITH READ LOCK;`<br>2. Buat dump SQL database Saya menggunakan `mysqldump` perintah seperti yang ditunjukkan berikut:<br>`PROMPT> sudo mysqldump --databases database_name --master-data=2 --single-transaction \ --order-by-primary -r backup.sql -u local_user -p`<br>3. Setelah Anda membuat dump, buka kunci tabel di SQL database Saya dengan perintah berikut:<br>`mysql> UNLOCK TABLES;` |

4. Muat dump ke target replika Anda (jika diperlukan)

Jika Anda berencana untuk memuat data dari dump SQL database Saya yang berada di luar AmazonRDS, Anda mungkin ingin membuat EC2 instance untuk menyalin file dump ke. Kemudian Anda dapat memuat data ke cluster DB atau instance DB Anda dari EC2 instance itu. Dengan menggunakan pendekatan ini, Anda dapat mengompres file dump sebelum menyalinnya ke EC2 instance untuk mengurangi biaya jaringan yang terkait dengan menyalin data ke Amazon. RDS Anda juga dapat mengenkripsi file dump untuk mengamankan data saat data tersebut ditransfer melewati jaringan.

> **Note**
>
> Jika Anda membuat cluster Aurora My SQL DB baru sebagai target replika Anda, maka Anda tidak perlu memuat file dump:
>
> - Anda dapat memulihkan dari snapshot cluster DB untuk membuat cluster DB baru. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.
> - Anda dapat mengkloning cluster DB sumber Anda untuk membuat cluster DB baru. Untuk informasi selengkapnya, lihat Mengkloning volume untuk klaster DB Amazon Aurora.
> - Anda dapat memigrasikan data dari snapshot instans DB ke cluster DB baru. Untuk informasi selengkapnya, lihat Migrasi data ke cluster Amazon Aurora My DB SQL.

Gunakan instruksi berikut untuk memuat dump sumber replikasi Anda ke target replika Anda untuk mesin database Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk memuat dump ke cluster Aurora SQL My DB<br>1. Salin output `mysqldump` perintah dari sumber replikasi Anda ke lokasi yang juga dapat terhubung ke cluster Aurora SQL My DB Anda.<br>2. Connect ke cluster Aurora My SQL DB Anda menggunakan perintah. `mysql` Berikut adalah contohnya.<br>`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`<br>3. Pada `mysql` prompt, jalankan `source` perintah dan berikan nama file dump database Anda untuk memuat data ke cluster Aurora SQL My DB, misalnya:<br>`mysql> source backup.sql;` |
| RDSuntuk saya SQL | Untuk memuat dump ke instans Amazon RDS DB<br>1. Salin output `mysqldump` perintah dari sumber replikasi Anda ke lokasi yang juga dapat terhubung ke instans My SQL DB Anda. |

| Mesin basis data | Petunjuk |
| --- | --- |
| | 2. Connect ke instans My SQL DB Anda menggunakan `mysql` perintah. Berikut adalah contohnya.<br>`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`<br>3. Pada `mysql` prompt, jalankan `source` perintah dan berikan nama file dump database Anda untuk memuat data ke instance My SQL DB, misalnya:<br>`mysql> source backup.sql;` |
| Saya SQL (eksternal) | Untuk memuat dump ke database saya SQL eksternal<br>Anda tidak dapat memuat snapshot DB atau snapshot cluster DB ke database Saya SQL eksternal. Sebagai gantinya, Anda harus menggunakan output dari perintah `mysqldump` .<br>1. Salin output `mysqldump` perintah dari sumber replikasi Anda ke lokasi yang juga dapat terhubung ke SQL database Saya.<br>2. Connect ke SQL database Saya menggunakan `mysql` perintah. Berikut adalah contohnya.<br>`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`<br>3. Pada `mysql` prompt, jalankan `source` perintah dan berikan nama file dump database Anda untuk memuat data ke SQL database Saya. Berikut adalah contohnya.<br>`mysql> source backup.sql;` |

5. Buat pengguna replikasi pada sumber replikasi Anda

Buat ID pengguna pada sumber yang digunakan hanya untuk replikasi. Contoh berikut adalah RDS untuk database My SQL atau eksternal My SQL source.

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

Untuk database Aurora My SQL source, parameter cluster `skip_name_resolve` DB diatur ke 1 (0N) dan tidak dapat dimodifikasi, jadi Anda harus menggunakan alamat IP untuk host alih-alih nama domain. Untuk informasi selengkapnya, lihat skip_name_resolve di dokumentasi Saya. SQL

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

Pengguna memerlukan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE`. Berikan hak akses ini kepada pengguna.

Jika Anda perlu menggunakan replikasi terenkripsi, memerlukan SSL koneksi untuk pengguna replikasi. Misalnya, Anda dapat menggunakan salah satu pernyataan berikut untuk meminta SSL koneksi pada akun penggunarepl_user.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> Jika `REQUIRE SSL` tidak disertakan, koneksi replikasi dapat kembali ke koneksi yang tidak terenkripsi tanpa peringatan.

6. Aktifkan replikasi pada target replika Anda

Sebelum Anda mengaktifkan replikasi, kami sarankan Anda mengambil snapshot manual dari cluster Aurora My SQL DB atau RDS untuk target replika instans My SQL DB. Jika masalah muncul dan Anda harus membuat ulang replikasi dengan klaster DB atau target replika instans DB, Anda dapat memulihkan klaster DB atau instans DB dari snapshot ini daripada harus mengimpor data ke dalam target replika Anda lagi.

Gunakan petunjuk berikut untuk mengaktifkan replikasi untuk mesin basis data Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk mengaktifkan replikasi dari cluster Aurora SQL My DB<br><br>1. Temukan tempat awal untuk replikasi. Anda memerlukan nama file binlog dan posisi binlog.<br><br>Jika target replika klaster DB Anda dibuat dari berikut ini:<br><br>• Snapshot atau klon cluster DB - Ambil nama dan posisi file binlog dari peristiwa terbaru untuk cluster DB baru Anda, seperti yang ditunjukkan pada. 3. Buat salinan atau dump sumber replikasi Anda<br>• Snapshot DB - Anda mengambil nama dan posisi file binlog dari perintah (`SHOW SLAVE STATUS`Aurora My SQL version 2) atau (`SHOW REPLICA STATUS`Aurora My SQL version 3) saat Anda membuat snapshot dari sumber replikasi Anda.<br><br>2. Hubungkan ke klaster DB dan panggil prosedur berikut untuk memulai replikasi dengan sumber replikasi Anda menggunakan nama dan lokasi file log biner dari langkah sebelumnya:<br><br>• mysql.rds_set_external_source ) SQL<br>• mysql.rds_set_external_master ) SQL SQL<br>• mysql.rds_start_replication (semua versi)<br><br>Contoh berikut adalah untuk Aurora My SQL versi 3.<br><br>`CALL mysql.rds_set_external_source ('mydbinstance.123456789012 .us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;`<br><br>Untuk menggunakan SSL enkripsi, tetapkan nilai akhir menjadi 1 bukan`0`. |

| Mesin basis data | Petunjuk |
| --- | --- |
| RDSuntuk saya SQL | Untuk mengaktifkan replikasi dari instans Amazon RDS DB<br><br>1. Jika target replika instans DB Anda dibuat dari snapshot DB, maka Anda memerlukan file binlog dan posisi binlog yang merupakan tempat awal untuk replikasi. Anda mengambil nilai-nilai ini dari perintah `SHOW SLAVE STATUS` (Aurora SQL My version 2) atau `SHOW REPLICA STATUS` (Aurora SQL My version 3) saat Anda membuat snapshot dari sumber replikasi Anda.<br>2. Hubungkan ke instans DB dan panggil prosedur mysql.rds_set_external_master ) SQL SQL atau mysql.rds_set_external_source ) SQL dan mysql.rds_start_replication untuk memulai replikasi dengan sumber replikasi Anda. Gunakan nama dan lokasi file log biner dari langkah sebelumnya. Berikut adalah contohnya.<br><br>`CALL mysql.rds_set_external_master ('mydbcluster.cluster-12345 6789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;`<br><br>Untuk menggunakan SSL enkripsi, tetapkan nilai akhir menjadi `1` bukan`0`. |

| Mesin basis data | Petunjuk |
| --- | --- |
| Saya SQL (eksternal) | Untuk mengaktifkan replikasi dari database Saya SQL eksternal<br><br>1. Ambil file binlog dan posisi binlog yang merupakan tempat awal untuk replikasi. Anda mengambil nilai-nilai ini dari perintah `SHOW SLAVE STATUS` (Aurora SQL My version 2) atau `SHOW REPLICA STATUS` (Aurora SQL My version 3) saat Anda membuat snapshot dari sumber replikasi Anda. Jika target SQL replika saya eksternal Anda diisi dari output `mysqldump` perintah dengan `--master-data=2` opsi, maka file binlog dan posisi binlog disertakan dalam output. Berikut adalah contohnya.<br><br>`-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;`<br><br>2. Connect ke target eksternal My SQL replica, dan issue `CHANGE MASTER TO` dan `START SLAVE` (Aurora SQL My version 2) atau `START REPLICA` (Aurora SQL My version 3) untuk memulai replikasi dengan sumber replikasi Anda menggunakan nama file log biner dan lokasi dari langkah sebelumnya, misalnya:<br><br>`CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 2 START REPLICA; -- Aurora MySQL version 3` |

Jika replikasi gagal, hal tersebut dapat mengakibatkan peningkatan besar I/O yang tidak disengaja pada replika, yang dapat menurunkan performa. Jika replikasi gagal atau tidak lagi diperlukan, Anda dapat menjalankan prosedur tersimpan mysql.rds_reset_external_master ) RDS SQL SQL atau mysql.rds_reset_external_source ) SQL SQL untuk menghapus konfigurasi replikasi.

Mengatur lokasi untuk menghentikan replikasi ke replika baca

Di Aurora My SQL versi 3.04 dan yang lebih tinggi, Anda dapat memulai replikasi dan kemudian menghentikannya di lokasi file log biner tertentu menggunakan prosedur yang disimpan. mysql.rds_start_replication_until (Aurora Versi saya 3) SQL

Untuk memulai replikasi ke replika baca dan menghentikan replikasi di lokasi tertentu

1. Menggunakan SQL klien Saya, sambungkan ke replika Aurora SQL My DB cluster sebagai pengguna utama.
2. Jalankan prosedur yang tersimpan di mysql.rds_start_replication_until (Aurora Versi saya 3) SQL.

   Contoh berikut memulai replikasi dan mereplikasi perubahan hingga mencapai lokasi `120` di file biner `mysql-bin-changelog.000777`. Dalam skenario pemulihan bencana, asumsikan bahwa lokasi `120` tepat sebelum bencana.

   ```
   call mysql.rds_start_replication_until(
     'mysql-bin-changelog.000777',
     120);
   ```

Replikasi berhenti secara otomatis ketika stop point tercapai. RDSPeristiwa berikut dihasilkan:`Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure`.

Jika Anda menggunakan replikasi GTID berbasis, gunakan prosedur yang mysql.rds_start_replication_until_gtid (Aurora Versi saya 3) SQL disimpan alih-alih prosedur yang mysql.rds_start_replication_until (Aurora Versi saya 3) SQL disimpan. Untuk informasi lebih lanjut tentang replikasi GTID berbasis, lihatMenggunakan replikasi GTID berbasis.

7. Pantau replika Anda

Saat Anda mengatur SQL replikasi Saya dengan cluster Aurora SQL My DB, Anda harus memantau peristiwa failover untuk cluster Aurora SQL My DB ketika itu adalah target replika. Jika terjadi failover,

maka klaster DB yang merupakan target replika Anda dapat dibuat ulang pada host baru dengan alamat jaringan yang berbeda. Untuk informasi tentang cara memantau peristiwa failover, lihat Bekerja dengan pemberitahuan RDS acara Amazon.

Anda juga dapat memantau seberapa jauh target replika berada di belakang sumber replikasi dengan menghubungkan ke target replika dan menjalankan perintah (`SHOW SLAVE STATUS`Aurora My version 2) atau (`SHOW REPLICA STATUS`Aurora My SQL version 3). SQL Dalam output perintah, bidang `Seconds Behind Master` akan memberi tahu Anda seberapa jauh target replika tertinggal di belakang sumber.

> **Important**
>
> Jika Anda memutakhirkan cluster DB dan menentukan grup parameter khusus, pastikan untuk me-reboot cluster secara manual setelah pemutakhiran selesai. Melakukannya membuat cluster menggunakan pengaturan parameter kustom baru Anda, dan memulai ulang replikasi binlog.

Menyinkronkan kata sandi antara sumber dan target replikasi

Saat Anda mengubah akun pengguna dan kata sandi pada sumber replikasi menggunakan SQL pernyataan, perubahan tersebut direplikasi ke target replikasi secara otomatis.

Jika Anda menggunakan AWS Management Console, the AWS CLI, atau RDS API untuk mengubah kata sandi utama pada sumber replikasi, perubahan tersebut tidak secara otomatis direplikasi ke target replikasi. Jika Anda ingin menyinkronkan pengguna master dan kata sandi master antara sistem sumber dan target, Anda harus membuat sendiri perubahan yang sama pada target replikasi.

## Menghentikan replikasi log biner untuk Aurora My SQL

Untuk menghentikan replikasi log biner dengan instans My SQL DB, SQL database eksternal Saya, atau cluster Aurora DB lainnya, ikuti langkah-langkah ini, dibahas secara rinci berikut dalam topik ini.

1. Hentikan replikasi log biner pada target replika

2. Nonaktifkan pencatatan log biner pada sumber replikasi

1. Hentikan replikasi log biner pada target replika

Gunakan petunjuk berikut untuk menghentikan replikasi log biner untuk mesin basis data Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk menghentikan replikasi log biner pada target replika cluster Aurora SQL My DB<br><br>Hubungkan ke klaster DB Aurora yang merupakan target replika, dan panggil prosedur mysql.rds_stop_replication. |
| RDSuntuk saya SQL | Untuk menghentikan replikasi log biner pada instans Amazon RDS DB<br><br>Connect ke instans RDS DB yang merupakan target replika dan panggil mysql.rds_stop_replication prosedurnya. |
| Saya SQL (eksternal) | Untuk menghentikan replikasi log biner pada database saya SQL eksternal<br><br>Connect ke SQL database Saya dan jalankan perintah `STOP SLAVE` (versi 5.7) atau `STOP REPLICA` (versi 8.0). |

2. Nonaktifkan pencatatan log biner pada sumber replikasi

Gunakan petunjuk dalam tabel berikut untuk menonaktifkan pencatatan log biner pada sumber replikasi untuk mesin basis data Anda.

| Mesin basis data | Petunjuk |
| --- | --- |
| Aurora Saya SQL | Untuk menonaktifkan pencatatan log biner pada klaster DB Amazon Aurora<br><br>1. Hubungkan ke klaster DB Aurora yang merupakan sumber replikasi.<br>2. Gunakan prosedur mysql.rds_set_configuration dan tentukan parameter konfigurasi `binlog retention hours`, dengan nilai `NULL`, seperti yang ditunjukkan pada contoh berikut.<br><br>`CALL mysql.rds_set_configuration('binlog retention hours', NULL);` |

| Mesin basis data | Petunjuk |
| --- | --- |
| | > **Note**<br>Anda tidak dapat menggunakan nilai `0` untuk `binlog retention hours`.<br><br>3. Atur parameter `binlog_format` ke `OFF` pada sumber replikasi. Parameter `binlog_format` ada di grup parameter klaster DB kustom yang terkait dengan klaster DB Anda.<br><br>Setelah Anda mengubah nilai parameter `binlog_format` , boot ulang klaster DB Anda agar perubahan dapat diterapkan.<br><br>Untuk informasi selengkapnya, silakan lihat Parameter klaster DB dan instans DB Amazon Aurora dan Memodifikasi parameter dalam grup parameter DB di . |
| RDSuntuk saya SQL | Untuk menonaktifkan pencatatan biner pada instans Amazon RDS DB<br><br>Anda tidak dapat mematikan pencatatan biner secara langsung untuk instans Amazon RDS DB, tetapi Anda dapat mematikannya dengan melakukan hal berikut:<br><br>1. Nonaktifkan cadangan otomatis untuk instans DB. Anda dapat menonaktifkan cadangan otomatis dengan memodifikasi instans DB yang sudah ada dan mengatur Periode Retensi Cadangan menjadi 0. Untuk informasi selengkapnya, lihat Memodifikasi instans Amazon RDS DB dan Bekerja dengan cadangan di Panduan Pengguna Layanan Amazon Relational Database Service.<br>2. Hapus semua replika baca untuk instans DB. Untuk informasi selengkapnya, lihat Bekerja dengan replika baca instance MariaDB, SQL My, dan SQL Postgre DB di Panduan Pengguna Layanan Amazon Relational Database Service. |

| Mesin basis data | Petunjuk |
| --- | --- |
| Saya SQL (eksternal) | Untuk mematikan logging biner pada SQL database saya eksternal<br><br>Connect ke SQL database Saya dan panggil `STOP REPLICATION` perintah.<br><br>1. Dari shell perintah, hentikan layanan `mysqld`.<br><br>```<br>sudo service mysqld stop<br>```<br><br>2. Edit file `my.cnf` (file ini biasanya ada dalam `/etc`).<br><br>```<br>sudo vi /etc/my.cnf<br>```<br><br>Hapus opsi `log_bin` dan `server_id` dari bagian `[mysqld]`.<br><br>Untuk informasi selengkapnya, silakan lihat Setting the replication source configurationdalam SQL dokumentasi saya.<br><br>3. Mulai layanan mysql.<br><br>```<br>sudo service mysqld start<br>``` |

## Penskalaan bacaan untuk SQL database Saya dengan Amazon Aurora

Anda dapat menggunakan Amazon Aurora dengan instans My SQL DB Anda untuk memanfaatkan kemampuan penskalaan baca Amazon Aurora dan memperluas beban kerja baca untuk instans DB Saya. SQL Untuk menggunakan Aurora untuk menskalakan pembacaan untuk instans My SQL DB Anda, buat klaster Amazon Aurora SQL My DB dan jadikan replika baca instans My DB Anda. SQL Ini berlaku untuk instans RDS for My SQL DB, atau SQL database Saya yang berjalan di luar AmazonRDS.

Untuk informasi tentang cara membuat klaster DB Amazon Aurora, lihat Membuat klaster DB Amazon Aurora.

Saat menyiapkan replikasi antara instans SQL DB Saya dan kluster DB Amazon Aurora, pastikan untuk mengikuti panduan berikut:

- Gunakan alamat titik akhir klaster Amazon Aurora DB saat Anda mereferensikan kluster Amazon Aurora My DB Anda. SQL Jika failover terjadi, maka Replika Aurora yang dipromosikan ke instance utama untuk cluster Aurora SQL My DB terus menggunakan alamat titik akhir cluster DB.
- Pertahankan binlog pada instans penulis Anda hingga Anda memverifikasi bahwa binlog tersebut telah diterapkan ke Replika Aurora. Dengan mempertahankannya, Anda akan dapat memulihkan instans penulis Anda jika terjadi kegagalan.

**Important**

Saat menggunakan replikasi yang dikelola sendiri, Anda bertanggung jawab untuk memantau dan menyelesaikan masalah replikasi yang mungkin terjadi. Untuk informasi selengkapnya, lihat Mendiagnosis dan mengatasi jeda di antara replika baca.

**Note**

Izin yang diperlukan untuk memulai replikasi pada klaster Aurora My SQL DB dibatasi dan tidak tersedia untuk pengguna master Amazon Anda. RDS Oleh karena itu, Anda harus menggunakan mysql.rds_set_external_master ) SQL SQL or mysql.rds_set_external_source ) SQL and mysql.rds_start_replication prosedur untuk mengatur replikasi antara cluster Aurora SQL My DB Anda dan instans My DB AndaSQL.

Mulai replikasi antara instance sumber eksternal dan cluster Aurora SQL My DB

1. Jadikan sumber Instans SQL DB saya hanya-baca:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. Jalankan `SHOW MASTER STATUS` perintah pada sumber My SQL DB instance untuk menentukan lokasi binlog. Anda akan menerima output yang serupa dengan contoh berikut:

```
File                        Position
-----------------------------------
 mysql-bin-changelog.000031      107
-----------------------------------
```

3. Salin database dari instans My SQL DB eksternal ke Amazon Aurora My SQL DB cluster menggunakan. `mysqldump` Untuk database yang sangat besar, Anda mungkin ingin menggunakan prosedur dalam Mengimpor data ke instans DB Saya atau SQL MariaDB dengan pengurangan waktu henti di Panduan Pengguna Amazon Relational Database Service.

   Untuk Linux, macOS, atau Unix:

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

   Untuk Windows:

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
        -p RDS_password
```

   > **Note**
   >
   > Pastikan tidak ada spasi antara opsi `-p` dan kata sandi yang dimasukkan.

   Gunakan opsi `--host`, `--user (-u)`, `--port`, dan `-p` dalam perintah `mysql` untuk menentukan nama host, nama pengguna, port, dan kata sandi untuk terhubung ke klaster DB

Aurora Anda. Nama host adalah DNS nama dari titik akhir cluster Amazon Aurora DB, misalnya,. `mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com` Anda dapat menemukan nilai endpoint di detail klaster di Amazon RDS Management Console.

4. Jadikan sumber Instans SQL DB saya dapat ditulis lagi:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Untuk informasi selengkapnya tentang membuat cadangan untuk digunakan dengan replikasi, lihat Backing up a source or replica by making it read onlydi dokumentasi Saya. SQL

5. Di Amazon RDS Management Console, tambahkan alamat IP server yang menghosting sumber SQL Database saya ke grup VPC keamanan untuk cluster Amazon Aurora DB. Untuk informasi selengkapnya tentang memodifikasi grup VPC keamanan, lihat Grup keamanan untuk Anda VPC di Panduan Pengguna Amazon Virtual Private Cloud.

   Anda mungkin juga perlu mengonfigurasi jaringan lokal Anda untuk mengizinkan koneksi dari alamat IP kluster Amazon Aurora DB Anda, sehingga dapat berkomunikasi dengan sumber Instans SayaSQL. Untuk menemukan alamat IP klaster DB Amazon Aurora, gunakan perintah `host`.

```
host aurora_endpoint_address
```

   Nama host adalah DNS nama dari titik akhir cluster Amazon Aurora DB.

6. Menggunakan klien pilihan Anda, sambungkan ke SQL instans Saya eksternal dan buat SQL pengguna Saya untuk digunakan untuk replikasi. Akun ini digunakan hanya untuk replikasi dan harus dibatasi pada domain Anda untuk meningkatkan keamanan. Berikut adalah contohnya.

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

7. Untuk SQL instans Saya eksternal, berikan `REPLICATION CLIENT` dan `REPLICATION SLAVE` hak istimewa kepada pengguna replikasi Anda. Misalnya, untuk memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna '`repl_user`' bagi domain Anda, jalankan perintah berikut.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com'
    IDENTIFIED BY 'password';
```

8. Ambil snapshot manual cluster Aurora SQL My DB untuk menjadi replika baca sebelum menyiapkan replikasi. Jika Anda perlu membangun kembali replikasi dengan cluster DB sebagai replika baca, Anda dapat memulihkan cluster Aurora My SQL DB dari snapshot ini alih-alih harus mengimpor data dari SQL instans My DB Anda ke cluster Aurora My DB baru. SQL
9. Jadikan klaster DB Amazon Aurora sebagai replika. Connect ke cluster Amazon Aurora DB sebagai pengguna utama dan identifikasi sumber SQL Database saya sebagai sumber replikasi dengan menggunakan mysql.rds_set_external_master ) SQL SQL atau mysql.rds_set_external_source ) SQL dan prosedur. mysql.rds_start_replication

   Gunakan nama file binlog dan posisi yang Anda tentukan di Langkah 2. Berikut adalah contohnya.

   ```
   For Aurora MySQL version 2:
   CALL mysql.rds_set_external_master ('mymasterserver.example.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

   For Aurora MySQL version 3:
   CALL mysql.rds_set_external_source ('mymasterserver.example.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
   ```
10. Pada klaster DB Amazon Aurora, panggil prosedur mysql.rds_start_replication untuk memulai replikasi.

    ```
    CALL mysql.rds_start_replication;
    ```

Setelah membuat replikasi antara instans My SQL DB sumber dan kluster DB Amazon Aurora, Anda dapat menambahkan Aurora Replicas ke cluster Amazon Aurora DB Anda. Kemudian, Anda dapat terhubung ke Replika Aurora untuk menskalakan baca data Anda. Untuk informasi tentang cara membuat Replika Aurora, lihat Menambahkan Replika Aurora ke klaster DB.

## Mengoptimalkan replikasi log biner untuk Aurora MySQL

Dari informasi berikut, Anda dapat mempelajari cara mengoptimalkan performa replikasi log biner dan memecahkan masalah terkait di Aurora MySQL.

> **Tip**
>
> Diskusi ini mengasumsikan bahwa Anda sudah mengenal mekanisme replikasi log biner MySQL dan cara kerjanya. Untuk informasi latar belakang, lihat Replication Implementation dalam dokumentasi MySQL.

### Replikasi log biner multithreaded

Dengan replikasi log biner multi-thread, thread SQL membaca peristiwa dari log relai dan mengantrekannya agar thread pekerja SQL dapat diterapkan. Thread pekerja SQL dikelola oleh thread koordinator. Peristiwa log biner diterapkan secara paralel jika memungkinkan. Tingkat paralelisme tergantung pada faktor-faktor termasuk versi, parameter, desain skema, dan karakteristik beban kerja.

Replikasi log biner multithreaded didukung di Aurora MySQL versi 3, dan di Aurora MySQL versi 2.12.1 dan lebih tinggi. Agar replika multithreaded dapat memproses peristiwa binlog secara paralel secara efisien, Anda harus mengonfigurasi sumber untuk replikasi log biner multithreaded, dan sumbernya harus menggunakan versi yang menyertakan informasi paralelisme pada file log binernya.

Ketika instance Aurora MySQL DB dikonfigurasi untuk menggunakan replikasi log biner, secara default instance replika menggunakan replikasi single-threaded untuk versi Aurora MySQL yang lebih rendah dari 3,04. Untuk mengaktifkan replikasi multithreaded, Anda memperbarui `replica_parallel_workers` parameter ke nilai yang lebih besar daripada 1 di grup parameter kustom Anda.

Untuk Aurora MySQL versi 3.04 dan lebih tinggi, replikasi multithreaded secara default, dengan disetel ke. `replica_parallel_workers` `4` Anda dapat memodifikasi parameter ini di grup parameter kustom Anda.

Untuk meningkatkan ketahanan database Anda terhadap penghentian yang tidak terduga, kami sarankan Anda mengaktifkan replikasi GTID pada sumber dan mengizinkan replika. GTIDs Untuk memungkinkan replikasi GTID, atur `gtid_mode` ke `ON_PERMISSIVE` sumber dan replika. Untuk informasi selengkapnya tentang replikasi, lihat Menggunakan replikasi GTID berbasis.

Opsi konfigurasi berikut dapat membantu Anda menyesuaikan replikasi multi-thread. Untuk informasi penggunaan, lihat Replication and Binary Logging Options and Variables dalam Panduan Referensi

MySQL. Untuk informasi selengkapnya tentang replikasi multithreaded, lihat MySQL Blog Improving the Parallel Applier with WriteSet-based Dependency Tracking.

Nilai parameter optimal tergantung pada beberapa faktor. Misalnya, performa replikasi log biner dipengaruhi oleh karakteristik beban kerja basis data Anda dan kelas instans DB tempat replika berjalan. Oleh karena itu, kami menyarankan Anda menguji secara menyeluruh semua perubahan pada parameter konfigurasi ini sebelum menerapkan pengaturan parameter baru ke instance produksi:

- `binlog_format recommended value`— diatur ke `ROW`
- `binlog_group_commit_sync_delay`
- `binlog_group_commit_sync_no_delay_count`
- `binlog_transaction_dependency_history_size`
- `binlog_transaction_dependency_tracking`Nilai yang direkomendasikan adalah `WRITESET`
- `replica_preserve_commit_order`
- `replica_parallel_type`Nilai yang direkomendasikan adalah `LOGICAL_CLOCK`
- `replica_parallel_workers`
- `replica_pending_jobs_size_max`
- `transaction_write_set_extraction`Nilai yang direkomendasikan adalah `XXHASH64`

Karakteristik skema dan beban kerja Anda adalah faktor yang memengaruhi replikasi secara paralel. Faktor yang paling umum adalah sebagai berikut.

- Tidak adanya kunci utama - RDS tidak dapat menetapkan ketergantungan writeset untuk tabel tanpa kunci primer. Dengan `ROW` format, pernyataan multi-baris tunggal dapat dicapai dengan pemindaian tabel penuh tunggal pada sumbernya, tetapi menghasilkan satu pemindaian tabel penuh per baris yang dimodifikasi pada replika. Tidak adanya kunci primer secara signifikan mengurangi throughput replikasi.
- Kehadiran kunci asing - Jika kunci asing ada, Amazon RDS tidak dapat menggunakan dependensi writeset untuk paralelisme tabel dengan hubungan FK.
- Ukuran transaksi — Jika satu transaksi mencakup puluhan atau ratusan megabyte atau gigabyte, thread koordinator dan salah satu thread pekerja mungkin menghabiskan waktu lama hanya memproses transaksi itu. Selama waktu itu, semua thread pekerja lainnya mungkin tetap menganggur setelah mereka selesai memproses transaksi mereka sebelumnya.

Di Aurora MySQL versi 3.06 dan lebih tinggi, Anda dapat meningkatkan kinerja replika log biner saat mereplikasi transaksi untuk tabel besar dengan lebih dari satu indeks sekunder. Fitur ini memperkenalkan kumpulan utas untuk menerapkan perubahan indeks sekunder secara paralel pada replika binlog. Fitur ini dikendalikan oleh parameter cluster `aurora_binlog_replication_sec_index_parallel_workers` DB, yang mengontrol jumlah total thread paralel yang tersedia untuk menerapkan perubahan indeks sekunder. Parameter diatur ke `0` (dinonaktifkan) secara default. Mengaktifkan fitur ini tidak memerlukan instance restart. Untuk mengaktifkan fitur ini, hentikan replikasi yang sedang berlangsung, atur jumlah thread parallel worker yang diinginkan, lalu mulai replikasi lagi.

Mengoptimalkan replikasi binlog

Di Aurora MySQL 2.10 dan lebih tinggi, Aurora secara otomatis menerapkan optimisasi yang dikenal sebagai cache I/O binlog ke replikasi log biner. Dengan membuat cache peristiwa binlog yang paling baru di-commit, optimisasi ini dirancang untuk meningkatkan performa thread dump binlog sambil membatasi dampak pada transaksi latar depan di instans sumber binlog.

> **Note**
>
> Memori yang digunakan untuk fitur ini tidak bergantung pada pengaturan `binlog_cache` MySQL.
> Fitur ini tidak berlaku untuk instans Aurora DB yang menggunakan kelas instans `db.t2` dan `db.t3`.

Anda tidak perlu menyesuaikan parameter konfigurasi apa pun untuk mengaktifkan optimisasi ini. Secara khusus, jika Anda telah menyesuaikan parameter konfigurasi `aurora_binlog_replication_max_yield_seconds` ke nilai bukan nol di versi MySQL Aurora sebelumnya, atur kembali ke nol untuk versi yang tersedia saat ini.

Variabel status `aurora_binlog_io_cache_reads` dan `aurora_binlog_io_cache_read_requests` membantu Anda untuk memantau seberapa sering data dibaca dari binlog I/O cache.

- `aurora_binlog_io_cache_read_requests` menunjukkan jumlah permintaan baca I/O binlog dari cache.
- `aurora_binlog_io_cache_reads`: menunjukkan jumlah baca I/O binlog yang mengambil informasi dari cache.

Kueri SQL berikut menghitung persentase permintaan baca binlog yang memanfaatkan informasi cache. Dalam hal ini, makin dekat rasionya ke 100, makin baik.

```
mysql> SELECT
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

Fitur cache I/O binlog juga menyertakan metrik baru yang terkait dengan thread dump binlog. Thread dump adalah thread yang dibuat saat replika binlog baru terhubung ke instans sumber binlog.

Metrik thread dump dicetak ke log basis data setiap 60 detik dengan awalan `[Dump thread metrics]`. Metrik ini mencakup informasi untuk setiap replika binlog seperti `Secondary_id`, `Secondary_uuid`, nama file binlog, dan posisi yang dibaca setiap replika. Metrik ini juga mencakup `Bytes_behind_primary` yang merepresentasikan jarak dalam byte antara sumber replikasi dan replika. Metrik ini mengukur lag thread I/O replika. Angka tersebut berbeda dengan lag thread pengaplikasi SQL replika, yang direpresentasikan oleh metrik `seconds_behind_master` pada replika binlog. Anda dapat menentukan apakah replika binlog mengimbangi atau tertinggal di belakang sumber dengan memeriksa apakah jaraknya berkurang atau bertambah.

## Menyiapkan binlog yang disempurnakan untuk Aurora MySQL

Binlog yang ditingkatkan akan mengurangi overhead performa komputasi yang disebabkan oleh pengaktifan binlog, yang dapat mencapai hingga 50% dalam kasus tertentu. Dengan binlog yang ditingkatkan, overhead ini dapat dikurangi menjadi sekitar 13%. Untuk mengurangi overhead, binlog yang ditingkatkan menulis log biner dan transaksi ke penyimpanan secara paralel, yang meminimalkan data yang ditulis pada waktu commit transaksi.

Menggunakan binlog yang ditingkatkan juga meningkatkan waktu pemulihan basis data setelah mulai ulang dan failover hingga 99% dibandingkan dengan binlog MySQL komunitas. Binlog yang ditingkatkan kompatibel dengan beban kerja berbasis binlog yang ada, dan Anda berinteraksi dengannya dengan cara yang sama seperti Anda berinteraksi dengan binlog MySQL komunitas.

Binlog yang ditingkatkan tersedia di Aurora MySQL versi 3.03.1 dan lebih tinggi.

Topik


- Mengonfigurasi parameter binlog yang ditingkatkan
- Parameter terkait lainnya
- Perbedaan antara binlog yang ditingkatkan dan binlog MySQL komunitas
- CloudWatch Metrik Amazon untuk binlog yang disempurnakan
- Batasan binlog yang ditingkatkan


### Mengonfigurasi parameter binlog yang ditingkatkan

Anda dapat beralih antara binlog MySQL komunitas dan binlog yang ditingkatkan dengan mengaktifkan/menonaktifkan parameter binlog yang ditingkatkan. Konsumen binlog yang ada dapat terus membaca dan mengonsumsi file binlog tanpa kesenjangan dalam urutan file binlog.

Untuk mengaktifkan binlog yang disempurnakan, atur parameter berikut:

| Parameter | Default | Deskripsi |
|---|---|---|
| `binlog_format` | – | Atur parameter `binlog_format` ke format pencatatan log biner pilihan Anda untuk mengaktifkan binlog yang ditingkatkan. Pastikan `binlog_format parameter` tidak diatur ke OFF. Untuk informasi selengkapnya, lihat Mengonfigurasi pencatatan log biner Aurora MySQL. |
| `aurora_enhanced_binlog` | `0` | Tetapkan nilai parameter ini ke 1 dalam grup parameter klaster DB yang terkait dengan klaster Aurora MySQL. Ketika Anda mengubah nilai parameter ini, Anda |

| Parameter | Default | Deskripsi |
| --- | --- | --- |
| | | harus melakukan boot ulang instans penulis ketika nilai `DBClusterParameterGroupStatus` ditampilkan sebagai `pending-reboot` . |
| `binlog_backup` | `1` | Nonaktifkan parameter ini untuk mengaktifkan binlog yang ditingkatkan. Untuk melakukannya, tetapkan nilai parameter ini ke `0`. |
| `binlog_replication_globaldb` | `1` | Nonaktifkan parameter ini untuk mengaktifkan binlog yang ditingkatkan. Untuk melakukannya, tetapkan nilai parameter ini ke `0`. |

**Important**

Anda dapat menonaktifkan `binlog_replication_globaldb` parameter `binlog_backup` dan hanya ketika Anda menggunakan binlog yang ditingkatkan.

Untuk mematikan binlog yang disempurnakan, atur parameter berikut:

| Parameter | Deskripsi |
| --- | --- |
| `aurora_enhanced_binlog` | Tetapkan nilai parameter ini ke `0` dalam grup parameter klaster DB yang terkait dengan klaster Aurora MySQL. Setiap kali Anda mengubah nilai parameter ini, Anda harus melakukan boot ulang instans penulis ketika nilai `DBClusterParameterGroupStatus` ditampilkan sebagai `pending-reboot` . |

| Parameter | Deskripsi |
| --- | --- |
| `binlog_backup` | Aktifkan parameter ini saat Anda menonakti fkan binlog yang ditingkatkan. Untuk melakukan nya, tetapkan nilai parameter ini ke 1. |
| `binlog_replication_globaldb` | Aktifkan parameter ini saat Anda menonakti fkan binlog yang ditingkatkan. Untuk melakukan nya, tetapkan nilai parameter ini ke 1. |

Untuk memeriksa apakah binlog yang ditingkatkan diaktifkan, gunakan perintah berikut di klien MySQL:

```
mysql>show status like 'aurora_enhanced_binlog';

+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| aurora_enhanced_binlog | ACTIVE |
+-------------------------+--------+
1 row in set (0.00 sec)
```

Saat binlog yang ditingkatkan diaktifkan, output-nya akan menampilkan `ACTIVE` untuk `aurora_enhanced_binlog`.

Parameter terkait lainnya

Saat Anda mengaktifkan binlog yang ditingkatkan, parameter berikut akan terpengaruh:

- Parameter `max_binlog_size` terlihat, tetapi tidak dapat dimodifikasi. Nilai default-nya, `134217728`, secara otomatis disesuaikan menjadi `268435456` saat binlog yang ditingkatkan diaktifkan.
- Tidak seperti di binlog MySQL komunitas, `binlog_checksum` tidak bertindak sebagai parameter dinamis ketika binlog yang ditingkatkan diaktifkan. Agar perubahan pada parameter ini berlaku, Anda harus melakukan boot ulang klaster DB secara manual bahkan jika `ApplyMethod` adalah `immediate`.

- Nilai yang Anda tetapkan pada parameter `binlog_order_commits` tidak berpengaruh pada urutan commit saat binlog yang ditingkatkan diaktifkan. Commit selalu diperintahkan tanpa implikasi performa lebih lanjut.

Perbedaan antara binlog yang ditingkatkan dan binlog MySQL komunitas

Binlog yang ditingkatkan berinteraksi secara berbeda dengan klon, cadangan, dan basis data global Aurora jika dibandingkan dengan binlog MySQL komunitas. Sebaiknya Anda memahami perbedaan berikut sebelum menggunakan binlog yang ditingkatkan.

- File binlog yang ditingkatkan dari klaster DB sumber tidak tersedia di klaster DB yang dikloning.
- File binlog yang ditingkatkan tidak disertakan dalam cadangan Aurora. Oleh karena itu, file binlog yang ditingkatkan dari klaster DB sumber tidak tersedia setelah memulihkan klaster DB meskipun ada periode retensi yang ditetapkan padanya.
- Ketika digunakan dengan basis data global Aurora, file binlog yang ditingkatkan untuk klaster DB primer tidak direplikasi ke klaster DB di wilayah sekunder.

Contoh

Contoh berikut menggambarkan perbedaan antara binlog yang ditingkatkan dan binlog MySQL komunitas.

Pada klaster DB yang dipulihkan atau dikloning

Saat binlog yang ditingkatkan diaktifkan, file binlog historis tidak tersedia di klaster DB yang dipulihkan atau dikloning. Setelah operasi pemulihan atau kloning, jika binlog dihidupkan, cluster DB baru mulai menulis urutan file binlognya sendiri, mulai dari 1 (.000001)mysql-bin-changelog.

Untuk mengaktifkan binlog yang ditingkatkan setelah operasi pemulihan atau kloning, atur parameter klaster DB yang diperlukan pada klaster DB yang dipulihkan atau dikloning. Untuk informasi selengkapnya, lihat Mengonfigurasi parameter binlog yang ditingkatkan.

Example Contoh: Operasi klon atau pemulihan dilakukan saat binlog yang ditingkatkan dihidupkan

Klaster DB Sumber:

```
mysql> show binary logs;

+---------------------------+-----------+-----------+
| Log_name                  | File_size | Encrypted |
```

```
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Pada klaster DB yang dipulihkan atau dikloning, file binlog tidak akan dicadangkan saat binlog yang ditingkatkan diaktifkan. Untuk menghindari diskontinuitas dalam data binlog, file binlog yang ditulis sebelum binlog yang ditingkatkan diaktifkan juga tidak akan tersedia.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example Contoh: Operasi klon atau pemulihan dilakukan saat binlog yang ditingkatkan dimatikan

Klaster DB sumber:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
```

```
6 rows in set (0.00 sec)
```

Binlog yang disempurnakan dinonaktifkan setelahnya`mysql-bin-changelog.000003`. Pada klaster DB yang dipulihkan atau dikloning, file binlog yang ditulis setelah binlog yang ditingkatkan dinonaktifkan akan tersedia.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Pada basis data global Amazon Aurora

Pada basis data global Amazon Aurora, data binlog klaster DB primer tidak direplikasi ke klaster DB sekunder. Setelah proses failover lintas Wilayah, data binlog tidak tersedia di klaster DB primer yang baru dipromosikan. Jika binlog dihidupkan, cluster DB yang baru dipromosikan memulai urutan file binlognya sendiri, mulai dari 1 (mysql-bin-changelog.000001).

Untuk mengaktifkan binlog yang ditingkatkan setelah failover, Anda harus mengatur parameter klaster DB yang diperlukan pada klaster DB sekunder. Untuk informasi selengkapnya, lihat Mengonfigurasi parameter binlog yang ditingkatkan.

Example Contoh: Operasi failover database global dilakukan saat binlog yang ditingkatkan diaktifkan

Klaster DB primer lama (sebelum failover):

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
```

```
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Klaster DB primer baru (setelah failover):

File binlog tidak direplikasi ke wilayah sekunder saat binlog yang ditingkatkan diaktifkan. Untuk menghindari diskontinuitas dalam data binlog, file binlog yang ditulis sebelum binlog yang ditingkatkan diaktifkan tidak akan tersedia.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example Contoh: Operasi failover database global dilakukan saat binlog yang disempurnakan dimatikan

Klaster DB Sumber:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
```

```
+---------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Klaster DB yang dipulihkan atau dikloning:

Binlog yang disempurnakan dinonaktifkan setelahnya`mysql-bin-changelog.000003`. File binlog yang ditulis setelah binlog yang ditingkatkan dinonaktifkan akan direplikasi dan tersedia di klaster DB yang baru dipromosikan.

```
mysql>show binary logs;

+---------------------------+-----------+-----------+
| Log_name                  | File_size | Encrypted |
+---------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+---------------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

CloudWatch Metrik Amazon untuk binlog yang disempurnakan

CloudWatch Metrik Amazon berikut diterbitkan hanya ketika binlog yang disempurnakan diaktifkan.

| CloudWatch metrik | Deskripsi | Unit |
| --- | --- | --- |
| ChangeLogBytesUsed | Jumlah penyimpanan yang digunakan oleh binlog yang ditingkatkan. | Byte |
| ChangeLogReadIOPs | Jumlah operasi I/O baca yang dilakukan di binlog yang ditingkatkan dalam interval 5 menit. | Hitungan per 5 menit |
| ChangeLogWriteIOPs | Jumlah operasi I/O disk tulis yang dilakukan di binlog yang | Hitungan per 5 menit |

| CloudWatch metrik | Deskripsi | Unit |
| --- | --- | --- |
| | ditingkatkan dalam interval 5 menit. | |

Batasan binlog yang ditingkatkan

Batasan berikut berlaku untuk klaster Amazon Aurora DB saat binlog yang ditingkatkan diaktifkan.

- Binlog yang ditingkatkan hanya didukung di Aurora MySQL version3.03.1 dan yang lebih tinggi.
- File binlog yang ditingkatkan yang ditulis pada klaster DB primer tidak disalin ke klaster DB yang dikloning atau dipulihkan.
- Saat digunakan dengan basis data global Amazon Aurora, file binlog yang ditingkatkan untuk klaster DB primer tidak direplikasi ke klaster DB sekunder. Oleh karena itu, setelah proses failover, data binlog historis tidak tersedia di klaster DB primer yang baru.
- Parameter konfigurasi binlog berikut diabaikan:
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- Anda tidak dapat menghapus atau mengganti nama tabel yang rusak dalam basis data. Untuk menjatuhkan tabel ini, Anda dapat menghubungi Dukungan.
- Cache I/O binlog dinonaktifkan saat binlog yang ditingkatkan diaktifkan. Untuk informasi selengkapnya, lihat Mengoptimalkan replikasi log biner untuk Aurora MySQL.

  > **Note**
  >
  > Binlog yang ditingkatkan memberikan peningkatan performa baca yang serupa dengan cache I/O binlog dan peningkatan performa tulis yang lebih baik.

- Fitur pelacakan mundur tidak didukung. Binlog yang ditingkatkan tidak dapat diaktifkan di klaster DB dalam kondisi berikut:
  - Klaster DB dengan fitur pelacakan mundur saat ini diaktifkan.
  - Cluster DB tempat fitur backtrack sebelumnya diaktifkan, tetapi sekarang dinonaktifkan.
  - Klaster DB dipulihkan dari klaster DB sumber atau snapshot dengan fitur pelacakan mundur diaktifkan.

# Menggunakan replikasi GTID berbasis

Konten berikut menjelaskan cara menggunakan pengidentifikasi transaksi global (GTIDs) dengan replikasi log biner (binlog) di antara DB. antara SQL cluster Aurora My dan sumber eksternal.

> **Note**
>
> Untuk Aurora, Anda dapat menggunakan fitur ini hanya dengan Aurora My SQL cluster yang menggunakan replikasi binlog ke atau dari database Saya eksternal. SQL Basis data lainnya mungkin berupa SQL instans Amazon RDS My, SQL database Saya lokal, atau klaster Aurora DB di tempat lain. Wilayah AWS Untuk mempelajari cara mengonfigurasi jenis replikasi semacam itu, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Jika Anda menggunakan replikasi binlog dan tidak terbiasa dengan replikasi GTID berbasis dengan MySQL, lihat Replikasi dengan pengidentifikasi transaksi global di dokumentasi Saya. SQL

GTIDreplikasi berbasis didukung untuk Aurora SQL My versi 2 dan 3.

Topik

- Ikhtisar pengidentifikasi transaksi global () GTIDs
- Parameter untuk replikasi GTID berbasis
- Mengaktifkan replikasi GTID berbasis untuk cluster Aurora My SQL
- Menonaktifkan replikasi GTID berbasis untuk Aurora My DB SQL

## Ikhtisar pengidentifikasi transaksi global () GTIDs

Pengidentifikasi transaksi global (GTIDs) adalah pengidentifikasi unik yang dihasilkan untuk transaksi Saya SQL yang berkomitmen. Anda dapat menggunakan GTIDs untuk membuat replikasi binlog lebih sederhana dan lebih mudah untuk memecahkan masalah.

> **Note**
>
> Saat Aurora menyinkronkan data antar-instans DB dalam sebuah klaster, mekanisme replikasi tersebut tidak melibatkan log biner (binlog). Untuk Aurora MySQL, replikasi GTID berbasis hanya berlaku ketika Anda juga menggunakan replikasi binlog untuk mereplikasi ke

dalam atau keluar dari cluster Aurora My SQL DB dari database eksternal yang kompatibel dengan Saya. SQL

Saya SQL menggunakan dua jenis transaksi yang berbeda untuk replikasi binlog:

- GTIDTransaksi — Transaksi yang diidentifikasi oleh aGTID.
- Transaksi anonim — Transaksi yang tidak GTID ditetapkan.

Dalam konfigurasi replikasi, GTIDs unik di semua instans DB. GTIDsmenyederhanakan konfigurasi replikasi karena ketika Anda menggunakannya, Anda tidak perlu merujuk ke posisi file log. GTIDsjuga memudahkan untuk melacak transaksi yang direplikasi dan menentukan apakah instance sumber dan replika konsisten.

Anda biasanya menggunakan replikasi GTID berbasis dengan Aurora saat mereplikasi dari database eksternal yang kompatibel dengan SQL Saya ke dalam cluster Aurora. Anda dapat mengatur konfigurasi replikasi ini sebagai bagian dari migrasi dari RDS database lokal atau Amazon ke Aurora Milikku. SQL Jika database eksternal sudah menggunakanGTIDs, mengaktifkan replikasi GTID berbasis untuk cluster Aurora menyederhanakan proses replikasi.

Anda mengonfigurasi replikasi GTID berbasis untuk klaster Aurora SQL My dengan terlebih dahulu menyetel parameter konfigurasi yang relevan dalam grup parameter cluster DB. Kemudian, hubungkan grup parameter tersebut dengan klaster.

## Parameter untuk replikasi GTID berbasis

Gunakan parameter berikut untuk mengkonfigurasi replikasi GTID berbasis.

| Parameter | Nilai valid | Deskripsi |
| --- | --- | --- |
| `gtid_mode` | `OFF`, `OFF_PERMISSIVE` , `ON_PERMISSIVE` , `ON` | `OFF` menetapkan bahwa transaksi baru adalah transaksi anonim (yaitu, tidak memiliki GTIDs), dan transaksi harus anonim agar dapat direplikasi.<br><br>`OFF_PERMISSIVE` menentukan bahwa transaksi baru adalah transaksi anonim, tetapi semua transaksi dapat direplikasi. |

| Parameter | Nilai valid | Deskripsi |
| --- | --- | --- |
| | | `ON_PERMISSIVE` menetapkan bahwa transaksi baru adalah GTID transaksi, tetapi semua transaksi dapat direplikasi.<br><br>`ON`menetapkan bahwa transaksi baru adalah GTID transaksi, dan transaksi harus berupa GTID transaksi yang akan direplikasi. |
| `enforce_gtid_consistency` | `OFF`, `ON`, `WARN` | `OFF`memungkinkan transaksi melanggar GTID konsistensi.<br><br>`ON`mencegah transaksi melanggar GTID konsistensi.<br><br>`WARN`memungkinkan transaksi melanggar GTID konsistensi tetapi menghasilkan peringatan ketika pelanggaran terjadi. |

> **Note**
>
> Dalam AWS Management Console, `gtid_mode` parameter muncul sebagai`gtid-mode`.

Untuk replikasi GTID berbasis, gunakan pengaturan ini untuk grup parameter cluster DB untuk cluster Aurora SQL My DB Anda:

- `ON`dan hanya `ON_PERMISSIVE` berlaku untuk replikasi keluar dari cluster Aurora My. SQL Kedua nilai ini menyebabkan cluster Aurora DB Anda digunakan GTIDs untuk transaksi yang direplikasi ke database eksternal. `ON`mengharuskan database eksternal juga menggunakan replikasi GTID berbasis. `ON_PERMISSIVE`membuat replikasi GTID berbasis opsional pada database eksternal.
- `OFF_PERMISSIVE`, jika diatur, artinya klaster DB Aurora Anda dapat menerima replikasi masuk dari basis data eksternal. Hal ini dapat dilakukan apakah database eksternal menggunakan replikasi GTID berbasis atau tidak.
- `OFF`, jika disetel, berarti klaster Aurora DB Anda hanya menerima replikasi masuk dari database eksternal yang tidak menggunakan replikasi berbasis. GTID

> **Tip**
>
> Replikasi masuk adalah skenario replikasi binlog yang paling umum untuk klaster Aurora My. SQL Untuk replikasi yang masuk, kami sarankan Anda mengatur GTID mode ke. `OFF_PERMISSIVE` Pengaturan itu memungkinkan replikasi masuk dari database eksternal terlepas dari GTID pengaturan di sumber replikasi.

Untuk informasi selengkapnya tentang grup parameter, lihat .

## Mengaktifkan replikasi GTID berbasis untuk cluster Aurora My SQL

Ketika replikasi GTID berbasis diaktifkan untuk cluster Aurora SQL My DB, GTID pengaturan berlaku untuk replikasi binlog masuk dan keluar.

Untuk mengaktifkan replikasi GTID berbasis untuk cluster Aurora My SQL

1. Buat atau edit grup parameter klaster DB menggunakan pengaturan parameter berikut:
   - `gtid_mode` – `ON` atau `ON_PERMISSIVE`
   - `enforce_gtid_consistency` – `ON`
2. Kaitkan grup parameter cluster DB dengan cluster Aurora MySQL. Untuk melakukannya, ikuti prosedur dalam .
3. (Opsional) Tentukan cara menetapkan GTIDs ke transaksi yang tidak menyertakannya. Untuk melakukannya, panggil prosedur yang tersimpan di mysql.rds_assign_gtids_to_anonymous_transactions (Aurora Versi saya 3) SQL.

## Menonaktifkan replikasi GTID berbasis untuk Aurora My DB SQL

Anda dapat menonaktifkan replikasi GTID berbasis untuk cluster Aurora SQL My DB. Melakukannya berarti cluster Aurora tidak dapat melakukan replikasi binlog masuk atau keluar dengan database eksternal yang menggunakan replikasi berbasis. GTID

> **Note**
>
> Dalam prosedur berikut, replika baca berarti target replikasi dalam konfigurasi Aurora dengan replikasi binlog ke atau dari basis data eksternal. Jadi, itu bukan instans DB Aurora Replica

hanya-baca. Misalnya, saat sebuah klaster Aurora menerima replikasi masuk dari sebuah sumber eksternal, instans primer Aurora bertindak sebagai replika baca untuk replikasi binlog.

Untuk detail selengkapnya tentang prosedur tersimpan yang disebutkan di bagian ini, lihat Aurora Referensi prosedur SQL tersimpan saya.

Untuk menonaktifkan replikasi GTID berbasis untuk replika baca

1. Pada replika Aurora, jalankan prosedur berikut:

   Untuk versi 3

   ```
   CALL mysql.rds_set_source_auto_position(0);
   ```

   Untuk versi 2

   ```
   CALL mysql.rds_set_master_auto_position(0);
   ```

2. Atur ulang `gtid_mode` ke `ON_PERMISSIVE`.

   a. Pastikan bahwa grup parameter cluster DB yang terkait dengan SQL cluster Aurora My telah `gtid_mode` disetel ke. `ON_PERMISSIVE`

      Untuk informasi selengkapnya tentang cara mengatur parameter konfigurasi menggunakan grup parameter, lihat .

   b. Mulai ulang cluster Aurora My SQL DB.

3. Atur ulang `gtid_mode` ke `OFF_PERMISSIVE`.

   a. Pastikan bahwa grup parameter cluster DB yang terkait dengan SQL cluster Aurora My telah `gtid_mode` disetel ke. `OFF_PERMISSIVE`

   b. Mulai ulang cluster Aurora My SQL DB.

4. Tunggu semua GTID transaksi diterapkan pada instance utama Aurora. Untuk memeriksa apakah ini diterapkan, lakukan langkah-langkah berikut:

   a. Pada instance utama Aurora, jalankan perintah. `SHOW MASTER STATUS`

      Output Anda harus mirip dengan output berikut.

```
File                          Position
-----------------------------------
mysql-bin-changelog.000031      107
-----------------------------------
```

Perhatikan file dan posisi dalam output Anda.

b. Pada setiap replika baca, gunakan file dan informasi posisi dari contoh sumbernya pada langkah sebelumnya untuk menjalankan kueri berikut:

Untuk versi 3

```
SELECT SOURCE_POS_WAIT('file', position);
```

Untuk versi 2

```
SELECT MASTER_POS_WAIT('file', position);
```

Misalnya, jika nama file `mysql-bin-changelog.000031` dan posisinya`107`, jalankan pernyataan berikut:

Untuk versi 3

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

Untuk versi 2

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. Setel ulang GTID parameter untuk menonaktifkan replikasi GTID berbasis.

a. Pastikan bahwa grup parameter cluster DB yang terkait dengan Aurora My SQL cluster memiliki pengaturan parameter berikut:

- `gtid_mode` – `OFF`
- `enforce_gtid_consistency` – `OFF`

b. Mulai ulang cluster Aurora My SQL DB.

# Menggunakan penerusan tulis lokal di klaster DB Amazon Aurora MySQL

Penerusan tulis lokal (dalam klaster) memungkinkan aplikasi Anda mengeluarkan transaksi baca/tulis langsung pada Replika Aurora. Transaksi ini kemudian diteruskan ke instans DB penulis untuk di-commit. Anda dapat menggunakan penerusan tulis lokal ketika aplikasi Anda memerlukan read-after-write konsistensi, yang merupakan kemampuan untuk membaca tulisan terbaru dalam suatu transaksi.

Replika baca menerima pembaruan secara asinkron dari penulis. Tanpa penerusan tulis, Anda harus bertransaksi pembacaan apa pun yang memerlukan read-after-write konsistensi pada instance DB penulis. Atau Anda harus mengembangkan logika aplikasi kustom yang kompleks untuk memanfaatkan beberapa replika baca untuk skalabilitas. Aplikasi Anda harus sepenuhnya membagi semua lalu lintas baca dan tulis, sehingga mempertahankan dua set koneksi basis data untuk mengirim lalu lintas ke titik akhir yang benar. Overhead pengembangan ini memperumit desain aplikasi ketika kueri merupakan bagian dari satu sesi logis, atau transaksi, di dalam aplikasi. Selain itu, karena lag replikasi dapat berbeda di antara replika baca, sulit untuk mencapai konsistensi baca global di semua instans di dalam basis data.

Penerusan tulis menghindari kebutuhan untuk membagi transaksi tersebut atau mengirimkannya secara eksklusif ke penulis, yang menyederhanakan pengembangan aplikasi. Kemampuan baru ini memudahkan pencapaian skala baca untuk beban kerja yang perlu membaca penulisan terbaru dalam suatu transaksi dan tidak sensitif terhadap latensi penulisan.

Penerusan tulis lokal berbeda dari penerusan tulis global, yang meneruskan penulisan dari klaster DB sekunder ke klaster DB primer dalam basis data global Aurora. Anda dapat menggunakan penerusan tulis lokal dalam klaster DB yang merupakan bagian dari basis data global Aurora. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Penerusan tulis lokal membutuhkan Aurora MySQL versi 3.04 atau versi yang lebih tinggi.

Topik

- Mengaktifkan penerusan tulis lokal
- Memeriksa apakah klaster DB memiliki penerusan tulis yang diaktifkan
- Kompatibilitas aplikasi dan SQL dengan penerusan tulis
- Tingkat isolasi untuk penerusan tulis

- Konsistensi baca untuk penerusan tulis
- Menjalankan pernyataan multibagian dengan penerusan tulis
- Transaksi dengan penerusan tulis
- Parameter konfigurasi untuk penerusan tulis
- CloudWatch Metrik Amazon dan Aurora Variabel status SQL saya untuk penerusan tulis
- Mengidentifikasi transaksi dan kueri yang diteruskan

## Mengaktifkan penerusan tulis lokal

Secara default, penerusan tulis lokal tidak diaktifkan untuk klaster Aurora My DB. SQL Anda mengaktifkan penerusan tulis lokal pada tingkat klaster, bukan pada tingkat instans.

> **Important**
>
> Anda juga dapat mengaktifkan penerusan tulis lokal untuk replika baca lintas wilayah yang menggunakan logging biner, tetapi operasi penulisan tidak diteruskan ke sumber Wilayah AWS. Mereka diteruskan ke instance DB penulis dari cluster replika baca binlog. Gunakan metode ini hanya jika Anda memiliki kasus penggunaan untuk menulis ke replika baca binlog di sekunder Wilayah AWS. Jika tidak, Anda mungkin berakhir dengan skenario "otak terbelah" di mana kumpulan data yang direplikasi tidak konsisten satu sama lain. Kami menyarankan Anda menggunakan penerusan tulis global dengan basis data global, daripada penerusan tulis lokal pada replika baca lintas wilayah, kecuali benar-benar diperlukan. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

### Konsol

Menggunakan AWS Management Console, pilih kotak centang Aktifkan penerusan tulis lokal di bawah Baca penerusan tulis replika saat Anda membuat atau memodifikasi cluster DB.

### AWS CLI

Untuk mengaktifkan penerusan tulis dengan AWS CLI, gunakan `--enable-local-write-forwarding` opsi. Opsi ini berfungsi saat Anda membuat klaster DB baru menggunakan perintah `create-db-cluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster DB yang ada dengan

menggunakan perintah `modify-db-cluster`. Anda dapat menonaktifkan penerusan tulis dengan menggunakan `--no-enable-local-write-forwarding` opsi dengan perintah yang sama CLI ini.

Contoh berikut membuat cluster Aurora My SQL DB dengan penerusan tulis diaktifkan.

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

Anda kemudian membuat instans DB penulis dan pembaca sehingga Anda dapat menggunakan penerusan tulis. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

RDS API

Untuk mengaktifkan penerusan tulis menggunakan Amazon RDSAPI, setel `EnableLocalWriteForwarding` parameter ke. `true` Parameter ini berfungsi saat Anda membuat klaster DB baru menggunakan operasi `CreateDBCluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster DB yang ada dengan menggunakan operasi `ModifyDBCluster`. Anda dapat menonaktifkan penerusan tulis dengan mengatur parameter `EnableLocalWriteForwarding` ke `false`.

## Mengaktifkan penerusan tulis untuk sesi basis data

Parameter `aurora_replica_read_consistency` adalah parameter DB dan parameter klaster DB yang memungkinkan penerusan tulis. Anda dapat menentukan `EVENTUAL`, `SESSION`, atau `GLOBAL` untuk tingkat konsistensi baca. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Konsistensi baca untuk penerusan tulis.

Aturan berikut berlaku untuk parameter ini:

- Nilai default-nya adalah " (nol).
- Penerusan tulis hanya tersedia jika Anda menyetel `aurora_replica_read_consistency` ke `EVENTUAL`, `SESSION`, atau `GLOBAL`. Parameter ini hanya relevan dalam instans pembaca klaster DB yang memiliki penerusan tulis diaktifkan.

- Anda tidak dapat mengatur parameter ini (saat kosong) atau membatalkan pengaturan (saat sudah diatur) di dalam transaksi multipernyataan. Anda dapat mengubahnya dari satu nilai yang valid ke nilai valid lainnya selama transaksi tersebut, tetapi kami tidak merekomendasikan tindakan ini.

## Memeriksa apakah klaster DB memiliki penerusan tulis yang diaktifkan

Untuk menentukan bahwa Anda dapat menggunakan penerusan tulis di klaster DB, konfirmasikan bahwa klaster memiliki atribut `LocalWriteForwardingStatus` disetel ke `enabled`.

Di AWS Management Console, pada tab Konfigurasi halaman detail untuk klaster, Anda melihat status Diaktifkan untuk penerusan tulis replika baca lokal.

Untuk melihat status setelan penerusan tulis untuk semua cluster Anda, jalankan perintah berikut.
AWS CLI

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

Sebuah klaster DB dapat memiliki nilai berikut untuk `LocalWriteForwardingStatus`:

- `disabled` – Penerusan tulis dinonaktifkan.
- `disabling` – Penerusan tulis dalam proses penonaktifan.
- `enabled` – Penerusan tulis diaktifkan.
- `enabling` – Penerusan tulis dalam proses pengaktifan.
- `null` – Penerusan tulis tidak tersedia untuk klaster DB ini.
- `requested` – Penerusan tulis telah diminta, tetapi belum aktif.

## Kompatibilitas aplikasi dan SQL dengan penerusan tulis

Anda dapat menggunakan jenis pernyataan SQL berikut dengan penerusan tulis:

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`. Ada beberapa pembatasan pada properti pernyataan ini yang dapat Anda gunakan dengan penerusan tulis, seperti yang dijelaskan di bawah ini.
- Pernyataan `SELECT ... LOCK IN SHARE MODE` dan `SELECT FOR UPDATE`.
- Pernyataan `PREPARE` dan `EXECUTE`.

Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil usang saat Anda menggunakannya dalam klaster DB dengan penerusan tulis. Selain itu, fungsi yang ditentukan pengguna dan prosedur yang ditentukan pengguna tidak didukung. Oleh karena itu, pengaturan `EnableLocalWriteForwarding` dinonaktifkan secara default untuk klaster DB. Sebelum diaktifkan, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.

Batasan berikut berlaku untuk pernyataan SQL yang Anda gunakan untuk penerusan tulis. Dalam beberapa kasus, Anda dapat menggunakan pernyataan pada klaster DB dengan penerusan tulis yang diaktifkan. Pendekatan ini berfungsi jika penerusan tulis tidak diaktifkan dalam sesi oleh parameter konfigurasi `aurora_replica_read_consistency`. Jika Anda mencoba menggunakan suatu pernyataan yang tidak diizinkan karena adanya penerusan tulis, Anda akan melihat pesan kesalahan seperti berikut ini:

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

Bahasa definisi data (DDL)

Hubungkan ke instans DB tulis untuk menjalankan pernyataan DDL. Anda tidak dapat menjalankannya dari instans DB pembaca.

Memperbarui tabel permanen menggunakan data dari tabel sementara

Anda dapat menggunakan tabel sementara pada klaster DB dengan penerusan tulis yang diaktifkan. Namun, Anda tidak dapat menggunakan pernyataan DML untuk mengubah tabel permanen jika pernyataan tersebut mengacu pada tabel sementara. Misalnya, Anda tidak dapat menggunakan pernyataan `INSERT ... SELECT` yang mengambil data dari tabel sementara.

Transaksi XA

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster DB yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Sebelum mengaktifkan penerusan tulis dalam sebuah sesi, periksa apakah kode Anda menggunakan pernyataan ini.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

Pernyataan LOAD untuk tabel permanen

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB dengan penerusan tulis yang diaktifkan.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

Pernyataan plugin

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB dengan penerusan tulis yang diaktifkan.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
```

```
UNINSTALL PLUGIN example;
```

Pernyataan SAVEPOINT

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster DB yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Periksa apakah kode Anda menggunakan pernyataan ini sebelum mengaktifkan penerusan tulis dalam sebuah sesi.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

## Tingkat isolasi untuk penerusan tulis

Dalam sesi yang menggunakan penerusan tulis, Anda hanya dapat menggunakan tingkat isolasi `REPEATABLE READ`. Meskipun Anda juga dapat menggunakan tingkat isolasi `READ COMMITTED` dengan Replika Aurora, tingkat isolasi tersebut tidak berfungsi dengan penerusan tulis. Untuk informasi tentang tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`, lihat Tingkat isolasi Aurora MySQL.

## Konsistensi baca untuk penerusan tulis

Anda dapat mengontrol tingkat konsistensi baca di klaster DB. Tingkat konsistensi baca menentukan berapa lama klaster DB menunggu sebelum operasi baca untuk memastikan beberapa atau semua perubahan direplikasi dari penulis. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi tulis yang diteruskan dari sesi Anda terlihat dalam klaster DB sebelum kueri berikutnya. Anda juga dapat menggunakan pengaturan ini untuk memastikan kueri pada klaster DB selalu melihat pembaruan terkini dari penulis. Pengaturan ini juga berlaku untuk kueri yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, pilih nilai untuk parameter DB `aurora_replica_read_consistency` atau parameter klaster DB.

> **Important**
>
> Selalu atur parameter DB `aurora_replica_read_consistency` atau parameter klaster DB saat Anda ingin meneruskan penulisan. Jika Anda tidak mengaturnya,

Aurora tidak akan meneruskan penulisan. Parameter ini mempunyai nilai kosong secara default, jadi pilih nilai tertentu apabila Anda menggunakan parameter ini. Parameter `aurora_replica_read_consistency` hanya memengaruhi klaster atau instans DB yang mengaktifkan penerusan tulis.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu untuk menunggu perubahan disebarkan di antara instans-instans DB. Anda dapat memilih keseimbangan antara waktu respons yang cepat dan memastikan bahwa perubahan yang dilakukan dalam instans DB lainnya sepenuhnya tersedia sebelum kueri Anda berjalan.

Anda dapat menentukan nilai berikut untuk parameter `aurora_replica_read_consistency`:

- `EVENTUAL` – Hasil operasi tulis dalam sesi yang sama tidak terlihat sampai operasi tulis dilakukan pada instans DB penulis. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi. Ini adalah konsistensi yang sama seperti untuk cluster Aurora My SQL DB yang tidak menggunakan penerusan tulis.
- `SESSION` – Semua kueri yang menggunakan penerusan tulis melihat hasil semua perubahan yang dilakukan dalam sesi tersebut. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi tulis yang diteruskan untuk direplikasi.
- `GLOBAL` – Sebuah sesi melihat semua perubahan yang dilakukan di semua sesi dan instans di dalam klaster DB. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlanjut ketika cluster DB up-to-date dengan semua data yang dikomit dari penulis, pada saat kueri dimulai.

Untuk informasi tentang parameter konfigurasi yang terlibat dalam penerusan tulis, lihat Parameter konfigurasi untuk penerusan tulis.

**Note**

Anda juga dapat menggunakan `aurora_replica_read_consistency` sebagai variabel sesi, misalnya:

```
mysql> set aurora_replica_read_consistency = 'session';
```

## Contoh menggunakan penerusan tulis

Contoh berikut ini menunjukkan efek parameter `aurora_replica_read_consistency` pada pernyataan `INSERT` yang berjalan diikuti oleh pernyataan `SELECT`. Hasilnya dapat berbeda, tergantung nilai `aurora_replica_read_consistency` dan waktu pernyataan.

Untuk mencapai konsistensi yang lebih tinggi, Anda mungkin menunggu sebentar sebelum mengeluarkan pernyataan `SELECT`. Atau Aurora dapat secara otomatis menunggu sampai hasil selesai mereplikasi sebelum melanjutkan dengan `SELECT`.

Untuk informasi tentang cara menetapkan parameter DB, lihat .

Example dengan **`aurora_replica_read_consistency`** diatur ke **`EVENTUAL`**

Menjalankan pernyataan `INSERT`, yang langsung diikuti dengan pernyataan `SELECT`, memunculkan nilai untuk `COUNT(*)` dengan jumlah baris sebelum baris baru disisipkan. Dengan menjalankan `SELECT` lagi beberapa saat kemudian, muncul hitungan baris yang diperbarui. Pernyataan `SELECT` tidak menunggu.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example dengan **`aurora_replica_read_consistency`** diatur ke **`SESSION`**

Pernyataan `SELECT` langsung setelah `INSERT` akan menunggu sampai perubahan dari pernyataan `INSERT` terlihat. Pernyataan `SELECT` selanjutnya tidak menunggu.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)

mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca tetap disetel ke `SESSION`, memperkenalkan waktu tunggu singkat setelah melaksanakan pernyataan `INSERT` membuat hitungan baris yang diperbarui tersedia pada saat pernyataan `SELECT` berikutnya berjalan.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
```

```
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example dengan **`aurora_replica_read_consistency`** diatur ke **`GLOBAL`**

Setiap pernyataan `SELECT` menunggu sampai semua perubahan data, sejak waktu mulai pernyataan, terlihat sebelum melakukan kueri. Waktu tunggu untuk setiap pernyataan `SELECT` bervariasi, tergantung jumlah lag replikasi.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

## Menjalankan pernyataan multibagian dengan penerusan tulis

Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke instans DB tulis dan berjalan di sana.

## Transaksi dengan penerusan tulis

Jika mode akses transaksi disetel ke hanya-baca, penerusan tulis tidak digunakan. Anda dapat menentukan mode akses untuk transaksi dengan menggunakan pernyataan `SET TRANSACTION` atau pernyataan `START TRANSACTION`. Anda juga dapat menentukan mode akses transaksi dengan mengubah nilai variabel sesi transaction_read_only. Anda dapat mengubah nilai sesi ini hanya saat Anda terhubung dengan klaster DB yang memiliki penerusan tulis diaktifkan.

Jika transaksi yang berlangsung lama tidak mengeluarkan pernyataan apa pun dalam waktu yang lama, transaksi tersebut kemungkinan melebihi periode batas waktu idle. Periode ini memiliki nilai default satu menit. Anda dapat mengatur parameter `aurora_fwd_writer_idle_timeout` untuk meningkatkannya hingga satu hari. Transaksi yang melebihi batas waktu idle dibatalkan oleh instans tulis. Pernyataan berikutnya yang Anda kirimkan akan menerima kesalahan waktu habis. Kemudian Aurora akan mengembalikan transaksi.

Jenis kesalahan ini dapat terjadi dalam kasus lain ketika penerusan tulis menjadi tidak tersedia. Misalnya, Aurora membatalkan transaksi apa pun yang menggunakan penerusan tulis jika Anda memulai ulang klaster DB atau jika Anda menonaktifkan penerusan tulis.

Ketika instans penulis dalam sebuah klaster yang menggunakan penerusan tulis lokal dimulai ulang, transaksi dan kueri yang aktif dan diteruskan pada instans pembaca yang menggunakan penerusan tulis lokal akan ditutup secara otomatis. Setelah instans penulis tersedia lagi, Anda dapat mencoba kembali transaksi ini.

## Parameter konfigurasi untuk penerusan tulis

Grup parameter DB Aurora mencakup pengaturan untuk fitur penerusan tulis. Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

| Parameter | Cakupan | Tipe | Nilai default | Nilai valid |
|---|---|---|---|---|
| `aurora_fwd_writer_idle_timeout` | Klaster | Integer tanpa tanda | 60 | 1-86.400 |
| `aurora_fwd_writer_max_connections_pct` | Klaster | Integer panjang | 10 | 0–90 |

| Parameter | Cakupan | Tipe | Nilai default | Nilai valid |
|---|---|---|---|---|
| | | tanpa tanda | | |
| aurora_replica_read_consistency | Klaster atau instans | Enum | " (null) | EVENTUAL, SESSION, GLOBAL |

Untuk mengontrol permintaan tulis yang masuk, gunakan pengaturan ini:

- aurora_fwd_writer_idle_timeout – Jumlah detik instans yang diperlukan oleh instans DB tulis untuk menunggu aktivitas pada koneksi yang diteruskan dari instans pembaca sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan membatalkan sesi.
- aurora_fwd_writer_max_connections_pct – Batas atas pada koneksi basis data yang dapat digunakan pada instans DB penulis untuk menangani kueri yang diteruskan dari instans pembaca. Ini dinyatakan sebagai persentase dari pengaturan max_connections untuk penulis. Misalnya, jika max_connections adalah 800 dan aurora_fwd_master_max_connections_pct atau aurora_fwd_writer_max_connections_pct adalah 10, maka penulis mengizinkan maksimal 80 sesi terusan serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan max_connections.

  Pengaturan ini hanya berlaku pada penulis saat penerusan tulis diaktifkan. Jika Anda menurunkan nilai, koneksi yang ada tidak akan terpengaruh. Aurora mempertimbangkan nilai baru pengaturan saat mencoba membuat koneksi baru dari klaster DB. Nilai default-nya adalah 10, mewakili 10% dari nilai max_connections.

> **Note**
>
> Karena aurora_fwd_writer_idle_timeout dan aurora_fwd_writer_max_connections_pct adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk parameter ini.

Untuk informasi selengkapnya tentang `aurora_replica_read_consistency`, lihat Konsistensi baca untuk penerusan tulis.

Untuk mengetahui informasi selengkapnya tentang grup parameter DB, lihat .

## CloudWatch Metrik Amazon dan Aurora Variabel status SQL saya untuk penerusan tulis

CloudWatch Metrik Amazon dan variabel status Aurora SQL My berikut berlaku saat Anda menggunakan penerusan tulis untuk Aurora My. SQL Untuk informasi selengkapnya tentang metrik untuk instans DB Aurora SQL My writer dan reader, lihat topik berikut.

Topik

- Metrik untuk penerusan tulis untuk Aurora Instans DB penulis saya SQL
- Metrik untuk penerusan tulis untuk Aurora Instans DB pembaca saya SQL

### Metrik untuk penerusan tulis untuk Aurora Instans DB penulis saya SQL

CloudWatch Metrik Amazon berikut berlaku saat Anda menggunakan penerusan tulis pada satu atau beberapa cluster DB. Metrik ini semuanya diukur pada instance DB penulis.

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| `ForwardingWriterDMLLatency` | Milidetik | Rata-rata waktu untuk memproses setiap DML pernyataan yang diteruskan pada instance DB penulis.<br><br>Ini tidak termasuk waktu yang diperlukan klaster DB untuk meneruskan permintaan tulis, atau waktu untuk mereplikasi perubahan kembali ke penulis. |
| `ForwardingWriterDMLThroughput` | Hitungan per detik | Jumlah DML pernyataan diteruskan diproses setiap detik oleh instance DB penulis ini. |

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| `ForwardingWriterOpenSessions` | Hitung | Jumlah sesi yang diteruskan pada instans DB tulis. |

Variabel SQL status Aurora My berikut berlaku saat Anda menggunakan penerusan tulis pada satu atau lebih cluster DB. Variabel status ini semuanya diukur pada instance DB penulis.

| Aurora Variabel status saya SQL | Unit | Deskripsi |
| --- | --- | --- |
| `Aurora_fwd_writer_dml_stmt_count` | Hitung | Total jumlah pernyataan DML yang diteruskan ke instans DB penulis ini. |
| `Aurora_fwd_writer_dml_stmt_duration` | Mikrodetik | Total durasi pernyataan DML yang diteruskan ke instans DB penulis ini. |
| `Aurora_fwd_writer_open_sessions` | Hitung | Jumlah sesi yang diteruskan pada instans DB tulis. |
| `Aurora_fwd_writer_select_stmt_count` | Hitung | Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB penulis ini. |
| `Aurora_fwd_writer_select_stmt_duration` | Mikrodetik | Total durasi pernyataan `SELECT` yang diteruskan ke instans DB penulis ini. |

## Metrik untuk penerusan tulis untuk Aurora Instans DB pembaca saya SQL

CloudWatch Metrik berikut diukur pada setiap instans DB pembaca dalam cluster DB dengan penerusan tulis diaktifkan.

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| `ForwardingReplicaDMLLatency` | Milidetik | Waktu respons rata-rata diteruskan DMLs pada replika. |
| `ForwardingReplicaDMLThroughput` | Hitungan per detik | Jumlah DML pernyataan yang diteruskan diproses setiap detik. |
| `ForwardingReplicaOpenSessions` | Hitung | Jumlah sesi yang menggunakan penerusan tulis pada instans DB pembaca. |
| `ForwardingReplicaReadWaitLatency` | Milidetik | Rata-rata waktu tunggu yang diperlukan sebuah pernyataan `SELECT` di instans DB pembaca untuk menunggu agar dapat mengimbangi penulis.<br><br>Batas tunggu instans DB pembaca sebelum memproses kueri ditentukan oleh pengaturan `aurora_replica_read_consistency` . |
| `ForwardingReplicaReadWaitThroughput` | Hitungan per detik | Jumlah total pernyataan `SELECT` yang diproses setiap detik di semua sesi yang meneruskan tulis. |
| `ForwardingReplicaSelectLatency` | Milidetik | Latensi `SELECT` yang diteruskan, dirata-ratakan atas semua pernyataan `SELECT` yang diteruskan dalam periode pemantauan. |

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| `ForwardingReplicaSelectThroughput` | Hitungan per detik | Rata-rata throughput `SELECT` per detik yang diteruskan dalam periode pemantauan. |

Variabel SQL status Aurora My berikut diukur pada setiap instans DB pembaca dalam cluster DB dengan penerusan tulis diaktifkan.

| Aurora Variabel status saya SQL | Unit | Deskripsi |
| --- | --- | --- |
| `Aurora_fwd_replica_dml_stmt_count` | Hitung | Total jumlah pernyataan DML yang diteruskan ke instans DB pembaca ini. |
| `Aurora_fwd_replica_dml_stmt_duration` | Mikrodetik | Total durasi semua DML pernyataan yang diteruskan dari instance DB pembaca ini. |
| `Aurora_fwd_replica_errors_session_limit` | Hitung | Jumlah sesi yang ditolak oleh klaster primer karena salah satu kondisi kesalahan berikut:<br>• penulis penuh<br>• Terlalu banyak pernyataan yang diteruskan dalam proses. |
| `Aurora_fwd_replica_open_sessions` | Hitung | Jumlah sesi yang menggunakan penerusan tulis pada instans DB pembaca. |
| `Aurora_fwd_replica_read_wait_count` | Hitung | Jumlah total read-after-write menunggu pada instans DB pembaca ini. |

| Aurora Variabel status saya SQL | Unit | Deskripsi |
| --- | --- | --- |
| `Aurora_fwd_replica_read_wait_duration` | Mikrodetik | Total durasi tunggu karena pengaturan konsistensi baca pada instans DB pembaca ini. |
| `Aurora_fwd_replica_select_stmt_count` | Hitungan | Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini. |
| `Aurora_fwd_replica_select_stmt_duration` | Mikrodetik | Total durasi pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini. |

## Mengidentifikasi transaksi dan kueri yang diteruskan

Anda dapat menggunakan tabel `information_schema.aurora_forwarding_processlist` untuk mengidentifikasi transaksi dan kueri yang diteruskan. Untuk informasi selengkapnya tentang tabel ini, lihat information_schema.aurora_forwarding_processlist.

Contoh berikut ini menunjukkan semua koneksi yang diteruskan pada instans DB tulis.

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+--------------------+----------+---------+------+--------------
+--------------------------------------------+--------------+-------------------
+---------------------------------+----------------------+----------------+
| ID  | USER     | HOST               | DB       | COMMAND | TIME | STATE        |
 INFO                                     | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER     | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+--------------------+----------+---------+------+--------------
+--------------------------------------------+--------------+-------------------
+---------------------------------+--------------------------------------+
| 648 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
```

```
| 650 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |               639 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+---------------------------------+---------------------+----------------+
```

Pada instans DB pembaca penerusan, Anda dapat melihat thread yang terkait dengan koneksi DB penulis ini dengan menjalankan `SHOW PROCESSLIST`. Nilai `REPLICA_SESSION_ID` pada penulis, 637 dan 639, sama dengan nilai `Id` pada pembaca.

```
mysql> select @@aurora_server_id;

+---------------------------------+
| @@aurora_server_id              |
+---------------------------------+
| my-db-cluster-instance-2        |
+---------------------------------+
1 row in set (0.00 sec)

mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State        | Info
                                        |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
12 rows in set (0.00 sec)
```

# Mengintegrasikan Amazon Aurora SQL My dengan layanan lain AWS

Amazon Aurora My SQL terintegrasi dengan AWS layanan lain sehingga Anda dapat memperluas cluster Aurora My SQL DB Anda untuk menggunakan kemampuan tambahan di Cloud. AWS Cluster Aurora My SQL DB Anda dapat menggunakan AWS layanan untuk melakukan hal berikut:

- Secara sinkron atau asinkron memanggil AWS Lambda fungsi menggunakan fungsi asli atau. `lambda_sync` `lambda_async` Untuk informasi selengkapnya, lihat Memanggil fungsi Lambda dari cluster Amazon Aurora My DB SQL.
- Muat data dari teks atau XML file yang disimpan di bucket Amazon Simple Storage Service (Amazon S3) ke dalam cluster DB menggunakan `LOAD DATA FROM S3` perintah atau. `LOAD XML FROM S3` Untuk informasi selengkapnya, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3.
- Simpan data ke berkas teks yang disimpan dalam bucket Amazon S3 dari klaster DB Anda menggunakan perintah `SELECT INTO OUTFILE S3`. Untuk informasi selengkapnya, lihat Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3.
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Application Auto Scaling. Untuk informasi selengkapnya, lihat Auto Scaling Amazon Aurora dengan Replika Aurora.
- Lakukan analisis sentimen dengan Amazon Comprehend, atau berbagai macam algoritma pembelajaran mesin dengan AI. SageMaker Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora.

Aurora mengamankan kemampuan untuk mengakses AWS layanan lain dengan menggunakan AWS Identity and Access Management (). IAM Anda memberikan izin untuk mengakses AWS layanan lain dengan membuat IAM peran dengan izin yang diperlukan, lalu mengaitkan peran tersebut dengan klaster DB Anda. Untuk detail dan petunjuk tentang cara mengizinkan klaster Aurora My SQL DB Anda mengakses AWS layanan lain atas nama Anda, lihat. Mengotorisasi Amazon Aurora MySQL untuk mengakses layanan AWS lainnya atas nama Anda

## Mengotorisasi Amazon Aurora MySQL untuk mengakses layanan AWS lainnya atas nama Anda

Agar klaster DB Aurora MySQL Anda dapat mengakses layanan lainnya atas nama Anda, buat dan konfigurasikan peran AWS Identity and Access Management (IAM). Peran ini mengotorisasi

pengguna basis data di klaster DB Anda untuk mengakses layanan AWS lainnya. Untuk informasi selengkapnya, lihat Menyiapkan IAM peran untuk mengakses AWS layanan.

Anda juga harus mengonfigurasi klaster DB Aurora Anda untuk mengizinkan koneksi keluar ke layanan AWS target. Untuk informasi selengkapnya, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

Jika Anda melakukannya, pengguna basis data dapat melakukan tindakan ini menggunakan layanan AWS lainnya:

- Secara sinkron atau asinkron menginvokasi fungsi AWS Lambda menggunakan fungsi native `lambda_sync` atau `lambda_async`. Atau, secara asinkron menginvokasi fungsi AWS Lambda menggunakan prosedur `mysql.lambda_async`. Untuk informasi selengkapnya, lihat Memanggil fungsi Lambda dengan fungsi Aurora My native SQL.
- Memuat data dari file teks atau XML yang disimpan dalam bucket Amazon S3 ke dalam klaster DB Anda dengan menggunakan pernyataan `LOAD DATA FROM S3` atau `LOAD XML FROM S3`. Untuk informasi selengkapnya, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3.
- Menyimpan data dari klaster DB Anda ke dalam file teks yang disimpan dalam bucket Amazon S3 dengan menggunakan pernyataan `SELECT INTO OUTFILE S3`. Untuk informasi selengkapnya, lihat Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3.
- Mengekspor data log ke Log Amazon CloudWatch MySQL. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Application Auto Scaling. Untuk informasi selengkapnya, lihat Auto Scaling Amazon Aurora dengan Replika Aurora.

## Menyiapkan IAM peran untuk mengakses AWS layanan

Untuk mengizinkan cluster Aurora DB Anda mengakses yang lain AWS layanan, lakukan hal berikut:

1. Buat IAM kebijakan yang memberikan izin kepada AWS layanan. Untuk informasi selengkapnya, lihat topik berikut.
   - Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda
   - Membuat kebijakan IAM untuk mengakses sumber daya CloudWatch Log

- Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS

2. Buat IAM peran dan lampirkan kebijakan yang Anda buat. Untuk informasi selengkapnya, lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Kaitkan IAM peran itu dengan cluster Aurora DB Anda. Untuk informasi selengkapnya, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

### Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3

Aurora dapat mengakses sumber daya Amazon S3 untuk memuat data ke atau menyimpan data dari klaster DB Aurora. Namun, Anda harus terlebih dahulu membuat kebijakan IAM yang memberikan izin bucket dan objek yang memungkinkan Aurora mengakses Amazon S3.

Tabel berikut mencantumkan fitur Aurora yang dapat mengakses bucket Amazon S3 atas nama Anda, serta izin bucket dan objek minimum yang diperlukan setiap fitur.

| Fitur | Izin bucket | Izin objek |
|---|---|---|
| `LOAD DATA FROM S3` | `ListBucket` | `GetObject`<br>`GetObjectVersion` |
| `LOAD XML FROM S3` | `ListBucket` | `GetObject`<br>`GetObjectVersion` |
| `SELECT INTO OUTFILE S3` | `ListBucket` | `AbortMultipartUpload`<br>`DeleteObject`<br>`GetObject`<br>`ListMultipartUploadParts`<br>`PutObject` |

Kebijakan berikut menambahkan izin yang mungkin diperlukan Aurora untuk mengakses bucket Amazon S3 atas nama Anda.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

> **Note**
>
> Pastikan untuk menyertakan kedua entri tersebut untuk nilai `Resource`. Aurora memerlukan izin pada bucket itu sendiri dan semua objek di dalam bucket.
> Berdasarkan kasus penggunaan Anda, Anda mungkin tidak perlu menambahkan semua izin dalam contoh kebijakan. Selain itu, izin lain mungkin diperlukan. Misalnya, jika bucket Amazon S3 Anda dienkripsi, Anda perlu menambahkan izin `kms:Decrypt`.

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk mengakses bucket Amazon S3 atas nama Anda. Untuk mengizinkan Aurora mengakses semua bucket Amazon S3 Anda, Anda dapat melewati langkah-langkah ini dan menggunakan kebijakan IAM standar `AmazonS3ReadOnlyAccess` atau `AmazonS3FullAccess` daripada membuatnya sendiri.

Untuk membuat kebijakan IAM untuk memberikan akses ke sumber daya Amazon S3 Anda

1. Buka Konsol Manajemen IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Pada tab Editor visual, klik Pilih layanan, lalu pilih S3.
5. Untuk Tindakan, pilih Perluas semua, lalu pilih izin bucket dan izin objek yang diperlukan untuk kebijakan IAM.

   Izin objek adalah izin untuk operasi objek di Amazon S3, dan perlu diberikan untuk objek dalam bucket, bukan bucket itu sendiri. Untuk informasi selengkapnya tentang izin untuk operasi objek di Amazon S3, lihat Izin untuk operasi objek.
6. Pilih Sumber Daya, lalu pilih Tambahkan ARN untuk bucket.
7. Di kotak dialog Tambahkan ARN, berikan detail tentang sumber daya Anda, dan pilih Tambahkan.

   Tentukan ke bucket Amazon S3 mana akses akan diizinkan. Misalnya, jika Anda ingin mengizinkan Aurora mengakses bucket Amazon S3 bernama *amzn-s3-demo-bucket*, maka atur nilai Amazon Resource Name (ARN) ke `arn:aws:s3:::`*amzn-s3-demo-bucket*.
8. Jika sumber daya objek dicantumkan, pilih Tambahkan ARN untuk objek.
9. Di kotak dialog Tambahkan ARN, berikan detail tentang sumber daya Anda.

   Untuk bucket Amazon S3, tentukan ke bucket Amazon S3 mana akses akan diizinkan. Untuk objeknya, Anda dapat memilih Apa pun untuk memberikan izin ke objek apa pun di bucket.

   > **Note**
   >
   > Anda dapat mengatur Amazon Resource Name (ARN) ke nilai ARN yang lebih spesifik untuk memungkinkan Aurora hanya mengakses file atau folder tertentu dalam bucket Amazon S3. Untuk informasi selengkapnya tentang cara menentukan kebijakan akses untuk Amazon S3, lihat Mengelola izin akses ke sumber daya Amazon S3 Anda.
10. (Opsional) Pilih Tambahkan ARN untuk bucket guna menambahkan bucket Amazon S3 lainnya ke kebijakan, dan ulangi langkah sebelumnya untuk bucket.

> **Note**
>
> Anda dapat mengulanginya untuk menambahkan pernyataan izin bucket yang sesuai ke kebijakan Anda untuk setiap bucket Amazon S3 yang Anda ingin agar diakses Aurora. Secara opsional, Anda juga dapat memberikan akses ke semua bucket dan objek di Amazon S3.

11. Pilih Tinjau kebijakan.
12. Untuk Nama, masukkan nama untuk kebijakan IAM Anda, misalnya `AllowAuroraToExampleBucket`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
13. Pilih Buat kebijakan.
14. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda

Anda dapat membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk menjalankan fungsi atas nama Anda. AWS Lambda

Kebijakan berikut menambahkan izin yang diperlukan oleh Aurora untuk menginvokasi fungsi AWS Lambda atas nama Anda.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleFunction",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource":
 "arn:aws:lambda:<region>:<123456789012>:function:<example_function>"
        }
    ]
```

```
}
```

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk menjalankan fungsi atas nama Anda. AWS Lambda Untuk memungkinkan Aurora menjalankan semua AWS Lambda fungsi Anda, Anda dapat melewati langkah-langkah ini dan menggunakan `AWSLambdaRole` kebijakan yang telah ditentukan alih-alih membuat sendiri.

Untuk membuat kebijakan IAM untuk mengizinkan invokasi fungsi AWS Lambda Anda

1. Buka konsol IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Pada tab Editor visual, klik Pilih layanan, lalu pilih Lambda.
5. Untuk Tindakan, pilih Perluas semua, lalu pilih izin AWS Lambda yang diperlukan untuk kebijakan IAM.

   Pastikan bahwa `InvokeFunction` dipilih. Ini adalah izin minimum yang diperlukan untuk mengaktifkan Amazon Aurora menjalankan fungsi. AWS Lambda
6. Pilih Sumber Daya dan pilih Tambahkan ARN untuk fungsi.
7. Di kotak dialog Tambahkan ARN, berikan detail tentang sumber daya Anda.

   Tentukan ke fungsi Lambda mana akses akan diizinkan. Misalnya, jika Anda ingin mengizinkan Aurora mengakses fungsi Lambda bernama `example_function`, atur nilai ARN ke `arn:aws:lambda:::function:example_function`.

   Untuk informasi selengkapnya tentang cara menentukan kebijakan akses AWS Lambda, lihat Otentikasi dan kontrol akses untuk AWS Lambda.
8. Secara opsional, pilih Tambahkan izin tambahan untuk menambahkan AWS Lambda fungsi lain ke kebijakan, dan ulangi langkah sebelumnya untuk fungsi tersebut.

   > **Note**
   >
   > Anda dapat mengulangi ini untuk menambahkan pernyataan izin fungsi yang sesuai ke kebijakan Anda untuk setiap AWS Lambda fungsi yang Anda ingin Aurora akses.
9. Pilih Tinjau kebijakan.

10. Atur Nama untuk nama kebijakan IAM Anda, misalnya `AllowAuroraToExampleFunction`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
11. Pilih Buat kebijakan.
12. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat kebijakan IAM untuk mengakses sumber daya CloudWatch Log

Aurora dapat mengakses CloudWatch Log untuk mengekspor data log audit dari cluster Aurora DB. Namun, Anda harus terlebih dahulu membuat kebijakan IAM yang menyediakan grup log dan izin aliran log yang memungkinkan Aurora mengakses Log. CloudWatch

Kebijakan berikut menambahkan izin yang diperlukan oleh Aurora untuk mengakses Log CloudWatch Amazon atas nama Anda, dan izin minimum yang diperlukan untuk membuat grup log dan mengekspor data.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid":
 "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
```

```
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
        }
    ]
}
```

Anda dapat mengubah kebijakan ARNs dalam untuk membatasi akses ke AWS Wilayah dan akun tertentu.

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk mengakses CloudWatch Log atas nama Anda. Untuk mengizinkan Aurora akses penuh ke CloudWatch Log, Anda dapat melewati langkah-langkah ini dan menggunakan kebijakan IAM yang `CloudWatchLogsFullAccess` telah ditentukan sebelumnya alih-alih membuat sendiri. Untuk informasi selengkapnya, lihat Menggunakan kebijakan berbasis identitas (kebijakan IAM) untuk Log CloudWatch di Panduan Pengguna Amazon. CloudWatch

Untuk membuat kebijakan IAM untuk memberikan akses ke sumber daya CloudWatch Log

1. Buka konsol IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di tab Editor visual, pilih Pilih layanan, lalu pilih CloudWatch Logs.
5. Untuk Tindakan, pilih Perluas semua (di sebelah kanan), lalu pilih izin CloudWatch Log Amazon yang diperlukan untuk kebijakan IAM.

   Pastikan izin berikut dipilih:

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. Pilih Sumber Daya dan pilih Tambahkan ARN untuk grup log.
7. Di kotak dialog Tambahkan ARN, masukkan nilai berikut:

- Wilayah — AWS Wilayah atau *
- Akun – Nomor akun atau *
- Nama Grup Log – `/aws/rds/*`

8. Di kotak dialog Tambahkan ARN, pilih Tambahkan.
9. Pilih Tambahkan ARN untuk log stream.
10. Di kotak dialog Tambahkan ARN, masukkan nilai berikut:

    - Wilayah — AWS Wilayah atau *
    - Akun – Nomor akun atau *
    - Nama Grup Log – `/aws/rds/*`
    - Nama Log Stream – *

11. Di kotak dialog Tambahkan ARN, pilih Tambahkan.
12. Pilih Tinjau kebijakan.
13. Atur Nama untuk nama kebijakan IAM Anda, misalnya `AmazonRDSCloudWatchLogs`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
14. Pilih Buat kebijakan.
15. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS

Aurora dapat mengakses AWS KMS keys yang digunakan untuk mengenkripsi cadangan basis datanya. Namun, Anda harus terlebih dahulu membuat kebijakan IAM yang memberikan izin yang memungkinkan Aurora mengakses kunci KMS.

Kebijakan berikut menambahkan izin yang diperlukan Aurora untuk mengakses kunci KMS atas nama Anda.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:<region>:<123456789012>:key/<key-ID>"
        }
    ]
}
```

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan Aurora untuk mengakses kunci KMS atas nama Anda.

Untuk membuat kebijakan IAM untuk memberikan akses ke kunci KMS Anda

1. Buka konsol IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di tab Editor visual, klik Pilih layanan, lalu pilih KMS.
5. Untuk Tindakan, pilih Tulis, lalu pilih Dekripsi.
6. Pilih Sumber Daya, lalu pilih Tambahkan ARN.
7. Di kotak dialog Tambahkan ARN, masukkan nilai berikut:
   - Wilayah — Ketik AWS Wilayah, seperti `us-west-2`.
   - Akun – Ketik nomor akun pengguna.
   - Nama Log Stream – Ketik pengidentifikasi kunci KMS.
8. Di kotak dialog Tambahkan ARN, pilih Tambahkan.
9. Pilih Tinjau kebijakan.
10. Atur Nama untuk nama kebijakan IAM Anda, misalnya `AmazonRDSKMSKey`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
11. Pilih Buat kebijakan.
12. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS

Setelah membuat kebijakan IAM untuk mengizinkan Aurora mengakses sumber daya AWS, Anda harus membuat peran IAM dan melampirkan kebijakan IAM ke peran IAM baru.

Agar peran IAM mengizinkan klaster Amazon RDS Anda berkomunikasi dengan layanan AWS lainnya atas nama Anda, lakukan langkah-langkah berikut.

Agar peran IAM mengizinkan Amazon RDS mengakses layanan AWS

1. Buka konsol IAM.
2. Di panel navigasi, pilih Peran.
3. Pilih Buat peran.
4. Di bagian Layanan AWS, pilih RDS.
5. Di bagian Pilih kasus penggunaan Anda, pilih RDS – Tambahkan Peran ke Basis Data.
6. Pilih Berikutnya.
7. Di halaman Kebijakan izin, masukkan nama kebijakan Anda di bidang Cari.
8. Saat muncul di daftar, pilih kebijakan yang Anda tentukan sebelumnya menggunakan petunjuk dalam salah satu bagian berikut:
   - Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda
   - Membuat kebijakan IAM untuk mengakses sumber daya CloudWatch Log
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS
9. Pilih Berikutnya.
10. Dalam Nama peran, masukkan nama untuk peran IAM Anda, misalnya `RDSLoadFromS3`. Anda juga dapat menambahkan nilai Deskripsi opsional.
11. Pilih Buat Peran.
12. Selesaikan langkah-langkah dalam Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL

Untuk mengizinkan pengguna basis data di klaster DB Amazon Aurora untuk mengakses layanan AWS lainnya, Anda perlu mengaitkan peran IAM yang Anda buat di Membuat peran IAM untuk

mengizinkan Amazon Aurora mengakses layanan AWS dengan klaster DB tersebut. Anda juga dapat meminta AWS untuk membuat peran IAM baru dengan mengaitkan layanan secara langsung.

> **Note**
>
> Anda tidak dapat mengaitkan peran IAM dengan klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.
> Anda dapat mengaitkan peran IAM dengan klaster DB Aurora Serverless v2.

Untuk mengaitkan peran IAM dengan klaster DB, lakukan dua hal berikut:

1. Tambahkan peran ke daftar peran terkait untuk klaster DB dengan menggunakan konsol RDS, perintah add-role-to-db-cluster AWS CLI, atau operasi API RDS AddRoleToDBCluster.

   Anda dapat menambahkan maksimum lima peran IAM untuk setiap klaster DB Aurora.
2. Atur parameter tingkat klaster untuk layanan AWS terkait ke ARN untuk peran IAM terkait.

   Tabel berikut menjelaskan nama parameter tingkat klaster untuk peran IAM yang digunakan untuk mengakses layanan AWS lainnya.

| Parameter tingkat klaster | Deskripsi |
| --- | --- |
| `aws_default_lambda_role` | Digunakan saat menginvokasi fungsi Lambda dari klaster DB Anda. |
| `aws_default_logs_role` | Parameter ini tidak lagi diperlukan untuk mengekspor data log dari klaster DB Anda ke Log Amazon CloudWatch. Aurora MySQL sekarang menggunakan peran terkait layanan untuk izin yang diperlukan. Untuk mengetahui informasi selengkapnya tentang peran terkait layanan, lihat Menggunakan peran terkait layanan untuk Amazon Aurora. |
| `aws_default_s3_role` | Digunakan saat menginvokasi pernyataan `LOAD DATA FROM S3`, `LOAD XML FROM` |

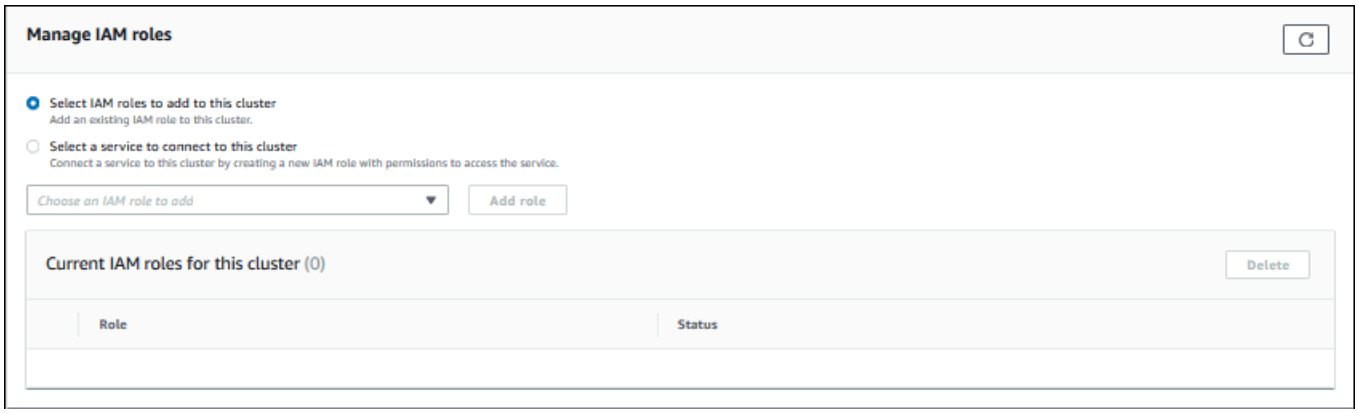
| Parameter tingkat klaster | Deskripsi | |
|---|---|---|
| | S3, atau SELECT INTO OUTFILE S3 dari klaster DB Anda.<br><br>Di Aurora MySQL versi 2, peran IAM yang ditentukan dalam parameter ini akan digunakan jika peran IAM tidak ditentukan untuk aurora_load_from_s3_role atau aurora_select_into_s3_role untuk pernyataan yang sesuai.<br><br>Di Aurora MySQL versi 3, peran IAM yang ditentukan untuk parameter ini selalu digunakan. | |
| aurora_load_from_s3_role | Digunakan saat menginvokasi pernyataan LOAD DATA FROM S3 atau LOAD XML FROM S3 dari klaster DB Anda. Jika peran IAM tidak ditentukan untuk parameter ini, peran IAM yang ditentukan dalam aws_default_s3_role akan digunakan.<br><br>Di Aurora MySQL versi 3, parameter ini tidak tersedia. | |
| aurora_select_into_s3_role | Digunakan saat menginvokasi pernyataan SELECT INTO OUTFILE S3 dari klaster DB Anda. Jika peran IAM tidak ditentukan untuk parameter ini, peran IAM yang ditentukan dalam aws_default_s3_role akan digunakan.<br><br>Di Aurora MySQL versi 3, parameter ini tidak tersedia. | |

Untuk mengaitkan peran IAM untuk mengizinkan klaster Amazon RDS Anda berkomunikasi dengan layanan AWS lain atas nama Anda, lakukan langkah-langkah berikut.

Konsol

Untuk mengaitkan peran IAM dengan klaster DB Aurora menggunakan konsol

1. Buka konsol RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih nama klaster DB Aurora yang ingin Anda kaitkan dengan peran IAM untuk menampilkan detailnya.
4. Pada tab Konektivitas & keamanan, di bagian Kelola peran IAM, lakukan salah satu hal berikut:
   - Pilih peran IAM untuk ditambahkan ke klaster ini (default)
   - Pilih layanan untuk dihubungkan ke klaster ini




5. Untuk menggunakan peran IAM yang ada, pilih peran IAM tersebut dari menu, lalu pilih Tambahkan peran.

   Jika penambahan peran berhasil, statusnya menunjukkan `Pending`, lalu `Available`.
6. Untuk menghubungkan layanan secara langsung:
   a. Klik Pilih layanan untuk dihubungkan ke klaster ini.
   b. Pilih layanannya dari menu, lalu pilih Hubungkan layanan.
   c. Untuk Hubungkan klaster ke ***`Nama Layanan`***, masukkan Amazon Resource Name (ARN) yang akan digunakan untuk terhubung ke layanan tersebut, lalu pilih Hubungkan layanan.

AWS membuat peran IAM baru untuk menghubungkan ke layanan. Statusnya menunjukkan `Pending`, lalu `Available`.

7. (Opsional) Untuk berhenti mengaitkan peran IAM dengan klaster DB dan menghapus izin terkait, pilih peran dan pilih Hapus.

Untuk mengatur parameter tingkat klaster untuk peran IAM terkait

1. Di konsol RDS, pilih Grup parameter di panel navigasi.
2. Jika Anda sudah menggunakan grup parameter DB kustom, Anda dapat memilih grup tersebut untuk digunakan daripada membuat grup parameter klaster DB baru. Jika Anda menggunakan grup parameter klaster DB default, buat grup parameter klaster DB baru, seperti yang dijelaskan di langkah-langkah berikut:

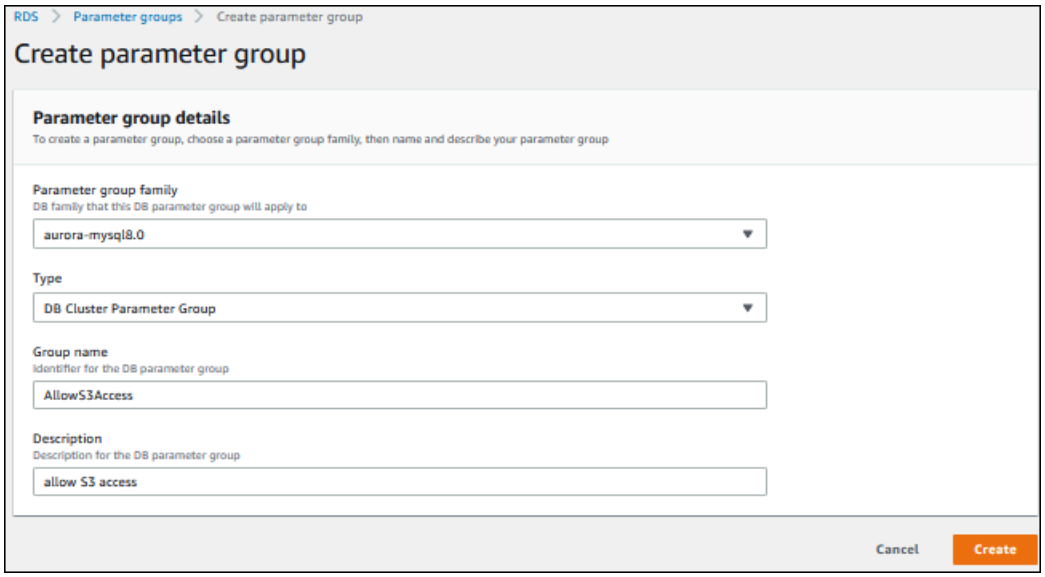
   a. Pilih Buat grup parameter.

   b. Untuk Rangkaian grup parameter, pilih `aurora-mysql8.0` untuk klaster DB yang kompatibel dengan Aurora MySQL 8.0, atau `aurora-mysql5.7` untuk klaster DB yang kompatibel dengan Aurora MySQL 5.7.

   c. Untuk Jenis, pilih Grup Parameter Klaster DB.

   d. Untuk Nama grup, ketik nama grup parameter klaster DB baru Anda.

   e. Untuk Deskripsi, ketik deskripsi untuk grup parameter klaster DB baru Anda.




   f. Pilih Buat.

3. Pada halaman Grup parameter, pilih grup parameter klaster DB Anda dan pilih Edit untuk Tindakan grup parameter.
4. Tetapkan parameter tingkat klaster yang sesuai ke nilai ARN milik peran IAM terkait.

   Misalnya, Anda hanya dapat mengatur parameter `aws_default_s3_role` ke `arn:aws:iam::123456789012:role/AllowS3Access`.
5. Pilih Simpan perubahan.
6. Untuk mengubah grup parameter klaster DB untuk klaster DB Anda, selesaikan langkah-langkah berikut:

   a. Pilih Basis data, lalu pilih klaster DB Aurora Anda.

   b. Pilih Ubah.

   c. Gulir ke Opsi basis data dan tetapkan Grup parameter klaster DB ke grup parameter klaster DB.

   d. Pilih Lanjutkan.

   e. Verifikasi perubahan Anda, lalu pilih Terapkan segera.

   f. Pilih Ubah klaster.

   g. Pilih Basis data, lalu pilih instans primer untuk klaster DB Anda.

   h. Untuk Tindakan, pilih Boot ulang.

      Saat instans di-boot ulang, peran IAM Anda dikaitkan dengan klaster DB Anda.

      Untuk informasi selengkapnya tentang grup parameter klaster, lihat Parameter konfigurasi Aurora MySQL.

CLI

Untuk mengaitkan peran IAM dengan klaster DB menggunakan AWS CLI

1. Panggil perintah `add-role-to-db-cluster` dari AWS CLI untuk menambahkan ARN untuk peran IAM Anda ke klaster DB, seperti yang ditunjukkan berikut.

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. Jika Anda menggunakan grup parameter klaster DB default, buat grup parameter klaster DB baru. Jika Anda sudah menggunakan grup parameter DB kustom, Anda dapat menggunakan grup tersebut untuk digunakan daripada membuat grup parameter klaster DB baru.

   Untuk membuat grup parameter klaster DB baru, panggil perintah `create-db-cluster-parameter-group` dari AWS CLI, seperti yang ditunjukkan berikut.

   ```
   PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
   name AllowAWSAccess \
        --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
    and AWS Lambda"
   ```

   Untuk klaster DB yang kompatibel dengan Aurora MySQL 5.7, tentukan `aurora-mysql5.7` untuk `--db-parameter-group-family`. Untuk klaster DB yang kompatibel dengan Aurora MySQL 8.0, tentukan `aurora-mysql8.0` untuk `--db-parameter-group-family`.

3. Tetapkan parameter atau parameter tingkat klaster yang sesuai dan nilai ARN peran IAM terkait di grup parameter klaster DB Anda, seperti yang ditunjukkan berikut.

   ```
   PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
    AllowAWSAccess \
       --parameters
    "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
   AllowAuroraS3Role,method=pending-reboot" \
       --parameters
    "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
   AllowAuroraLambdaRole,method=pending-reboot"
   ```

4. Ubah klaster DB untuk menggunakan grup parameter klaster DB baru lalu boot ulang klaster, seperti yang ditunjukkan berikut.

   ```
   PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
   parameter-group-name AllowAWSAccess
   PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
   ```

   Saat instans di-boot ulang, peran IAM Anda akan dikaitkan dengan klaster DB Anda.

   Untuk informasi selengkapnya tentang grup parameter klaster, lihat Parameter konfigurasi Aurora MySQL.

## Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS

Untuk menggunakan AWS layanan tertentu lainnya dengan Amazon Aurora, konfigurasi jaringan cluster Aurora DB Anda harus mengizinkan koneksi keluar ke titik akhir untuk layanan tersebut. Operasi berikut memerlukan konfigurasi jaringan ini.

- Memanggil AWS Lambda fungsi. Untuk mempelajari tentang fitur ini, lihat Memanggil fungsi Lambda dengan fungsi Aurora My native SQL.
- Mengakses file dari Amazon S3. Untuk mempelajari tentang fitur ini, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3 dan Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3.
- Mengakses titik AWS KMS akhir. AWS KMS akses diperlukan untuk menggunakan aliran aktivitas database dengan SQL Aurora My. Untuk mempelajari tentang fitur ini, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.
- Mengakses titik akhir SageMaker AI. SageMaker Akses AI diperlukan untuk menggunakan pembelajaran mesin SageMaker AI dengan Aurora My. SQL Untuk mempelajari tentang fitur ini, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL.

Aurora menampilkan pesan kesalahan berikut jika tidak dapat terhubung ke titik akhir layanan.

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

Untuk aliran aktivitas database yang menggunakan Aurora SQL My, aliran aktivitas berhenti berfungsi jika kluster DB tidak dapat mengakses AWS KMS titik akhir. Aurora memberi tahu Anda tentang masalah ini menggunakan Acara. RDS

Jika Anda menemukan pesan-pesan ini saat menggunakan AWS layanan terkait, periksa apakah klaster Aurora DB Anda bersifat publik atau pribadi. Jika klaster DB Aurora Anda bersifat privat, Anda harus mengonfigurasinya untuk mengizinkan koneksi.

Agar menjadi bersifat publik, klaster DB Aurora harus ditandai sebagai dapat diakses publik. Jika Anda melihat detail untuk cluster DB di AWS Management Console, Publicly Accessible is Yes jika ini

masalahnya. Cluster DB juga harus berada di subnet VPC publik Amazon. Lihat informasi yang lebih lengkap tentang instans basis data yang dapat diakses publik di Menggunakan klaster DB di VPC. Untuk informasi selengkapnya tentang VPC subnet Amazon publik, lihat Anda VPC dan subnet.

Jika klaster Aurora DB Anda tidak dapat diakses publik dan di subnet VPC publik, itu bersifat pribadi. Anda mungkin memiliki klaster DB yang bersifat privat dan ingin menggunakan salah satu fitur yang memerlukan konfigurasi jaringan ini. Jika demikian, konfigurasikan cluster sehingga dapat terhubung ke alamat Internet melalui Network Address Translation (NAT). Sebagai alternatif untuk Amazon S3, Amazon SageMaker AI, dan AWS Lambda, Anda dapat mengonfigurasi VPC untuk memiliki VPC titik akhir untuk layanan lain yang terkait dengan tabel rute cluster DB, lihat. Menggunakan klaster DB di VPC Untuk informasi selengkapnya tentang mengonfigurasi NAT di dalamVPC, lihat NATgateway. Untuk informasi selengkapnya tentang mengonfigurasi VPC titik akhir, lihat VPC titik akhir. Anda juga dapat membuat titik akhir gateway S3 untuk mengakses bucket S3 Anda. Untuk informasi selengkapnya, lihat Titik akhir gateway untuk Amazon S3.

Anda mungkin juga harus membuka port sementara untuk daftar kontrol akses jaringan (ACLs) dalam aturan keluar untuk grup keamanan AndaVPC. Untuk informasi selengkapnya tentang port sementara untuk jaringanACLs, lihat Port Ephemeral di Panduan Pengguna Amazon Virtual Private Cloud.

### Topik terkait

- Mengintegrasikan Aurora dengan layanan lain AWS
- Mengelola klaster DB Amazon Aurora

## Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3

Anda dapat menggunakan pernyataan `LOAD DATA FROM S3` atau `LOAD XML FROM S3` untuk memuat data dari file yang tersimpan di bucket Amazon S3. Di Aurora MySQL, file pertama kali disimpan di disk lokal, dan kemudian diimpor ke database. Setelah impor ke database selesai, file lokal dihapus.

> **Note**
>
> Memuat data ke dalam tabel dari file teks tidak didukung untuk Aurora Serverless v1. Hal ini didukung untuk Aurora Serverless v2.

Daftar Isi


- Memberi Aurora akses ke Amazon S3
- Memberikan hak istimewa untuk memuat data di Amazon Aurora My SQL
- Menentukan path (URI) ke bucket Amazon S3
- LOADDATAFROMS3
  - Sintaks
  - Parameter
  - Menggunakan manifes untuk menentukan file data yang akan dimuat
    - Memverifikasi file yang dimuat menggunakan tabel aurora_s3_load_history
  - Contoh
- LOADXMLFROMS3
  - Sintaksis
  - Parameter


## Memberi Aurora akses ke Amazon S3

Sebelum Anda dapat memuat data dari bucket Amazon S3, Anda harus terlebih dahulu memberikan izin klaster Aurora My SQL DB Anda untuk mengakses Amazon S3.

Untuk memberi Aurora SQL akses Saya ke Amazon S3

1. Buat kebijakan AWS Identity and Access Management (IAM) yang menyediakan izin bucket dan objek yang memungkinkan klaster Aurora SQL My DB Anda mengakses Amazon S3. Untuk petunjuk, silakan lihat Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3.

   > **Note**
   >
   > Di Aurora My SQL versi 3.05 dan yang lebih tinggi, Anda dapat memuat objek yang dienkripsi menggunakan yang dikelola pelanggan. AWS KMS keys Untuk melakukannya, sertakan `kms:Decrypt` izin dalam IAM kebijakan Anda. Untuk informasi selengkapnya, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS.
   > Anda tidak memerlukan izin ini untuk memuat objek yang dienkripsi menggunakan Kunci yang dikelola AWS atau kunci SSE terkelola Amazon S3 (-S3).

2. Buat IAM peran, dan lampirkan IAM kebijakan yang Anda buat Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3 ke IAM peran baru. Untuk petunjuk, silakan lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Pastikan klaster DB menggunakan grup parameter klaster DB kustom.

   Untuk informasi selengkapnya tentang membuat grup parameter klaster DB kustom, lihat Membuat grup parameter cluster DB di Amazon Aurora.
4. Untuk Aurora My SQL version 2, setel parameter cluster `aurora_load_from_s3_role` atau `aws_default_s3_role` DB ke Amazon Resource Name (ARN) dari peran baruIAM. Jika IAM peran tidak ditentukan untuk`aurora_load_from_s3_role`, Aurora menggunakan IAM peran yang ditentukan dalam. `aws_default_s3_role`

   Untuk Aurora My SQL versi 3, gunakan. `aws_default_s3_role`

   Jika klaster adalah bagian dari basis data global Aurora, atur parameter ini untuk setiap klaster Aurora dalam basis data global. Meskipun hanya klaster primer dalam basis data global Aurora yang dapat memuat data, klaster lain mungkin dipromosikan oleh mekanisme failover dan menjadi klaster primer.

   Untuk informasi lebih lanjut tentang parameter klaster DB, lihat Parameter klaster DB dan instans DB Amazon Aurora.
5. Untuk mengizinkan pengguna database di klaster Aurora My SQL DB mengakses Amazon S3, kaitkan peran yang Anda buat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan cluster DB. Untuk basis data global Aurora, kaitkan peran dengan setiap klaster Aurora di basis data global. Untuk informasi tentang mengaitkan IAM peran dengan cluster DB, lihatMengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.
6. Konfigurasikan klaster Aurora My SQL DB Anda untuk mengizinkan koneksi keluar ke Amazon S3. Untuk petunjuk, silakan lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

   Jika cluster DB Anda tidak dapat diakses publik dan di subnet VPC publik, itu bersifat pribadi. Anda dapat membuat titik akhir gateway S3 untuk mengakses bucket S3 Anda. Untuk informasi selengkapnya, lihat Titik akhir gateway untuk Amazon S3.

   Untuk basis data global Aurora, aktifkan koneksi keluar untuk setiap klaster Aurora di basis data global.

## Memberikan hak istimewa untuk memuat data di Amazon Aurora My SQL

Pengguna basis data yang mengeluarkan pernyataan `LOAD XML FROM S3` atau `LOAD DATA FROM S3` harus memiliki peran atau hak akses tertentu untuk mengeluarkan pernyataan. Di Aurora My SQL versi 3, Anda memberikan peran tersebut. `AWS_LOAD_S3_ACCESS` Di Aurora My SQL versi 2, Anda memberikan hak istimewa. `LOAD FROM S3` Pengguna administratif untuk klaster DB akan diberi peran atau hak akses yang sesuai secara default. Anda dapat memberikan hak akses kepada pengguna lain dengan menggunakan salah satu pernyataan berikut.

Gunakan pernyataan berikut untuk Aurora My SQL versi 3:

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Saat Anda menggunakan teknik peran di Aurora My SQL versi 3, Anda juga dapat mengaktifkan peran dengan menggunakan pernyataan `SET ROLE role_name` atau`SET ROLE ALL`. Jika Anda tidak terbiasa dengan sistem peran My SQL 8.0, Anda dapat mempelajari lebih lanjut diModel hak akses berbasis peran. Untuk detail selengkapnya, lihat Menggunakan peran di Manual SQL Referensi Saya.
> Hal ini hanya berlaku untuk sesi aktif saat ini. Ketika Anda terhubung kembali, Anda harus menjalankan `SET ROLE` pernyataan lagi untuk memberikan hak istimewa. Untuk informasi lebih lanjut, lihat SETROLEpernyataan di Manual SQL Referensi Saya.
> Anda dapat menggunakan parameter klaster DB `activate_all_roles_on_login` untuk mengaktifkan semua peran secara otomatis saat pengguna terhubung ke instans DB. Ketika parameter ini disetel, Anda biasanya tidak perlu memanggil `SET ROLE` pernyataan secara eksplisit untuk mengaktifkan peran. Untuk informasi selengkapnya, lihat activate_all_roles_on_login di Manual Referensi Saya. SQL
> Namun, Anda harus memanggil secara `SET ROLE ALL` eksplisit di awal prosedur tersimpan untuk mengaktifkan peran, ketika prosedur yang disimpan dipanggil oleh pengguna yang berbeda.

Gunakan pernyataan berikut untuk Aurora My SQL versi 2:

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`AWS_LOAD_S3_ACCESS`Peran dan `LOAD FROM S3` hak istimewa khusus untuk Amazon Aurora dan tidak tersedia untuk database SQL Saya eksternal RDS atau untuk SQL instans DB Saya. Jika Anda telah menyiapkan replikasi antara cluster Aurora DB sebagai sumber replikasi dan database SQL Saya sebagai klien replikasi, maka `GRANT` pernyataan untuk peran atau hak istimewa menyebabkan replikasi berhenti dengan kesalahan. Anda dapat melewati kesalahan ini dengan aman untuk melanjutkan replikasi. Untuk melewati kesalahan pada SQL instance RDS Saya, gunakan prosedur mysql_rds_skip_repl_error. Untuk melewati kesalahan pada SQL database Saya eksternal, gunakan variabel sistem slave_skip_errors (Aurora My version 2) atau variabel sistem replica_skip_errors (Aurora SQL My version 3). SQL

> **Note**
>
> Pengguna database harus memiliki `INSERT` hak istimewa untuk database di mana ia memuat data.

## Menentukan path (URI) ke bucket Amazon S3

Sintaks untuk menentukan path (URI) ke file yang disimpan di bucket Amazon S3 adalah sebagai berikut.

```
s3-region://amzn-s3-demo-bucket/file-name-or-prefix
```

Jalur tersebut mencakup nilai-nilai berikut:

- `region`(opsional) - AWS Wilayah yang berisi bucket Amazon S3 untuk dimuat. Nilai ini bersifat opsional. Jika Anda tidak menentukan nilai `region`, Aurora memuat file Anda dari Amazon S3 di wilayah yang sama dengan klaster DB Anda.
- `bucket-name` – Nama bucket Amazon S3 yang berisi data yang akan dimuat. Awalan objek yang mengidentifikasi jalur folder virtual didukung.
- `file-name-or-prefix`— Nama file atau XML file teks Amazon S3, atau awalan yang mengidentifikasi satu atau beberapa teks atau XML file yang akan dimuat. Anda juga dapat menentukan file manifes yang mengidentifikasi satu atau beberapa file teks yang akan dimuat. Untuk informasi selengkapnya tentang menggunakan file manifes untuk memuat file teks dari Amazon S3, lihat Menggunakan manifes untuk menentukan file data yang akan dimuat.

Untuk menyalin file URI for dalam bucket S3

1. Masuk ke AWS Management Console dan buka konsol Amazon S3 di. https://console.aws.amazon.com/s3/
2. Di panel navigasi, pilih Bucket, lalu pilih bucket yang ingin URI Anda salin.
3. Pilih awalan atau file yang ingin Anda muat dari S3.
4. Pilih Salin S3 URI.

## LOADDATAFROMS3

Anda dapat menggunakan `LOAD DATA FROM S3` pernyataan untuk memuat data dari format file teks apa pun yang didukung oleh SQL LOADDATAINFILEpernyataan Saya, seperti data teks yang dibatasi koma. file terkompresi tidak didukung.

> **Note**
>
> Pastikan klaster Aurora My SQL DB Anda memungkinkan koneksi keluar ke S3. Untuk informasi selengkapnya, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

Sintaks

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
```

```
    [SET col_name = expr,...]
```

> **Note**
>
> Di Aurora My SQL versi 3.05 dan lebih tinggi, kata kunci `FROM` adalah opsional.

Parameter

Pernyataan `LOAD DATA FROM S3` menggunakan parameter wajib dan opsional berikut. Anda dapat menemukan detail lebih lanjut tentang beberapa parameter ini di LOADDATAPernyataan di SQL dokumentasi Saya.

FILE | PREFIX | MANIFEST

Mengidentifikasi apakah akan memuat data dari satu file, dari semua file yang cocok dengan awalan tertentu, atau dari semua file dalam manifes yang ditentukan. `FILE` adalah opsi default.

S3- URI

Menentukan URI teks atau file manifes yang akan dimuat, atau awalan Amazon S3 yang akan digunakan. Tentukan URI menggunakan sintaks yang dijelaskan dalamMenentukan path (URI) ke bucket Amazon S3.

REPLACE | IGNORE

Menentukan tindakan yang harus diambil jika baris input memiliki nilai kunci unik yang sama dengan baris yang ada dalam tabel basis data.

- Tentukan `REPLACE` jika Anda ingin baris input menggantikan baris yang ada dalam tabel.
- Tentukan `IGNORE` jika Anda ingin membuang baris input.

INTO TABLE

Mengidentifikasi nama tabel basis data yang akan dimuati dengan baris input.

PARTITION

Mengharuskan semua baris input disisipkan ke partisi yang diidentifikasi oleh daftar nama partisi yang dipisahkan koma yang ditentukan. Jika baris input tidak dapat disisipkan ke salah satu partisi yang ditentukan, maka pernyataan ini akan gagal dan kesalahan akan ditampilkan.

CHARACTER SET

Mengidentifikasi kumpulan karakter data dalam file input.

FIELDS | COLUMNS

Mengidentifikasi cara bidang atau kolom dalam file input dibatasi. Bidang dibatasi tab secara default.

LINES

Mengidentifikasi cara baris dalam file input dibatasi. Baris dibatasi oleh karakter baris baru (`'\n'`) secara default.

IGNORE ***number*** LINES | ROWS

Menentukan untuk mengabaikan jumlah tertentu baris di awal file input. Misalnya, Anda dapat menggunakan `IGNORE 1 LINES` untuk melewati baris header awal yang berisi nama kolom, atau `IGNORE 2 ROWS` untuk melewati dua baris pertama data dalam file input. Jika Anda juga menggunakan `PREFIX`, `IGNORE` akan melewati jumlah tertentu baris di awal file input pertama.

col_name_or_user_var, ...

– Menentukan daftar yang dipisahkan koma yang berisi satu atau beberapa nama kolom atau variabel pengguna yang mengidentifikasi kolom mana yang akan dimuat berdasarkan nama. Nama variabel pengguna yang digunakan untuk tujuan ini harus cocok dengan nama elemen dari file teks, yang diawali dengan @. Anda dapat menggunakan variabel pengguna untuk menyimpan nilai bidang terkait untuk digunakan kembali nanti.

Misalnya, pernyataan berikut memuat kolom pertama dari file input ke kolom pertama `table1`, dan menetapkan nilai kolom `table_column2` di `table1` ke nilai input kolom kedua dibagi 100.

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

Menentukan daftar operasi penetapan yang dipisahkan koma yang mengatur nilai kolom dalam tabel ke nilai yang tidak disertakan dalam file input.

Misalnya, pernyataan berikut mengatur dua kolom pertama `table1` ke nilai di dua kolom pertama dari file input, lalu mengatur nilai `column3` di `table1` ke stempel waktu saat ini.

```
LOAD DATA FROM S3  's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, column2)
```

```
    SET column3 = CURRENT_TIMESTAMP;
```

Anda dapat menggunakan subkueri di sisi kanan penetapan SET. Untuk subkueri yang menampilkan nilai yang akan ditetapkan ke kolom, Anda hanya dapat menggunakan subkueri skalar. Selain itu, Anda tidak dapat menggunakan subkueri untuk memilih dari tabel yang sedang dimuat.

Anda tidak dapat menggunakan kata kunci LOCAL dari pernyataan LOAD DATA FROM S3 jika memuat data dari bucket Amazon S3.

Menggunakan manifes untuk menentukan file data yang akan dimuat

Anda dapat menggunakan LOAD DATA FROM S3 pernyataan dengan MANIFEST kata kunci untuk menentukan file manifes dalam JSON format yang mencantumkan file teks yang akan dimuat ke dalam tabel di cluster DB Anda.

JSONSkema berikut menjelaskan format dan isi file manifes.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
                        "maxLength": 1024,
                        "minLength": 1,
                        "type": "string"
                    }
```

```
                },
                "required": [
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

Masing-masing `url` dalam manifes harus menentukan a URL dengan nama bucket dan path objek lengkap untuk file tersebut, bukan hanya awalan. Anda dapat menggunakan manifes untuk memuat file dari bucket yang berbeda-beda, wilayah yang berbeda-beda, atau file yang tidak memiliki awalan yang sama. Jika suatu wilayah tidak ditentukan dalamURL, wilayah cluster DB Aurora target digunakan. Contoh berikut menunjukkan file manifes yang memuat empat file dari bucket yang berbeda.

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
  ]
}
```

Flag `mandatory` opsional menentukan apakah `LOAD DATA FROM S3` harus menampilkan kesalahan jika file tidak ditemukan. Flag `mandatory` diatur secara default ke `false`. Terlepas dari cara `mandatory` diatur, `LOAD DATA FROM S3` akan diterminasi jika tidak ada file yang ditemukan.

File manifes dapat memiliki ekstensi. Contoh berikut menjalankan pernyataan `LOAD DATA FROM S3` dengan manifes di contoh sebelumnya, yang bernama **`customer.manifest`**.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

Setelah pernyataan selesai, entri untuk setiap file yang berhasil dimuat akan ditulis ke tabel `aurora_s3_load_history`.

Memverifikasi file yang dimuat menggunakan tabel aurora_s3_load_history

Setiap pernyataan `LOAD DATA FROM S3` yang berhasil akan memperbarui tabel `aurora_s3_load_history` dalam skema `mysql` dengan entri untuk setiap file yang dimuat.

Setelah Anda menjalankan pernyataan `LOAD DATA FROM S3`, Anda dapat memverifikasi file mana yang dimuat dengan mengueri tabel `aurora_s3_load_history`. Untuk melihat file yang dimuat dari satu iterasi pernyataan, gunakan `WHERE` klausa untuk memfilter catatan di Amazon URI S3 untuk file manifes yang digunakan dalam pernyataan. Jika Anda telah menggunakan file manifes yang sama sebelumnya, filter hasilnya menggunakan bidang `timestamp`.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

Tabel berikut menjelaskan bidang dalam tabel `aurora_s3_load_history`.

| Bidang | Deskripsi |
| --- | --- |
| `load_prefix` | URIYang ditentukan dalam pernyataan beban. Ini URI dapat memetakan ke salah satu dari berikut ini:<br>• File data tunggal untuk pernyataan `LOAD DATA FROM S3 FILE` |

| Bidang | Deskripsi |
| --- | --- |
| | • Awalan Amazon S3 yang dipetakan ke beberapa file data untuk pernyataan `LOAD DATA FROM S3 PREFIX`<br>• File manifes tunggal yang berisi nama file yang akan dimuat untuk pernyataan `LOAD DATA FROM S3 MANIFEST` |
| `file_name` | Nama file yang dimuat ke Aurora dari Amazon S3 menggunakan yang diidentifikasi URI di lapangan. `load_prefix` |
| `version_number` | Nomor versi file yang diidentifikasi oleh bidang `file_name` yang dimuat, jika bucket Amazon S3 memiliki nomor versi. |
| `bytes_loaded` | Ukuran file yang dimuat, dalam byte. |
| `load_timestamp` | Stempel waktu saat pernyataan `LOAD DATA FROM S3` selesai. |

Contoh

Pernyataan berikut memuat data dari bucket Amazon S3 yang berada di wilayah yang sama dengan klaster DB Aurora. Pernyataan tersebut membaca data yang dibatasi koma dalam file `customerdata.txt` yang ada di *`amzn-s3-demo-bucket`* Bucket Amazon S3, lalu memuat data ke dalam tabel. `store-schema.customer-table`

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

Pernyataan berikut memuat data dari bucket Amazon S3 yang berada di wilayah yang berbeda dari klaster DB Aurora. Pernyataan tersebut membaca data yang dibatasi koma dari semua file yang cocok dengan awalan objek di `employee-data` *`amzn-s3-demo-bucket`* Amazon S3 bucket di `us-west-2` wilayah tersebut, lalu memuat data ke dalam tabel. `employees`

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://amzn-s3-demo-bucket/employee_data'
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
```

```
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

Pernyataan berikut memuat data dari file yang ditentukan dalam file JSON manifes bernama q1_sales.json ke dalam tabel. `sales`

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://amzn-s3-demo-bucket1/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (MONTH, STORE, GROSS, NET);
```

## LOADXMLFROMS3

Anda dapat menggunakan `LOAD XML FROM S3` pernyataan untuk memuat data dari XML file yang disimpan di bucket Amazon S3 dalam salah satu dari tiga format berbedaXML:

- Nama kolom sebagai atribut elemen `<row>`. Nilai atribut mengidentifikasi konten bidang tabel.

  ```
  <row column1="value1" column2="value2" .../>
  ```

- Nama kolom sebagai elemen turunan dari elemen `<row>`. Nilai elemen turunan mengidentifikasi konten bidang tabel.

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- Nama kolom di atribut `name` dari elemen `<field>` dalam elemen `<row>`. Nilai elemen `<field>` mengidentifikasi konten bidang tabel.

  ```
  <row>
    <field name='column1'>value1</field>
    <field name='column2'>value2</field>
  </row>
  ```

### Sintaksis

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
```

```
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Parameter

Pernyataan `LOAD XML FROM S3` menggunakan parameter wajib dan opsional berikut. Anda dapat menemukan detail lebih lanjut tentang beberapa parameter ini di LOADXMLPernyataan di SQL dokumentasi Saya.

FILE | PREFIX

Mengidentifikasi apakah akan memuat data dari satu file, atau dari semua file yang cocok dengan awalan yang diberikan. `FILE` adalah opsi default.

REPLACE | IGNORE

Menentukan tindakan yang harus diambil jika baris input memiliki nilai kunci unik yang sama dengan baris yang ada dalam tabel basis data.

- Tentukan `REPLACE` jika Anda ingin baris input menggantikan baris yang ada dalam tabel.
- Tentukan `IGNORE` jika Anda ingin membuang baris input. `IGNORE` adalah opsi default.

INTO TABLE

Mengidentifikasi nama tabel basis data yang akan dimuati dengan baris input.

PARTITION

Mengharuskan semua baris input disisipkan ke partisi yang diidentifikasi oleh daftar nama partisi yang dipisahkan koma yang ditentukan. Jika baris input tidak dapat disisipkan ke salah satu partisi yang ditentukan, maka pernyataan ini akan gagal dan kesalahan akan ditampilkan.

CHARACTER SET

Mengidentifikasi kumpulan karakter data dalam file input.

ROWSIDENTIFIEDOLEH

Mengidentifikasi nama elemen yang mengidentifikasi baris dalam file input. Default-nya adalah `<row>`.

IGNORE *number* LINES | ROWS

Menentukan untuk mengabaikan jumlah tertentu baris di awal file input. Misalnya, Anda dapat menggunakan `IGNORE 1 LINES` untuk melewati baris pertama dalam file teks, atau `IGNORE 2 ROWS` untuk melewati dua baris pertama data dalam inputXML.

field_name_or_user_var, ...

Menentukan daftar dipisahkan koma dari satu atau lebih nama XML elemen atau variabel pengguna yang mengidentifikasi elemen untuk memuat dengan nama. Nama variabel pengguna yang digunakan untuk tujuan ini harus sesuai dengan nama elemen dari XML file, diawali dengan @. Anda dapat menggunakan variabel pengguna untuk menyimpan nilai bidang terkait untuk digunakan kembali nanti.

Misalnya, pernyataan berikut memuat kolom pertama dari file input ke kolom pertama `table1`, dan menetapkan nilai kolom `table_column2` di `table1` ke nilai input kolom kedua dibagi 100.

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
   SET table_column2 = @var1/100;
```

SET

Menentukan daftar operasi penetapan yang dipisahkan koma yang mengatur nilai kolom dalam tabel ke nilai yang tidak disertakan dalam file input.

Misalnya, pernyataan berikut mengatur dua kolom pertama `table1` ke nilai di dua kolom pertama dari file input, lalu mengatur nilai `column3` di `table1` ke stempel waktu saat ini.

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, column2)
   SET column3 = CURRENT_TIMESTAMP;
```

Anda dapat menggunakan subkueri di sisi kanan penetapan `SET`. Untuk subkueri yang menampilkan nilai yang akan ditetapkan ke kolom, Anda hanya dapat menggunakan subkueri skalar. Selain itu, Anda tidak dapat menggunakan subkueri untuk memilih dari tabel yang sedang dimuat.

# Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3

Anda dapat menggunakan `SELECT INTO OUTFILE S3` pernyataan untuk menanyakan data dari kluster Amazon Aurora My SQL DB dan menyimpannya ke dalam file teks yang disimpan dalam bucket Amazon S3. Di Aurora MySQL, file pertama kali disimpan di disk lokal, dan kemudian diekspor ke S3. Setelah ekspor selesai, file lokal dihapus.

Anda dapat mengenkripsi bucket Amazon S3 menggunakan kunci terkelola Amazon S3 SSE (-S3) AWS KMS key atau SSE (KMS- Kunci yang dikelola AWS : atau kunci yang dikelola pelanggan).

`LOAD DATA FROM S3`Pernyataan tersebut dapat menggunakan file yang dibuat oleh `SELECT INTO OUTFILE S3` pernyataan untuk memuat data ke dalam cluster Aurora DB. Untuk informasi selengkapnya, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3.

> **Note**
>
> Fitur ini tidak didukung untuk klaster DB Aurora Serverless v1. Hal ini didukung untuk klaster DB Aurora Serverless v2.
> Anda juga dapat menyimpan data cluster DB dan data snapshot cluster DB ke Amazon S3 menggunakan AWS Management Console AWS CLI,, atau Amazon. RDS API Untuk informasi selengkapnya, silakan lihat Mengekspor data klaster DB ke Amazon S3 dan Mengekspor data snapshot klaster DB ke Amazon S3.

Daftar Isi


- Memberikan Aurora SQL akses Saya ke Amazon S3
- Memberikan hak istimewa untuk menyimpan data di Aurora My SQL
- Menentukan jalur ke bucket Amazon S3
- Membuat manifes untuk menampilkan daftar file data
- SELECTINTOOUTFILES3
  - Sintaks
  - Parameter
  - Pertimbangan
  - Contoh

## Memberikan Aurora SQL akses Saya ke Amazon S3

Sebelum Anda dapat menyimpan data ke dalam bucket Amazon S3, Anda harus terlebih dahulu memberikan izin cluster Aurora My SQL DB Anda untuk mengakses Amazon S3.

Untuk memberi Aurora SQL akses Saya ke Amazon S3

1. Buat kebijakan AWS Identity and Access Management (IAM) yang menyediakan izin bucket dan objek yang memungkinkan klaster Aurora SQL My DB Anda mengakses Amazon S3. Untuk petunjuk, silakan lihat Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3.

   > **Note**
   >
   > Di Aurora My SQL versi 3.05 dan yang lebih tinggi, Anda dapat mengenkripsi objek menggunakan AWS KMS kunci yang dikelola pelanggan. Untuk melakukannya, sertakan `kms:GenerateDataKey` izin dalam IAM kebijakan Anda. Untuk informasi selengkapnya, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS. Anda tidak memerlukan izin ini untuk mengenkripsi objek menggunakan Kunci yang dikelola AWS atau kunci terkelola Amazon S3 SSE (-S3).

2. Buat IAM peran, dan lampirkan IAM kebijakan yang Anda buat Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3 ke IAM peran baru. Untuk petunjuk, silakan lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

3. Untuk Aurora My SQL version 2, setel parameter cluster `aurora_select_into_s3_role` atau `aws_default_s3_role` DB ke Amazon Resource Name (ARN) dari peran baruIAM. Jika IAM peran tidak ditentukan untuk`aurora_select_into_s3_role`, Aurora menggunakan IAM peran yang ditentukan dalam. `aws_default_s3_role`

   Untuk Aurora My SQL versi 3, gunakan. `aws_default_s3_role`

   Jika klaster adalah bagian dari basis data global Aurora, atur parameter ini untuk setiap klaster Aurora dalam basis data global.

   Untuk informasi selengkapnya tentang parameter klaster DB, lihat Parameter klaster DB dan instans DB Amazon Aurora.

4. Untuk mengizinkan pengguna database di klaster Aurora My SQL DB mengakses Amazon S3, kaitkan peran yang Anda buat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan cluster DB.

Untuk basis data global Aurora, kaitkan peran dengan setiap klaster Aurora di basis data global.

Untuk informasi tentang mengaitkan IAM peran dengan cluster DB, lihatMengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

5. Konfigurasikan klaster Aurora My SQL DB Anda untuk mengizinkan koneksi keluar ke Amazon S3. Untuk petunjuk, silakan lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

   Untuk basis data global Aurora, aktifkan koneksi keluar untuk setiap klaster Aurora di basis data global.

## Memberikan hak istimewa untuk menyimpan data di Aurora My SQL

Pengguna basis data yang mengeluarkan pernyataan `SELECT INTO OUTFILE S3` harus memiliki peran atau hak akses tertentu. Di Aurora My SQL versi 3, Anda memberikan peran tersebut. `AWS_SELECT_S3_ACCESS` Di Aurora My SQL versi 2, Anda memberikan hak istimewa. `SELECT INTO S3` Pengguna administratif untuk klaster DB akan diberi peran atau hak akses yang sesuai secara default. Anda dapat memberikan hak akses kepada pengguna lain dengan menggunakan salah satu pernyataan berikut.

Gunakan pernyataan berikut untuk Aurora My SQL versi 3:

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Saat Anda menggunakan teknik peran di Aurora My SQL versi 3, Anda juga dapat mengaktifkan peran dengan menggunakan pernyataan `SET ROLE role_name` atau`SET ROLE ALL`. Jika Anda tidak terbiasa dengan sistem peran My SQL 8.0, Anda dapat mempelajari lebih lanjut diModel hak akses berbasis peran. Untuk detail selengkapnya, lihat Menggunakan peran di Manual SQL Referensi Saya.
> Hal ini hanya berlaku untuk sesi aktif saat ini. Ketika Anda terhubung kembali, Anda harus menjalankan `SET ROLE` pernyataan lagi untuk memberikan hak istimewa. Untuk informasi selengkapnya, lihat SETROLEpernyataan di Manual SQL Referensi Saya.
> Anda dapat menggunakan parameter klaster DB `activate_all_roles_on_login` untuk mengaktifkan semua peran secara otomatis saat pengguna terhubung ke instans DB. Ketika parameter ini disetel, Anda biasanya tidak perlu memanggil `SET ROLE`

pernyataan secara eksplisit untuk mengaktifkan peran. Untuk informasi selengkapnya, lihat activate_all_roles_on_login di Manual Referensi Saya. SQL
Namun, Anda harus memanggil secara `SET ROLE ALL` eksplisit di awal prosedur tersimpan untuk mengaktifkan peran, ketika prosedur yang disimpan dipanggil oleh pengguna yang berbeda.

Gunakan pernyataan berikut untuk Aurora My SQL versi 2:

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`AWS_SELECT_S3_ACCESS`Peran dan `SELECT INTO S3` hak istimewa khusus untuk Amazon Aurora SQL My dan tidak tersedia untuk database Saya RDS atau untuk SQL SQL instans DB Saya. Jika Anda telah menyiapkan replikasi antara cluster Aurora SQL My DB sebagai sumber replikasi dan database SQL Saya sebagai klien replikasi, maka `GRANT` pernyataan untuk peran atau hak istimewa menyebabkan replikasi berhenti dengan kesalahan. Anda dapat melewati kesalahan ini dengan aman untuk melanjutkan replikasi. Untuk melewati kesalahan pada instance RDS for My SQL DB, gunakan prosedur mysql_rds_skip_repl_error. Untuk melewati kesalahan pada SQL database Saya eksternal, gunakan variabel sistem slave_skip_errors (Aurora My version 2) atau variabel sistem replica_skip_errors (Aurora SQL My version 3). SQL

### Menentukan jalur ke bucket Amazon S3

Sintaksis untuk menentukan jalur untuk menyimpan data dan file manifes di bucket Amazon S3 serupa dengan yang digunakan dalam pernyataan `LOAD DATA FROM S3 PREFIX`, seperti yang ditunjukkan berikut ini.

```
s3-region://bucket-name/file-prefix
```

Jalur tersebut mencakup nilai-nilai berikut:

- `region`(opsional) - AWS Wilayah yang berisi bucket Amazon S3 untuk menyimpan data. Nilai ini bersifat opsional. Jika Anda tidak menentukan nilai `region`, Aurora akan menyimpan file Anda ke Amazon S3 di wilayah yang sama dengan klaster DB Anda.
- `bucket-name` – Nama bucket Amazon S3 untuk menyimpan data. Awalan objek yang mengidentifikasi jalur folder virtual didukung.
- `file-prefix` – Awalan objek Amazon S3 yang mengidentifikasi file yang akan disimpan di Amazon S3.

Berkas data yang dibuat oleh pernyataan `SELECT INTO OUTFILE S3` menggunakan jalur berikut, yang *`00000`* mewakili bilangan bulat 5 digit berbasis nol.

```
s3-region://bucket-name/file-prefix.part_00000
```

Misalnya, pernyataan `SELECT INTO OUTFILE S3` menetapkan `s3-us-west-2://bucket/prefix` sebagai jalur untuk menyimpan berkas data dan membuat tiga berkas data. Bucket Amazon S3 yang ditentukan berisi file data berikut.

- s3-us-west-2://bucket/prefix.part_00000
- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

## Membuat manifes untuk menampilkan daftar file data

Anda dapat menggunakan `SELECT INTO OUTFILE S3` pernyataan dengan `MANIFEST ON` opsi untuk membuat file manifes dalam JSON format yang mencantumkan file teks yang dibuat oleh pernyataan. `LOAD DATA FROM S3`Pernyataan tersebut dapat menggunakan file manifes untuk memuat file data kembali ke cluster Aurora My SQL DB. Untuk informasi selengkapnya tentang penggunaan manifes untuk memuat file data dari Amazon S3 ke dalam klaster Aurora My SQL DB, lihat. Menggunakan manifes untuk menentukan file data yang akan dimuat

File data yang disertakan dalam manifes yang dibuat oleh pernyataan `SELECT INTO OUTFILE S3` akan dicantumkan dalam urutan pembuatannya oleh pernyataan tersebut. Misalnya, pernyataan `SELECT INTO OUTFILE S3` menetapkan `s3-us-west-2://bucket/prefix` sebagai jalur untuk menyimpan file data serta membuat tiga file data dan sebuah file manifes. Bucket Amazon S3 yang ditentukan berisi file manifes bernama `s3-us-west-2://bucket/prefix.manifest`, yang berisi informasi berikut.

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
```

```
    }
  ]
}
```

## SELECTINTOOUTFILES3

Anda dapat menggunakan pernyataan `SELECT INTO OUTFILE S3` untuk meminta data dari klaster DB dan menyimpannya secara langsung ke dalam file teks yang dibatasi yang tersimpan di bucket Amazon S3.

File terkompresi tidak didukung. File terenkripsi didukung mulai dari Aurora My versi 2.09.0. SQL

Sintaks

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
        [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
```

```
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

Parameter

Pernyataan `SELECT INTO OUTFILE S3` menggunakan parameter wajib dan opsional berikut yang khusus untuk Aurora.

s3-uri

Menentukan URI untuk awalan Amazon S3 untuk digunakan. Gunakan sintaks yang dijelaskan dalam Menentukan jalur ke bucket Amazon S3.

FORMAT {CSV|TEXT} [HEADER]

Secara opsional menyimpan data dalam CSV format.

`TEXT`Opsi ini adalah default dan menghasilkan format SQL ekspor Saya yang ada.

Opsi `CSV` menghasilkan nilai data yang dipisahkan koma. CSVFormatnya mengikuti spesifikasi di RFC-4180. Jika Anda menentukan kata kunci opsional `HEADER`, file output akan berisi satu baris header. Label di baris header sesuai dengan nama kolom dari pernyataan `SELECT`. Anda dapat menggunakan CSV file untuk melatih model data untuk digunakan dengan layanan AWS ML. Untuk informasi selengkapnya tentang penggunaan data Aurora yang diekspor AWS dengan layanan ML, lihat. Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan)

MANIFEST{PADA |OFF}

Menunjukkan apakah file manifes dibuat di Amazon S3. File manifes adalah file JavaScript Object Notation (JSON) yang dapat digunakan untuk memuat data ke dalam cluster Aurora DB dengan `LOAD DATA FROM S3 MANIFEST` pernyataan tersebut. Untuk informasi selengkapnya tentang `LOAD DATA FROM S3 MANIFEST`, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3.

Jika `MANIFEST ON` ditentukan dalam kueri, file manifes dibuat di Amazon S3 setelah semua file data dibuat dan diunggah. file manifes dibuat menggunakan jalur berikut:

```
s3-region://bucket-name/file-prefix.manifest
```

Untuk informasi selengkapnya tentang format konten file manifes, lihat Membuat manifes untuk menampilkan daftar file data.

OVERWRITE{PADA |OFF}

Menunjukkan apakah file yang ada di bucket Amazon S3 yang ditentukan akan ditimpa. Jika `OVERWRITE ON` ditentukan, file yang ada yang cocok dengan awalan file dalam yang URI ditentukan dalam akan `s3-uri` ditimpa. Jika tidak, kesalahan akan muncul.

ENCRYPTION{PADA | OFF | SSE _S3 | SSE _ KMS ['*cmk_id*']}

Menunjukkan apakah akan menggunakan enkripsi sisi server dengan kunci terkelola Amazon S3 (SSE-S3) atau AWS KMS keys (SSE-KMS, termasuk dan kunci yang dikelola pelanggan). Kunci yang dikelola AWS `SSE_KMS`Pengaturan `SSE_S3` dan tersedia di Aurora My SQL versi 3.05 dan lebih tinggi.

Anda juga dapat menggunakan variabel sesi `aurora_select_into_s3_encryption_default` bukan klausa `ENCRYPTION`, seperti yang ditunjukkan pada contoh berikut. Gunakan salah satu SQL klausa atau variabel sesi, tetapi tidak keduanya.

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

`SSE_KMS`Pengaturan `SSE_S3` dan tersedia di Aurora My SQL versi 3.05 dan lebih tinggi.

Jika Anda mengatur `aurora_select_into_s3_encryption_default` ke nilai berikut:

- `OFF` – Kebijakan enkripsi default bucket S3 akan diikuti. Nilai default `aurora_select_into_s3_encryption_default` adalah `OFF`.
- `ON`atau `SSE_S3` — Objek S3 dienkripsi menggunakan kunci terkelola Amazon S3 (-S3). SSE
- `SSE_KMS`— Objek S3 dienkripsi menggunakan file. AWS KMS key

  Dalam hal ini, Anda juga menyertakan variabel sesi `aurora_s3_default_cmk_id`, misalnya:

  ```
  set session aurora_select_into_s3_encryption_default={SSE_KMS};
  set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
  ```

  - Saat `aurora_s3_default_cmk_id` bernilai `NULL`, objek S3 dienkripsi menggunakan Kunci yang dikelola AWS.

- Jika `aurora_s3_default_cmk_id` berupa string `cmk_id` yang tidak kosong, objek S3 dienkripsi menggunakan kunci yang dikelola pelanggan.

  Nilai `cmk_id` tidak boleh berupa string kosong.

Saat Anda menggunakan perintah `SELECT INTO OUTFILE S3`, Aurora menentukan enkripsi sebagai berikut:

- Jika `ENCRYPTION` klausa hadir dalam SQL perintah, Aurora hanya bergantung pada nilai`ENCRYPTION`, dan tidak menggunakan variabel sesi.
- Jika klausa `ENCRYPTION` tidak ada, Aurora mengandalkan nilai variabel sesi.

Untuk informasi selengkapnya, lihat Menggunakan enkripsi sisi server dengan kunci terkelola Amazon S3 (SSE-S3) dan Menggunakan enkripsi sisi server dengan kunci AWS KMS (-SSE) KMS di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Anda dapat menemukan detail lebih lanjut tentang parameter lain dalam SELECTLOADDATApernyataan dan pernyataan, di SQL dokumentasi Saya.

Pertimbangan

Jumlah file yang ditulis ke bucket Amazon S3 tergantung pada jumlah data yang dipilih oleh `SELECT INTO OUTFILE S3` pernyataan dan ambang batas ukuran file untuk Aurora My. SQL Ambang batas ukuran file default adalah 6 gigabyte (GB). Jika data yang dipilih oleh pernyataan kurang dari ambang batas ukuran file, satu file akan dibuat; jika tidak, banyak file akan dibuat. Pertimbangan lain untuk file yang dibuat oleh pernyataan ini mencakup hal-hal berikut:

- Aurora My SQL menjamin bahwa baris dalam file data tidak dibagi melintasi batas file. Untuk banyak file, ukuran setiap file data kecuali yang terakhir biasanya hampir sama dengan ambang batas ukuran file. Namun, terkadang jika ambang batas ukuran file tidak tercapai, akibatnya baris akan terbagi menjadi dua file data. Dalam hal ini, Aurora My SQL membuat file data yang menjaga baris tetap utuh, tetapi mungkin lebih besar dari ambang ukuran file.
- Karena setiap `SELECT` pernyataan di Aurora My SQL berjalan sebagai transaksi atom, `SELECT INTO OUTFILE S3` pernyataan yang memilih kumpulan data besar mungkin berjalan selama beberapa waktu. Jika pernyataan gagal karena alasan apa pun, Anda mungkin perlu memulai kembali dan mengeluarkan pernyataan tersebut lagi. Namun, jika pernyataan gagal, file yang sudah diunggah ke Amazon S3 tetap berada di bucket Amazon S3 yang ditentukan. Anda dapat menggunakan pernyataan lain untuk mengunggah data yang tersisa daripada memulai dari awal lagi.

- Jika jumlah data yang akan dipilih berukuran besar (lebih dari 25 GB), kami menyarankan Anda menggunakan beberapa pernyataan `SELECT INTO OUTFILE S3` untuk menyimpan data ke Amazon S3. Setiap pernyataan harus memilih bagian data yang berbeda untuk disimpan, dan juga menentukan `file_prefix` yang berbeda dalam parameter `s3-uri` untuk digunakan saat menyimpan file data. Dengan mempartisi data yang akan dipilih menggunakan beberapa pernyataan, kesalahan di satu pernyataan akan lebih mudah untuk dipulihkan. Jika terjadi kesalahan untuk satu pernyataan, hanya sebagian data yang perlu dipilih kembali dan diunggah ke Amazon S3. Penggunaan banyak pernyataan juga membantu menghindari satu transaksi yang berjalan lama, yang dapat meningkatkan performa.
- Jika beberapa pernyataan `SELECT INTO OUTFILE S3` yang menggunakan `file_prefix` yang sama di parameter `s3-uri` dijalankan secara paralel untuk memilih data yang diunggah ke dalam Amazon S3, perilakunya tidak akan ditentukan.
- Metadata, seperti skema tabel atau metadata file, tidak diunggah oleh Aurora My ke Amazon S3. SQL
- Dalam beberapa kasus, Anda mungkin perlu menjalankan kembali kueri `SELECT INTO OUTFILE S3`, seperti untuk memulihkan dari kegagalan. Dalam kasus ini, Anda harus menghapus file data apa pun yang ada di bucket Amazon S3 dengan awalan file yang sama yang ditentukan di `s3-uri`, atau menyertakan `OVERWRITE ON` di kueri `SELECT INTO OUTFILE S3`.

`SELECT INTO OUTFILE S3`Pernyataan tersebut mengembalikan nomor SQL kesalahan saya yang khas dan respons pada keberhasilan atau kegagalan. Jika Anda tidak memiliki akses ke nomor SQL kesalahan dan respons saya, cara termudah untuk menentukan kapan selesai adalah dengan menentukan `MANIFEST ON` dalam pernyataan. File manifes adalah file terakhir yang ditulis oleh pernyataan tersebut. Dengan kata lain, jika Anda memiliki file manifes, berarti pernyataan tersebut telah selesai.

Saat ini, tidak ada cara untuk memantau secara langsung progres pernyataan `SELECT INTO OUTFILE S3` saat dijalankan. Namun, misalkan Anda menulis sejumlah besar data dari Aurora My SQL ke Amazon S3 menggunakan pernyataan ini, dan Anda tahu ukuran data yang dipilih oleh pernyataan tersebut. Dalam kasus ini, Anda dapat memperkirakan progresnya dengan memantau pembuatan file data di Amazon S3.

Untuk melakukannya, Anda dapat menggunakan fakta bahwa sebuah file data akan dibuat di bucket Amazon S3 yang ditentukan untuk setiap 6 GB data yang dipilih oleh pernyataan tersebut. Bagilah ukuran data yang dipilih dengan 6 GB untuk mendapatkan perkiraan jumlah file data yang akan

dibuat. Anda kemudian dapat memperkirakan progres pernyataan dengan memantau jumlah file yang diunggah ke Amazon S3 saat pernyataan berjalan.

Contoh

Pernyataan berikut memilih semua data dalam `employees` tabel dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah berbeda dari cluster Aurora SQL My DB. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`). Pernyataan tersebut menampilkan kesalahan jika file yang cocok dengan awalan file `sample_employee_data` ada di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

Pernyataan berikut memilih semua data dalam `employees` tabel dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang sama dengan cluster Aurora SQL My DB. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`), serta membuat juga sebuah file manifes. Pernyataan tersebut menampilkan kesalahan jika file yang cocok dengan awalan file `sample_employee_data` ada di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

Pernyataan berikut memilih semua data di tabel `employees` dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang berbeda dari klaster DB Aurora. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`). Pernyataan tersebut menimpa file apa pun yang ada yang cocok dengan awalan file `sample_employee_data` di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
```

```
    OVERWRITE ON;
```

Pernyataan berikut memilih semua data dalam `employees` tabel dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang sama dengan cluster Aurora SQL My DB. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`), serta membuat juga sebuah file manifes. Pernyataan tersebut menimpa file apa pun yang ada yang cocok dengan awalan file `sample_employee_data` di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

## Memanggil fungsi Lambda dari cluster Amazon Aurora My DB SQL

Anda dapat memanggil AWS Lambda berfungsi dari cluster DB Amazon Aurora My SQL -Compatible Edition dengan fungsi asli atau. `lambda_sync` `lambda_async` Sebelum menjalankan fungsi Lambda dari Aurora MySQL, cluster Aurora DB harus memiliki akses ke Lambda. Untuk detail tentang pemberian akses ke Aurora SQL My, lihat. Memberi Aurora akses ke Lambda Untuk informasi tentang fungsi tersimpan `lambda_sync` dan `lambda_async`, lihat Memanggil fungsi Lambda dengan fungsi Aurora My native SQL.

Anda juga dapat memanggil AWS Lambda berfungsi dengan menggunakan prosedur tersimpan. Namun, penggunaan prosedur tersimpan sudah dihentikan. Kami sangat menyarankan menggunakan fungsi Aurora My SQL native jika Anda menggunakan salah satu versi Aurora My berikut: SQL

- Aurora SQL Versi saya 2, untuk cluster My SQL 5.7 yang kompatibel.
- Aurora SQL Versi saya 3.01 dan lebih tinggi, untuk cluster My SQL 8.0 yang kompatibel. Prosedur tersimpan tidak tersedia di Aurora My SQL versi 3.

Untuk informasi tentang memberi Aurora akses ke Lambda dan menjalankan fungsi Lambda, lihat topik berikut.

Topik

- Memberi Aurora akses ke Lambda
- Memanggil fungsi Lambda dengan fungsi Aurora My native SQL
- Memanggil fungsi Lambda dengan Aurora Prosedur tersimpan SQL Saya (tidak digunakan lagi)

## Memberi Aurora akses ke Lambda

Sebelum Anda dapat menjalankan fungsi Lambda dari cluster Aurora SQL My DB, pastikan untuk terlebih dahulu memberikan izin klaster Anda untuk mengakses Lambda.

Untuk memberikan Aurora SQL akses saya ke Lambda

1. Buat sebuah AWS Identity and Access Management (IAM) kebijakan yang memberikan izin yang memungkinkan klaster Aurora SQL My DB Anda menjalankan fungsi Lambda. Untuk petunjuk, silakan lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda.
2. Buat IAM peran, dan lampirkan IAM kebijakan yang Anda buat Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda ke IAM peran baru. Untuk petunjuk, silakan lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Setel parameter cluster `aws_default_lambda_role` DB ke Amazon Resource Name (ARN) dari IAM peran baru.

   Jika klaster tersebut adalah bagian dari basis data global Aurora, terapkan pengaturan yang sama untuk setiap klaster Aurora di basis data global.

   Untuk informasi lebih lanjut tentang parameter klaster DB, lihat Parameter klaster DB dan instans DB Amazon Aurora.
4. Untuk mengizinkan pengguna database di klaster Aurora My SQL DB menjalankan fungsi Lambda, kaitkan peran yang Anda buat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan cluster DB. Untuk informasi tentang mengaitkan IAM peran dengan cluster DB, lihatMengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

   Jika klaster adalah bagian dari basis data global Aurora, kaitkan peran dengan setiap klaster Aurora di basis data global.
5. Konfigurasikan cluster Aurora My SQL DB Anda untuk memungkinkan koneksi keluar ke Lambda. Untuk petunjuk, silakan lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

Jika klaster adalah bagian dari basis data global Aurora, aktifkan koneksi keluar untuk setiap klaster Aurora di basis data global.

## Memanggil fungsi Lambda dengan fungsi Aurora My native SQL

> **Note**
>
> Anda dapat memanggil fungsi asli `lambda_sync` dan `lambda_async` ketika Anda menggunakan Aurora My SQL versi 2, atau Aurora My SQL versi 3.01 dan lebih tinggi. Untuk informasi lebih lanjut tentang Aurora SQL versi Saya, lihat. Pembaruan mesin database untuk Amazon Aurora My SQL

Anda dapat memanggil AWS Lambda fungsi dari cluster Aurora SQL My DB dengan memanggil fungsi asli `lambda_sync` dan. `lambda_async` Pendekatan ini dapat berguna ketika Anda ingin mengintegrasikan database Anda yang berjalan di Aurora My SQL dengan layanan lain AWS . Misalnya, Anda mungkin ingin mengirim pemberitahuan menggunakan Amazon Simple Notification Service (AmazonSNS) setiap kali baris dimasukkan ke dalam tabel tertentu dalam database Anda.

Daftar Isi


- Menggunakan fungsi native untuk menginvokasi fungsi Lambda
  - Memberikan peran dalam Aurora My versi 3 SQL
  - Memberikan hak istimewa di Aurora My versi 2 SQL
  - Sintaksis untuk fungsi lambda_sync
  - Parameter untuk fungsi lambda_sync
  - Contoh untuk fungsi lambda_sync
  - Sintaksis untuk fungsi lambda_async
  - Parameter untuk fungsi lambda_async
  - Contoh untuk fungsi lambda_async
  - Menginvokasi fungsi Lambda di dalam pemicu

Menggunakan fungsi native untuk menginvokasi fungsi Lambda

Fungsi `lambda_sync` dan `lambda_async` adalah fungsi native default yang menginvokasi fungsi Lambda secara sinkron atau asinkron. Ketika Anda harus mengetahui hasil dari fungsi Lambda sebelum beralih ke tindakan lain, gunakan fungsi sinkron `lambda_sync`. Ketika Anda tidak harus mengetahui hasil dari fungsi Lambda sebelum beralih ke tindakan lain, gunakan fungsi asinkron `lambda_async`.

Memberikan peran dalam Aurora My versi 3 SQL

Di Aurora My SQL versi 3, pengguna yang menjalankan fungsi asli harus diberikan peran tersebut. `AWS_LAMBDA_ACCESS` Untuk memberikan peran ini kepada pengguna, hubungkan ke instans DB sebagai pengguna administratif, dan jalankan pernyataan berikut.

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

Anda dapat mencabut peran ini dengan menjalankan pernyataan berikut.

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> Saat Anda menggunakan teknik peran di Aurora My SQL versi 3, Anda juga dapat mengaktifkan peran dengan menggunakan pernyataan `SET ROLE role_name` atau`SET ROLE ALL`. Jika Anda tidak terbiasa dengan sistem peran My SQL 8.0, Anda dapat mempelajari lebih lanjut diModel hak akses berbasis peran. Untuk detail selengkapnya, lihat Menggunakan peran di Manual SQL Referensi Saya.
>
> Hal ini hanya berlaku untuk sesi aktif saat ini. Ketika Anda terhubung kembali, Anda harus menjalankan `SET ROLE` pernyataan lagi untuk memberikan hak istimewa. Untuk informasi lebih lanjut, lihat SETROLEpernyataan di Manual SQL Referensi Saya.
>
> Anda dapat menggunakan parameter klaster DB `activate_all_roles_on_login` untuk mengaktifkan semua peran secara otomatis saat pengguna terhubung ke instans DB. Ketika parameter ini disetel, Anda biasanya tidak perlu memanggil `SET ROLE` pernyataan secara eksplisit untuk mengaktifkan peran. Untuk informasi selengkapnya, lihat activate_all_roles_on_login di Manual Referensi Saya. SQL
>
> Namun, Anda harus memanggil secara `SET ROLE ALL` eksplisit di awal prosedur tersimpan untuk mengaktifkan peran, ketika prosedur yang disimpan dipanggil oleh pengguna yang berbeda.

Jika Anda mendapatkan kesalahan seperti berikut ketika Anda mencoba menginvokasi fungsi Lambda, kemudian jalankan pernyataan `SET ROLE`.

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

Pastikan Anda memberikan peran kepada pengguna yang benar, seperti yang ditunjukkan pada entri `mysql.users` tabel. Mungkin ada beberapa pengguna dengan nama yang sama, tetapi pada host yang berbeda. Bergantung pada aplikasi atau host mana yang menjalankan `lambda_sync` fungsi, My SQL memilih pengguna dengan kecocokan terbaik sesuai dengan entri `host` kolom.

Memberikan hak istimewa di Aurora My versi 2 SQL

Di Aurora My SQL versi 2, pengguna yang menjalankan fungsi asli harus diberikan hak istimewa. `INVOKE LAMBDA` Untuk memberikan hak akses ini kepada pengguna, hubungkan ke instans DB sebagai pengguna administratif, lalu jalankan pernyataan berikut.

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

Anda dapat mencabut hak akses ini dengan menjalankan pernyataan berikut.

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

Sintaksis untuk fungsi lambda_sync

Anda menginvokasi fungsi `lambda_sync` secara sinkron dengan jenis invokasi `RequestResponse`. Fungsi mengembalikan hasil pemanggilan Lambda dalam payload. JSON Fungsi tersebut memiliki sintaksis berikut.

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

Parameter untuk fungsi lambda_sync

Fungsi `lambda_sync` memiliki parameter berikut.

lambda_fungsi_ ARN

Nama Sumber Daya Amazon (ARN) dari fungsi Lambda untuk dipanggil.

JSON_muatan

Payload untuk fungsi Lambda yang dipanggil, dalam format. JSON

> **Note**
>
> Aurora SQL Versi saya 3 mendukung fungsi JSON parsing dari My 8.0. SQL Namun, Aurora My SQL version 2 tidak menyertakan fungsi-fungsi tersebut. JSONparsing tidak diperlukan ketika fungsi Lambda mengembalikan nilai atom, seperti angka atau string.

Contoh untuk fungsi lambda_sync

Kueri berikut berdasarkan `lambda_sync` memanggil fungsi Lambda secara sinkron `BasicTestLambda` menggunakan fungsi tersebut. ARN Payload untuk fungsi tersebut adalah `{"operation": "ping"}`.

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

Sintaksis untuk fungsi lambda_async

Anda menginvokasi fungsi `lambda_async` secara asinkron dengan jenis invokasi `Event`. Fungsi mengembalikan hasil pemanggilan Lambda dalam payload. JSON Fungsi tersebut memiliki sintaksis berikut.

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

Parameter untuk fungsi lambda_async

Fungsi `lambda_async` memiliki parameter berikut.

lambda_fungsi_ ARN

Nama Sumber Daya Amazon (ARN) dari fungsi Lambda untuk dipanggil.

JSON_muatan

Payload untuk fungsi Lambda yang dipanggil, dalam format. JSON

> **Note**
>
> Aurora SQL Versi saya 3 mendukung fungsi JSON parsing dari My 8.0. SQL Namun, Aurora My SQL version 2 tidak menyertakan fungsi-fungsi tersebut. JSONparsing tidak diperlukan ketika fungsi Lambda mengembalikan nilai atom, seperti angka atau string.

Contoh untuk fungsi lambda_async

Kueri berikut berdasarkan `lambda_async` memanggil fungsi Lambda secara asinkron `BasicTestLambda` menggunakan fungsi tersebut. ARN Payload untuk fungsi tersebut adalah `{"operation": "ping"}`.

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

Menginvokasi fungsi Lambda di dalam pemicu

Anda dapat menggunakan pemicu untuk memanggil Lambda pada pernyataan pengubahan data. Contoh berikut menggunakan fungsi native `lambda_async` dan menyimpan hasilnya dalam variabel.

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
mysql>DELIMITER ;
```

> **Note**
>
> Pemicu tidak dijalankan sekali per SQL pernyataan, tetapi sekali per baris dimodifikasi, satu baris pada satu waktu. Saat pemicu berjalan, prosesnya sinkron. Pernyataan modifikasi data hanya menampilkan hasil saat pemicu selesai.
> Hati-hati saat menjalankan AWS Lambda fungsi dari pemicu pada tabel yang mengalami lalu lintas tulis tinggi. `INSERT`,`UPDATE`, dan `DELETE` pemicu diaktifkan per baris. Beban kerja berat tulis di atas meja dengan`INSERT`,`UPDATE`, atau `DELETE` pemicu menghasilkan sejumlah besar panggilan ke fungsi Anda. AWS Lambda

## Memanggil fungsi Lambda dengan Aurora Prosedur tersimpan SQL Saya (tidak digunakan lagi)

Anda dapat memanggil AWS Lambda fungsi dari cluster Aurora My SQL DB dengan memanggil prosedur. `mysql.lambda_async` Pendekatan ini dapat berguna ketika Anda ingin mengintegrasikan database Anda yang berjalan di Aurora My SQL dengan yang lain AWS layanan. Misalnya, Anda mungkin ingin mengirim pemberitahuan menggunakan Amazon Simple Notification Service (AmazonSNS) setiap kali baris dimasukkan ke dalam tabel tertentu dalam database Anda.

Daftar Isi


- Aurora Pertimbangan versi saya SQL
- Menggunakan prosedur mysql.lambda_async untuk menginvokasi fungsi Lambda (sudah dihentikan)
  - Sintaksis
  - Parameter
  - Contoh


### Aurora Pertimbangan versi saya SQL

Mulai di Aurora My SQL versi 2, Anda dapat menggunakan metode fungsi asli alih-alih prosedur tersimpan ini untuk memanggil fungsi Lambda. Untuk informasi selengkapnya tentang fungsi native, lihat Menggunakan fungsi native untuk menginvokasi fungsi Lambda.

Di Aurora My SQL versi 2, prosedur yang disimpan tidak lagi `mysql.lambda_async` didukung. Sebaiknya gunakan fungsi Lambda native sebagai gantinya.

Di Aurora My SQL versi 3, prosedur yang disimpan tidak tersedia.

Menggunakan prosedur mysql.lambda_async untuk menginvokasi fungsi Lambda (sudah dihentikan)

Prosedur `mysql.lambda_async` adalah prosedur tersimpan default yang menginvokasi fungsi Lambda secara asinkron. Untuk menggunakan prosedur ini, pengguna basis data Anda harus memiliki hak akses `EXECUTE` pada prosedur tersimpan `mysql.lambda_async`.

Sintaksis

Prosedur `mysql.lambda_async` memiliki sintaksis berikut.

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
)
```

Parameter

Prosedur `mysql.lambda_async` memiliki parameter berikut.

lambda_fungsi_ ARN

Nama Sumber Daya Amazon (ARN) dari fungsi Lambda untuk dipanggil.

lambda_function_input

String input, dalam JSON format, untuk fungsi Lambda yang dipanggil.

Contoh

Sebagai praktik terbaik, sebaiknya selesaikan panggilan ke prosedur `mysql.lambda_async` dalam prosedur tersimpan yang dapat dipanggil dari sumber berbeda seperti pemicu atau kode klien. Pendekatan ini dapat membantu menghindari masalah ketidakcocokan impedansi dan memudahkan Anda untuk menginvokasi fungsi Lambda.

> **Note**
>
> Hati-hati saat memanggil AWS Lambda berfungsi dari pemicu pada tabel yang mengalami lalu lintas tulis tinggi. `INSERT`,`UPDATE`, dan `DELETE` pemicu diaktifkan per baris. Beban kerja yang berat di atas meja dengan`INSERT`,`UPDATE`, atau `DELETE` pemicu menghasilkan sejumlah besar panggilan ke Anda AWS Lambda fungsi.

Meskipun panggilan ke prosedur `mysql.lambda_async` bersifat asinkron, pemicunya bersifat sinkron. Pernyataan yang menghasilkan sejumlah besar aktivasi pemicu tidak menunggu panggilan ke AWS Lambda berfungsi untuk menyelesaikan, tetapi menunggu pemicu selesai sebelum mengembalikan kontrol ke klien.

Example Contoh: Memanggil AWS Lambda fungsi untuk mengirim email

Contoh berikut membuat prosedur tersimpan yang dapat Anda panggil dalam kode basis data Anda untuk mengirim email menggunakan fungsi Lambda.

AWS Lambda Fungsi

```
import boto3

ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

Prosedur Tersimpan

```
DROP PROCEDURE IF EXISTS SES_send_email;
```

```
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                  IN email_to VARCHAR(255),
                                  IN subject VARCHAR(255),
                                  IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

Panggil Prosedur Tersimpan untuk Memanggil AWS Lambda Fungsi

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example Contoh: Memanggil AWS Lambda fungsi untuk mempublikasikan peristiwa dari pemicu

Contoh berikut membuat prosedur tersimpan yang menerbitkan acara dengan menggunakan AmazonSNS. Kode tersebut memanggil prosedur dari pemicu ketika baris ditambahkan ke tabel.

AWS Lambda Fungsi

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

Prosedur Tersimpan

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

### Tabel

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

### Pemicu

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

### Sisipkan Baris ke dalam Tabel untuk Memicu Notifikasi

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

# Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch

Anda dapat mengonfigurasi cluster DB MySQL Aurora Anda untuk mempublikasikan data log umum, lambat, audit, dan kesalahan ke grup log di Amazon Logs. CloudWatch Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Anda dapat menggunakan CloudWatch Log untuk menyimpan catatan log Anda dalam penyimpanan yang sangat tahan lama.

Untuk mempublikasikan CloudWatch log ke Log, masing-masing log harus diaktifkan. Log kesalahan diaktifkan secara default, tetapi Anda harus mengaktifkan jenis log lain secara eksplisit. Untuk informasi tentang cara mengaktifkan log dalam MySQL, lihat Selecting general query and slow query log output destinations dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang mengaktifkan log audit Aurora MySQL, lihat Mengaktifkan Audit Lanjutan.

> **Note**
>
> - Jika pengeksporan data log dinonaktifkan, Aurora tidak menghapus grup log atau log stream yang ada. Jika mengekspor data log dinonaktifkan, data log yang ada tetap tersedia di CloudWatch Log, tergantung pada penyimpanan log, dan Anda masih dikenakan biaya untuk data log audit yang disimpan. Anda dapat menghapus aliran log dan grup log menggunakan konsol CloudWatch Log, API AWS CLI, atau CloudWatch Logs.
> - Cara alternatif untuk mempublikasikan log audit ke CloudWatch Log adalah dengan mengaktifkan Audit Lanjutan, lalu membuat grup parameter cluster DB kustom dan menyetel `server_audit_logs_upload` parameternya ke. `1` Default untuk parameter klaster DB `server_audit_logs_upload` adalah `0`. Untuk informasi tentang mengaktifkan Audit Lanjutan, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.
>
>   Jika Anda menggunakan metode alternatif ini, Anda harus memiliki peran IAM untuk mengakses CloudWatch Logs dan mengatur parameter tingkat cluster `aws_default_logs_role` ke ARN untuk peran ini. Untuk informasi tentang membuat peran, lihat Menyiapkan IAM peran untuk mengakses AWS layanan. Namun, jika Anda memiliki peran tertaut-layanan `AWSServiceRoleForRDS`, ini memberikan akses ke CloudWatch Logs dan menimpa peran yang ditentukan khusus. Untuk informasi tentang peran terkait layanan untuk Amazon RDS, lihat Menggunakan peran terkait layanan untuk Amazon Aurora.

- Jika Anda tidak ingin mengekspor log audit ke CloudWatch Log, pastikan semua metode mengekspor log audit dinonaktifkan. Metode ini adalah AWS Management Console, AWS CLI, API RDS, dan parameter `server_audit_logs_upload`.
- Prosedurnya sedikit berbeda untuk Aurora Serverless v1 Cluster DB daripada untuk cluster DB dengan disediakan atau Aurora Serverless v2 Instans DB. Aurora Serverless v1 cluster secara otomatis mengunggah semua log yang Anda aktifkan melalui parameter konfigurasi.

  Oleh karena itu, Anda mengaktifkan atau menonaktifkan unggahan log untuk Aurora Serverless v1 Cluster DB dengan mengaktifkan dan menonaktifkan jenis log yang berbeda di grup parameter cluster DB. Anda tidak mengubah pengaturan cluster itu sendiri melalui AWS Management Console, AWS CLI, atau RDS API. Untuk informasi tentang mengaktifkan dan menonaktifkan log MySQL Aurora Serverless v1 kluster, lihat Kelompok parameter untuk Aurora Serverless v1.

Konsol

Anda dapat menerbitkan log Aurora MySQL untuk klaster yang disediakan ke Log dengan konsol. CloudWatch

Untuk menerbitkan log Aurora MySQL dari konsol

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB Aurora MySQL yang data log-nya ingin Anda terbitkan.
4. Pilih Ubah.
5. Di bagian Log ekspor, pilih log yang ingin Anda mulai terbitkan ke CloudWatch Log.
6. Pilih Lanjutkan, lalu pilih Modifikasi Klaster DB di halaman ringkasan.

AWS CLI

Anda dapat mempublikasikan log Aurora MySQL untuk cluster yang disediakan dengan CLI. AWS Untuk melakukannya, Anda menjalankan modify-db-cluster AWS CLI perintah dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--cloudwatch-logs-export-configuration`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora MySQL dengan menjalankan salah satu dari perintah AWS CLI berikut:

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Jalankan salah satu AWS CLI perintah ini dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--engine` – Mesin basis data.
- `--enable-cloudwatch-logs-exports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Opsi lain mungkin diperlukan tergantung pada AWS CLI perintah yang Anda jalankan.

Example

Perintah berikut memodifikasi cluster Aurora MySQL DB yang ada untuk mempublikasikan file log ke Log. CloudWatch

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Example

Perintah berikut membuat cluster Aurora MySQL DB untuk mempublikasikan file log ke Log. CloudWatch

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

API RDS

Anda dapat menerbitkan log Aurora MySQL untuk klaster terprovisi dengan API RDS. Untuk melakukannya, Anda menjalankan DBCluster operasi Modify dengan opsi berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `CloudwatchLogsExportConfiguration`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora MySQL menggunakan API RDS dengan menjalankan salah satu dari operasi API RDS berikut:

- Buat DBCluster
- Kembalikan DBCluster dari3
- MemulihkanDBClusterFromSnapshot
- MemulihkanDBClusterToPointInTime

Jalankan operasi API RDS dengan parameter berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `Engine` – Mesin basis data.
- `EnableCloudwatchLogsExports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Parameter lain mungkin diperlukan tergantung pada AWS CLI perintah yang Anda jalankan.

## Memantau peristiwa log di Amazon CloudWatch

Setelah mengaktifkan peristiwa log Aurora MySQL, Anda dapat memantau peristiwa di Amazon Logs. CloudWatch Grup log baru secara otomatis dibuat untuk klaster DB Aurora menggunakan awalan berikut, dengan *`cluster-name`* merepresentasikan nama klaster DB, dan *`log_type`* merepresentasikan jenis log.

```
/aws/rds/cluster/cluster-name/log_type
```

Misalnya, jika Anda mengonfigurasi fungsi ekspor untuk menyertakan log kueri yang lambat untuk klaster DB bernama `mydbcluster`, data kueri yang lambat disimpan di grup log `/aws/rds/cluster/mydbcluster/slowquery`.

Peristiwa dari semua instans di klaster Anda didorong ke grup log menggunakan log stream yang berbeda-beda. Perilakunya bergantung pada mana yang berlaku dari kondisi berikut ini:

- Grup log dengan nama yang ditentukan tersedia.

  Aurora menggunakan grup log yang ada untuk mengekspor data log untuk klaster. Untuk membuat grup log dengan periode retensi log, filter metrik, dan akses pelanggan standar, Anda dapat menggunakan konfigurasi otomatis, seperti AWS CloudFormation.
- Grup log dengan nama yang ditentukan tidak ada.

  Ketika entri log yang cocok terdeteksi dalam file log untuk contoh, Aurora MySQL membuat grup log baru di Log secara otomatis. CloudWatch Grup log menggunakan periode retensi log default Tidak Pernah Kedaluwarsa.

  Untuk mengubah periode penyimpanan log, gunakan konsol CloudWatch Log, API AWS CLI, atau CloudWatch Logs. Untuk informasi selengkapnya tentang mengubah periode penyimpanan CloudWatch log di Log, lihat Mengubah penyimpanan data CloudWatch log di Log.

Untuk mencari informasi dalam peristiwa log untuk klaster DB, gunakan konsol CloudWatch Log, API AWS CLI, atau CloudWatch Logs. Untuk informasi selengkapnya tentang mencari dan memfilter data log, lihat Menelusuri dan memfilter data log.

# Mode lab Amazon Aurora MySQL

> **Important**
>
> Mode lab diperkenalkan di Aurora MySQL versi 2 untuk mengaktifkan optimasi Fast DDL, yang meningkatkan efisiensi operasi DDL tertentu. Di Aurora MySQL versi 3, mode lab telah dihapus, dan Fast DDL telah digantikan oleh fitur MySQL 8.0 yang disebut Instant DDL.

Mode lab Aurora digunakan untuk mengaktifkan fitur Aurora yang tersedia dalam versi basis data Aurora saat ini, tetapi tidak diaktifkan secara default. Meskipun fitur mode lab Aurora tidak disarankan untuk digunakan dalam klaster DB produksi, Anda dapat menggunakan mode lab Aurora untuk mengaktifkan fitur ini untuk klaster DB di lingkungan pengembangan dan pengujian Anda. Untuk informasi selengkapnya tentang fitur Aurora yang tersedia saat mode lab Aurora diaktifkan, lihat Fitur mode lab Aurora.

Parameter `aurora_lab_mode` adalah parameter tingkat instans yang berada dalam grup parameter default. Parameter ini diatur ke `0` (dinonaktifkan) dalam grup parameter default. Untuk mengaktifkan mode lab Aurora, buat grup parameter kustom, atur parameter `aurora_lab_mode` ke `1` (diaktifkan) dalam grup parameter kustom tersebut, dan ubah satu atau beberapa instans DB dalam klaster Aurora Anda agar menggunakan grup parameter kustom tersebut. Kemudian, hubungkan ke titik akhir instans yang sesuai untuk mencoba fitur mode lab. Untuk informasi tentang mengubah grup parameter DB, lihat Memodifikasi parameter dalam grup parameter DB di . Untuk informasi tentang grup parameter dan Amazon Aurora, lihat Parameter konfigurasi Aurora MySQL.

## Fitur mode lab Aurora

Fitur Aurora berikut saat ini tersedia saat mode lab Aurora diaktifkan. Anda harus mengaktifkan mode lab Aurora sebelum salah satu fitur ini dapat digunakan.

DDL Cepat

Fitur ini memungkinkan Anda menjalankan operasi `ALTER TABLE` *`tbl_name`* `ADD COLUMN` *`col_name column_definition`* hampir seketika. Operasi diselesaikan tanpa mengharuskan tabel untuk disalin dan tanpa secara material memengaruhi pernyataan DML lainnya. Karena tidak menggunakan penyimpanan sementara untuk salinan tabel, itu membuat pernyataan DDL praktis bahkan untuk tabel besar pada kelas instance kecil.

DDL Cepat saat ini hanya mendukung penambahan kolom yang dapat dibuat null, tanpa nilai default, di akhir tabel. Untuk informasi selengkapnya tentang penggunaan fitur ini, lihat Mengubah tabel di Amazon Aurora menggunakan DDL Cepat.

# Praktik terbaik dengan Amazon Aurora My SQL

Topik ini mencakup informasi tentang praktik terbaik dan opsi untuk menggunakan atau memigrasi data ke klaster Amazon Aurora SQL My DB. Informasi dalam topik ini merangkum dan mengulangi beberapa pedoman dan prosedur yang dapat Anda temukan di Mengelola klaster DB Amazon Aurora.

Daftar Isi


- Menentukan ke instans DB mana Anda terhubung
- Praktik terbaik untuk performa dan penskalaan Aurora MySQL
  - Menggunakan kelas instans T untuk pengembangan dan pengujian
  - Mengoptimalkan kueri join yang diindeks MySQL Aurora dengan prefetch kunci asinkron
    - Mengaktifkan prefetch kunci asinkron
    - Mengoptimalkan kueri untuk prefetch kunci asinkron
  - Mengoptimalkan kueri join MySQL Aurora besar dengan hash join
    - Mengaktifkan hash join
    - Mengoptimalkan kueri untuk hash join
  - Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda
  - Mengoptimalkan operasi stempel waktu
  - Kesalahan luapan ID indeks virtual
- Praktik terbaik untuk Aurora Ketersediaan tinggi saya SQL
  - Menggunakan Amazon Aurora untuk Pemulihan Bencana dengan database Saya SQL
  - Bermigrasi dari My SQL ke Amazon Aurora SQL My dengan waktu henti yang dikurangi
  - Menghindari kinerja lambat, restart otomatis, dan failover untuk instans Aurora My DB SQL
- Rekomendasi untuk SQL fitur Saya di Aurora My SQL
  - Menggunakan replikasi multithreaded di Aurora My SQL
  - Memohon AWS Lambda fungsi menggunakan SQL fungsi saya asli
  - Menghindari transaksi XA dengan Amazon Aurora My SQL
  - Menyimpan kunci asing dihidupkan selama DML pernyataan
  - Mengonfigurasi seberapa sering buffer log di-flush
  - Meminimalkan dan memecahkan masalah Aurora Kebuntuan saya SQL
    - Meminimalkan deadlock InnoDB

    - Memantau deadlock InnoDB
- Mengevaluasi penggunaan instans DB untuk Aurora My SQL dengan metrik Amazon CloudWatch

## Menentukan ke instans DB mana Anda terhubung

Untuk menentukan instans DB mana dalam cluster Aurora My SQL DB yang terhubung dengan koneksi, periksa variabel `innodb_read_only` global, seperti yang ditunjukkan pada contoh berikut.

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

Variabel `innodb_read_only` diatur ke `0N` jika Anda terhubung ke instans DB pembaca. Pengaturan ini berstatus `0FF` jika Anda terhubung ke instans DB penulis, seperti instans primer dalam klaster terprovisi.

Pendekatan ini dapat membantu jika Anda ingin menambahkan logika ke kode aplikasi Anda untuk menyeimbangkan beban kerja atau untuk memastikan bahwa operasi tulis menggunakan koneksi yang tepat.

## Praktik terbaik untuk performa dan penskalaan Aurora MySQL

Anda dapat menerapkan praktik terbaik berikut untuk meningkatkan performa dan skalabilitas klaster Aurora MySQL Anda.

Topik

- Menggunakan kelas instans T untuk pengembangan dan pengujian
- Mengoptimalkan kueri join yang diindeks MySQL Aurora dengan prefetch kunci asinkron
- Mengoptimalkan kueri join MySQL Aurora besar dengan hash join
- Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda
- Mengoptimalkan operasi stempel waktu
- Kesalahan luapan ID indeks virtual

### Menggunakan kelas instans T untuk pengembangan dan pengujian

Instans Amazon Aurora MySQL yang menggunakan kelas instans DB `db.t2`, `db.t3` atau `db.t4g` paling cocok untuk aplikasi yang tidak mendukung beban kerja yang tinggi untuk waktu yang lama. Instans T dirancang untuk memberikan performa dasar sedang dan kemampuan untuk melakukan

burst performa yang secara signifikan lebih tinggi sesuai dengan yang dibutuhkan beban kerja Anda. Instans ini ditujukan untuk beban kerja yang tidak menggunakan CPU penuh secara sering atau konsisten, tetapi kadang-kadang memerlukan burst. Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Instans performa yang dapat melonjak.

Jika klaster Aurora Anda lebih besar dari 40 TB, jangan gunakan kelas instans T. Ketika basis data Anda memiliki volume data yang besar, overhead memori untuk mengelola objek skema dapat melebihi kapasitas instans T.

Jangan aktifkan Skema Performa MySQL pada instans T Amazon Aurora MySQL. Jika Skema Performa diaktifkan, instans ini dapat kehabisan memori.

> **Tip**
>
> Jika database Anda terkadang menganggur tetapi di lain waktu memiliki beban kerja yang besar, Anda dapat menggunakannya Aurora Serverless v2 sebagai alternatif untuk contoh T. Dengan Aurora Serverless v2, Anda menentukan rentang kapasitas dan Aurora secara otomatis menskalakan database Anda ke atas atau ke bawah tergantung pada beban kerja saat ini. Untuk detail penggunaan, lihat Menggunakan Aurora Serverless v2. Untuk versi mesin database yang dapat Anda gunakan dengan Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2.

Saat Anda menggunakan instans T sebagai instans DB di klaster DB Aurora MySQL kami menyarankan hal berikut:

- Gunakan kelas instans DB yang sama untuk semua instans di klaster DB Anda. Misalnya, jika Anda menggunakan `db.t2.medium` untuk instans penulis Anda, maka kami sarankan Anda menggunakan `db.t2.medium` untuk instans pembaca Anda juga.
- Jangan sesuaikan pengaturan konfigurasi terkait memori apa pun, seperti `innodb_buffer_pool_size`. Aurora menggunakan serangkaian nilai default yang sudah sangat disesuaikan untuk buffer memori pada instans T. Default khusus ini diperlukan agar Aurora dapat berjalan pada instans yang dibatasi memori. Jika Anda mengubah pengaturan terkait memori pada instance T, Anda jauh lebih mungkin menghadapi out-of-memory kondisi, bahkan jika perubahan Anda dimaksudkan untuk meningkatkan ukuran buffer.
- Pantau Saldo Kredit CPU Anda (`CPUCreditBalance`) untuk memastikannya berada pada tingkat yang layak. Artinya, kredit CPU diakumulasi pada kecepatan yang sama dengan yang digunakan.

Ketika Anda telah menghabiskan kredit CPU untuk instans, Anda akan mengalami penurunan yang cepat pada CPU yang tersedia serta peningkatan latensi baca dan tulis untuk instans. Situasi ini menyebabkan penurunan yang parah dalam performa keseluruhan instans.

Jika saldo kredit CPU Anda tidak berada pada tingkat yang layak, maka kami menyarankan untuk memodifikasi instans DB Anda agar menggunakan salah satu kelas instans DB R yang didukung (penskalaan komputasi).

Untuk informasi selengkapnya tentang pemantauan metrik, lihat Melihat metrik di konsol Amazon RDS.

- Pantau lag replika (`AuroraReplicaLag`) antara instans penulis dan instans pembaca.

  Jika instans pembaca kehabisan kredit CPU sebelum instans penulis, lag yang dihasilkan dapat menyebabkan instans pembaca sering diaktifkan ulang. Hasil ini biasa terjadi jika aplikasi memiliki beban operasi baca berat yang didistribusikan di antara instans pembaca, pada waktu yang sama saat instans penulis memiliki beban operasi tulis minimal.

  Jika Anda melihat peningkatan berkelanjutan dalam lag replika, pastikan saldo kredit CPU Anda untuk instans pembaca di klaster DB Anda tidak habis.

  Jika saldo kredit CPU Anda tidak berada pada tingkat yang layak, maka kami menyarankan untuk memodifikasi instans DB Anda agar menggunakan salah satu kelas instans DB R yang didukung (penskalaan komputasi).
- Pertahankan jumlah penyisipan per transaksi di bawah 1 juta untuk klaster DB yang memiliki pencatatan log biner.

  Jika grup parameter cluster DB untuk cluster DB Anda memiliki `binlog_format` parameter yang disetel ke nilai selain`OFF`, maka cluster DB Anda mungkin mengalami out-of-memory kondisi jika cluster DB menerima transaksi yang berisi lebih dari 1 juta baris untuk disisipkan. Anda dapat memantau metrik memori yang dapat dibebaskan (`FreeableMemory`) untuk menentukan apakah klaster DB Anda kehabisan memori. Kemudian, Anda memeriksa metrik operasi tulis (`VolumeWriteIOPS`) untuk melihat apakah instans penulis menerima beban berat dari operasi tulis. Jika demikian, kami menyarankan agar Anda memperbarui aplikasi Anda untuk membatasi jumlah penyisipan dalam transaksi hingga kurang dari 1 juta. Alternatifnya, Anda dapat memodifikasi instans Anda untuk menggunakan salah satu dari kelas instans DB R yang didukung (penskalaan komputasi).

## Mengoptimalkan kueri join yang diindeks MySQL Aurora dengan prefetch kunci asinkron

Aurora MySQL dapat menggunakan fitur prefetch kunci asinkron (AKP) untuk meningkatkan performa kueri yang menggabungkan tabel di seluruh indeks. Fitur ini meningkatkan performa dengan mengantisipasi baris yang diperlukan untuk menjalankan kueri yang mengharuskan kueri JOIN menggunakan fitur optimisasi Batched Key Access (BKA) dan Multi-Range Read (MRR). Untuk informasi selengkapnya tentang BKA dan MRR, lihat Block nested-loop and batched key access joins dan Multi-range read optimization dalam dokumentasi MySQL.

Untuk memanfaatkan fitur AKP, kueri harus menggunakan BKA dan MRR. Biasanya, kueri tersebut terjadi saat klausa JOIN dari kueri menggunakan indeks sekunder, tetapi juga memerlukan beberapa kolom dari indeks primer. Misalnya, Anda dapat menggunakan AKP ketika klausa JOIN merepresentasikan equijoin pada nilai indeks antara tabel luar kecil dan tabel dalam besar, dan indeksnya sangat selektif pada tabel yang lebih besar. AKP beroperasi bersama dengan BKA dan MRR untuk melakukan pencarian indeks sekunder hingga primer selama evaluasi klausa JOIN. AKP mengidentifikasi baris yang diperlukan untuk menjalankan kueri selama evaluasi klausa JOIN. Kemudian, fitur ini menggunakan thread latar belakang untuk secara asinkron memuat halaman yang berisi baris tersebut ke dalam memori sebelum menjalankan kueri.

AKP tersedia untuk Aurora MySQL versi 2.10 dan lebih tinggi, serta versi 3. Untuk informasi selengkapnya tentang versi Aurora MySQL, lihat Pembaruan mesin database untuk Amazon Aurora My SQL.

### Mengaktifkan prefetch kunci asinkron

Anda dapat mengaktifkan fitur AKP dengan mengatur `aurora_use_key_prefetch`, sebuah variabel server MySQL, ke `on`. Secara default, nilai ini diatur ke `on`. Namun, AKP tidak dapat diaktifkan sampai Anda juga mengaktifkan algoritma BKA Join dan menonaktifkan fungsionalitas MRR berbasis biaya. Untuk melakukannya, Anda harus menetapkan nilai berikut untuk `optimizer_switch`, sebuah variabel server MySQL:

- Atur `batched_key_access` ke `on`. Nilai ini mengontrol penggunaan algoritma BKA Join. Secara default, nilai ini diatur ke `off`.
- Atur `mrr_cost_based` ke `off`. Nilai ini mengontrol penggunaan fungsionalitas MRR berbasis biaya. Secara default, nilai ini diatur ke `on`.

Saat ini, Anda dapat mengatur nilai ini hanya di tingkat sesi. Contoh berikut menggambarkan cara menetapkan nilai-nilai ini untuk mengaktifkan AKP untuk sesi saat ini dengan mengeksekusi pernyataan SET.

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

Demikian pula, Anda dapat menggunakan pernyataan SET untuk menonaktifkan AKP dan algoritma BKA Join serta mengaktifkan ulang fungsi MRR berbasis biaya untuk sesi saat ini, seperti yang ditunjukkan dalam contoh berikut.

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

Untuk informasi selengkapnya tentang switch pengoptimisasi batched_key_access dan mrr_cost_based lihat Switchable optimizations dalam dokumentasi MySQL.

Mengoptimalkan kueri untuk prefetch kunci asinkron

Anda dapat mengonfirmasi apakah kueri dapat memanfaatkan fitur AKP. Untuk melakukannya, gunakan pernyataan `EXPLAIN` untuk membuat profil kueri sebelum menjalankannya. Pernyataan `EXPLAIN` memberikan informasi tentang rencana eksekusi yang akan digunakan untuk kueri tertentu.

Pada output pernyataan `EXPLAIN`, kolom `Extra` menjelaskan informasi tambahan, termasuk rencana eksekusi. Jika fitur AKP berlaku pada tabel yang digunakan dalam kueri, kolom ini mencakup salah satu nilai berikut:

- `Using Key Prefetching`
- `Using join buffer (Batched Key Access with Key Prefetching)`

Contoh berikut menunjukkan penggunaan `EXPLAIN` untuk melihat rencana eksekusi untuk kueri yang dapat memanfaatkan AKP.

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
```

```
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
    ->     and s_nationkey = n_nationkey
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
| id | select_type | table    | type | possible_keys         | key           | key_len
 | ref                               | rows | filtered | Extra
                              |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
|  1 | PRIMARY     | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where; Using temporary;
 Using filesort                |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                              |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where
                              |
```

```
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                              |
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

Untuk informasi selengkapnya tentang format output `EXPLAIN`, lihat Extended EXPLAIN output format dalam dokumentasi MySQL.

## Mengoptimalkan kueri join MySQL Aurora besar dengan hash join

Saat Anda harus menggabungkan sejumlah besar data dengan menggunakan equijoin, hash join dapat meningkatkan performa kueri. Anda dapat mengaktifkan hash join untuk Aurora MySQL.

Kolom hash join dapat berupa ekspresi rumit apa pun. Dalam kolom hash join, Anda dapat membandingkan jenis data dengan cara berikut:

- Anda dapat membandingkan apa pun dalam kategori jenis data numerik yang tepat, seperti `int`, `bigint`, `numeric`, dan `bit`.
- Anda dapat membandingkan apa pun dalam kategori perkiraan jenis data numerik, seperti `float` dan `double`.
- Anda dapat membandingkan item di seluruh jenis string jika jenis string memiliki kumpulan karakter dan kolasi yang sama.
- Anda dapat membandingkan item dengan jenis data tanggal dan stempel waktu jika jenisnya sama.

> **Note**
>
> Anda tidak dapat membandingkan jenis data dalam kategori yang berbeda.

Batasan berikut berlaku untuk hash join untuk Aurora MySQL:

- Outer join kiri-kanan tidak didukung untuk Aurora MySQL versi 2, tetapi didukung untuk versi 3.

- Semijoin seperti subkueri tidak didukung, kecuali jika subkueri dimaterialkan terlebih dahulu.
- Pembaruan atau penghapusan multi-tabel tidak didukung.

> **Note**
>
> Pembaruan atau penghapusan satu tabel didukung.

- Kolom jenis data spasial dan BLOB tidak dapat menjadi kolom join dalam hash join.

Mengaktifkan hash join

Untuk mengaktifkan hash join:

- Aurora MySQL versi 2 - Atur parameter DB atau parameter klaster DB `aurora_disable_hash_join` ke `0`. Menonaktifkan `aurora_disable_hash_join` akan menetapkan nilai `optimizer_switch` ke `hash_join=on`.
- Aurora MySQL versi 3 - Atur parameter server MySQL `optimizer_switch` ke `block_nested_loop=on`.

Hash join diaktifkan secara default di Aurora MySQL versi 3 dan dinonaktifkan secara default di Aurora MySQL versi 2. Contoh berikut mengilustrasikan cara mengaktifkan hash join untuk Aurora MySQL versi 3. Anda dapat mengeluarkan pernyataan `select @@optimizer_switch` terlebih dahulu untuk melihat apa saja pengaturan lain yang ada dalam string parameter `SET`. Memperbarui satu pengaturan dalam parameter `optimizer_switch` tidak akan menghapus atau memodifikasi pengaturan lainnya.

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> Untuk Aurora MySQL versi 3, dukungan hash join tersedia di semua versi minor dan diaktifkan secara default.
> Untuk Aurora MySQL versi 2, dukungan hash join tersedia di semua versi minor. Di Aurora MySQL versi 2, fitur hash join selalu dikendalikan oleh nilai `aurora_disable_hash_join`.

Dengan pengaturan ini, pengoptimisasi memilih untuk menggunakan hash join berdasarkan biaya, karakteristik kueri, dan ketersediaan sumber daya. Jika estimasi biaya salah, Anda dapat memaksa pengoptimisasi untuk memilih hash join. Caranya adalah dengan mengatur `hash_join_cost_based`, sebuah variabel server MySQL, ke `off`. Contoh berikut mengilustrasikan cara memaksa pengoptimisasi untuk memilih hash join.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> Pengaturan ini mengganti keputusan pengoptimisasi berbasis biaya. Meskipun pengaturan ini dapat berguna untuk pengujian dan pengembangan, kami menyarankan Anda untuk tidak menggunakannya dalam produksi.

### Mengoptimalkan kueri untuk hash join

Untuk mengetahui apakah kueri dapat memanfaatkan hash join, gunakan pernyataan `EXPLAIN` untuk membuat profil kueri terlebih dahulu. Pernyataan `EXPLAIN` memberikan informasi tentang rencana eksekusi yang akan digunakan untuk kueri tertentu.

Pada output pernyataan `EXPLAIN`, kolom `Extra` menjelaskan informasi tambahan, termasuk rencana eksekusi. Jika hash join berlaku pada tabel yang digunakan dalam kueri, kolom ini akan mencakup nilai seperti yang berikut ini:

- `Using where; Using join buffer (Hash Join Outer table` *`table1_name`*`)`
- `Using where; Using join buffer (Hash Join Inner table` *`table2_name`*`)`

Contoh berikut menunjukkan penggunaan EXPLAIN untuk melihat rencana eksekusi untuk kueri hash join.

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                          |
```

```
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                    |
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)  |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2) |
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
3 rows in set (0.04 sec)
```

Dalam output ini, `Hash Join Inner table` adalah tabel yang digunakan untuk membuat tabel hash, dan `Hash Join Outer table` adalah tabel yang digunakan untuk menyelidiki tabel hash.

Untuk informasi selengkapnya tentang format output `EXPLAIN` yang diperluas, lihat Extended EXPLAIN Output Format dalam dokumentasi produk MySQL.

Di Aurora MySQL 2.08 dan yang lebih tinggi, Anda dapat menggunakan petunjuk SQL untuk memengaruhi apakah kueri menggunakan hash join atau tidak, dan tabel mana yang digunakan untuk sisi build dan probe dari join. Untuk detailnya, lihat Aurora Petunjuk saya SQL.

## Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda

Anda dapat menggunakan Amazon Aurora dengan instans DB MySQL Anda untuk memanfaatkan kemampuan penskalaan baca Amazon Aurora dan memperluas beban kerja baca untuk instans DB MySQL Anda. Untuk menggunakan Aurora untuk menskalakan baca instans DB MySQL Anda, buat klaster DB Aurora MySQL dan jadikan sebagai replika baca dari instans DB MySQL Anda. Kemudian, hubungkan ke klaster Aurora MySQL untuk memproses kueri baca. Basis data sumber dapat berupa instans DB RDS for MySQL, atau sebuah basis data MySQL yang dijalankan secara eksternal di luar Amazon RDS. Untuk informasi selengkapnya, lihat Penskalaan bacaan untuk SQL database Saya dengan Amazon Aurora.

## Mengoptimalkan operasi stempel waktu

Ketika nilai variabel sistem `time_zone` diatur ke `SYSTEM`, setiap panggilan fungsi MySQL yang memerlukan perhitungan zona waktu akan membuat panggilan pustaka sistem. Saat Anda menjalankan pernyataan SQL yang menampilkan atau mengubah nilai `TIMESTAMP` tersebut pada konkurensi tinggi, Anda mungkin akan mengalami latensi, pertentangan kunci, dan penggunaan CPU yang meningkat. Untuk informasi selengkapnya, lihat time_zone dalam dokumentasi MySQL.

Untuk menghindari perilaku ini, sebaiknya Anda mengubah nilai parameter klaster DB `time_zone` menjadi `UTC`. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.

Meskipun parameter `time_zone` bersifat dinamis (tidak memerlukan pengaktifan ulang server basis data), nilai baru hanya digunakan untuk koneksi baru. Untuk memastikan bahwa semua koneksi diperbarui untuk menggunakan nilai `time_zone` baru, kami sarankan Anda mendaur ulang koneksi aplikasi Anda setelah memperbarui parameter klaster DB.

### Kesalahan luapan ID indeks virtual

Aurora MySQL membatasi nilai untuk indeks virtual IDs hingga 8 bit mencegah masalah yang disebabkan oleh format undo di MySQL. Jika indeks melebihi batas ID indeks virtual, klaster Anda mungkin tidak tersedia. Ketika indeks mendekati batas ID indeks virtual atau ketika Anda mencoba membuat indeks di atas batas ID indeks virtual, RDS mungkin membuang kode kesalahan `63955` atau kode `63955` peringatan. Untuk mengatasi kesalahan batas ID indeks virtual, kami sarankan Anda membuat ulang database Anda dengan dump dan restore logis.

Untuk informasi selengkapnya tentang dump logis dan restore untuk Amazon Aurora MySQL, lihat Memigrasikan database yang sangat besar ke Amazon Aurora MySQL menggunakan dan. MyDumper MyLoader Untuk informasi selengkapnya tentang mengakses log kesalahan di Amazon Aurora, lihat. Memantau log Amazon Aurora Amazon

## Praktik terbaik untuk Aurora Ketersediaan tinggi saya SQL

Anda dapat menerapkan praktik terbaik berikut untuk meningkatkan ketersediaan klaster Aurora My SQL Anda.

Topik

- Menggunakan Amazon Aurora untuk Pemulihan Bencana dengan database Saya SQL
- Bermigrasi dari My SQL ke Amazon Aurora SQL My dengan waktu henti yang dikurangi
- Menghindari kinerja lambat, restart otomatis, dan failover untuk instans Aurora My DB SQL

### Menggunakan Amazon Aurora untuk Pemulihan Bencana dengan database Saya SQL

Anda dapat menggunakan Amazon Aurora dengan instans My SQL DB Anda untuk membuat cadangan di luar lokasi untuk pemulihan bencana. Untuk menggunakan Aurora untuk pemulihan bencana instans My SQL DB Anda, buat klaster DB Amazon Aurora dan jadikan itu replika baca

instans My DB Anda. SQL Ini berlaku untuk instans RDS for My SQL DB, atau SQL database Saya yang berjalan di luar AmazonRDS.

> **Important**
>
> Ketika Anda mengatur replikasi antara instans SQL DB Saya dan kluster Amazon Aurora SQL My DB, Anda harus memantau replikasi untuk memastikan bahwa itu tetap sehat dan memperbaikinya jika perlu.

Untuk petunjuk tentang cara membuat kluster Amazon Aurora My SQL DB dan menjadikannya replika baca instans My SQL DB Anda, ikuti prosedurnya di. Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda

Untuk informasi selengkapnya tentang model pemulihan bencana, lihat Cara memilih opsi pemulihan bencana terbaik untuk klaster Amazon Aurora My SQL Anda.

## Bermigrasi dari My SQL ke Amazon Aurora SQL My dengan waktu henti yang dikurangi

Saat mengimpor data dari SQL database Saya yang mendukung aplikasi langsung ke kluster Amazon Aurora SQL My DB, Anda mungkin ingin mengurangi waktu layanan terputus saat Anda bermigrasi. Untuk melakukannya, Anda dapat menggunakan prosedur yang didokumentasikan dalam Mengimpor data ke instans DB Saya SQL atau MariaDB dengan waktu henti yang dikurangi di Panduan Pengguna Layanan Amazon Relational Database Service. Prosedur ini terutama dapat membantu jika Anda beroperasi dengan basis data yang sangat besar. Anda dapat menggunakan prosedur untuk mengurangi biaya impor dengan meminimalkan jumlah data yang dilewatkan di seluruh jaringan AWS.

Prosedur ini mencantumkan langkah-langkah untuk mentransfer salinan data database Anda ke EC2 instans Amazon dan mengimpor data ke instans SQL DB Saya yang baruRDS. Karena Amazon Aurora kompatibel dengan MySQL, Anda dapat menggunakan cluster Amazon Aurora DB untuk instans RDS Amazon SQL My DB target.

## Menghindari kinerja lambat, restart otomatis, dan failover untuk instans Aurora My DB SQL

Jika Anda menjalankan beban kerja berat atau beban kerja yang melonjak melampaui sumber daya yang dialokasikan untuk instans DB Anda, Anda dapat menghabiskan sumber daya yang Anda

gunakan untuk menjalankan aplikasi dan basis data Aurora. Untuk mendapatkan metrik pada instans database Anda seperti CPU pemanfaatan, penggunaan memori, dan jumlah koneksi database yang digunakan, Anda dapat merujuk ke metrik yang disediakan oleh Amazon, Performance CloudWatch Insights, dan Enhanced Monitoring. Untuk informasi selengkapnya tentang pemantauan instans DB, lihat Memantau metrik di klaster Amazon Aurora.

Jika beban kerja Anda menghabiskan sumber daya yang Anda gunakan, instans DB Anda mungkin akan menjadi lambat, diaktifkan ulang, atau bahkan melakukan failover ke instans DB lain. Untuk menghindari hal ini, pantau pemanfaatan sumber daya Anda, periksa beban kerja yang berjalan pada instans DB Anda, dan buat pengoptimalan jika diperlukan. Jika pengoptimalan tidak meningkatkan metrik instans dan mengurangi kehabisan sumber daya, pertimbangkan untuk menaikkan skala instans DB Anda sebelum mencapai batasnya. Untuk informasi selengkapnya tentang kelas instans DB yang tersedia dan spesifikasinya, lihat Kelas instans Amazon Aurora DB.

## Rekomendasi untuk SQL fitur Saya di Aurora My SQL

Fitur berikut tersedia dalam kompatibilitas Aurora My SQL for MySQL. Namun, fitur-fitur ini memiliki masalah performa, skalabilitas, stabilitas, atau kompatibilitas di lingkungan Aurora. Oleh karena itu, kami menyarankan Anda mengikuti pedoman tertentu dalam penggunaan fitur-fitur ini. Misalnya, kami sarankan Anda tidak menggunakan fitur tertentu untuk deployment Aurora produksi.

Topik

- Menggunakan replikasi multithreaded di Aurora My SQL
- Memohon AWS Lambda fungsi menggunakan SQL fungsi saya asli
- Menghindari transaksi XA dengan Amazon Aurora My SQL
- Menyimpan kunci asing dihidupkan selama DML pernyataan
- Mengonfigurasi seberapa sering buffer log di-flush
- Meminimalkan dan memecahkan masalah Aurora Kebuntuan saya SQL

### Menggunakan replikasi multithreaded di Aurora My SQL

Dengan replikasi log biner multithreaded, sebuah SQL thread membaca peristiwa dari log relai dan mengantrekannya agar thread pekerja dapat diterapkan. SQL Thread SQL pekerja dikelola oleh utas koordinator. Peristiwa log biner diterapkan secara paralel jika memungkinkan.

Replikasi multithreaded didukung di Aurora My versi SQL 3, dan di Aurora My versi 2.12.1 dan lebih tinggi. SQL

Untuk SQL versi Aurora My yang lebih rendah dari 3,04, Aurora menggunakan replikasi single-threaded secara default ketika cluster Aurora SQL My DB digunakan sebagai replika baca untuk replikasi log biner.

Versi sebelumnya dari Aurora My SQL versi 2 mewarisi beberapa masalah mengenai replikasi multithreaded dari My Community Edition. SQL Untuk versi tersebut, kami menyarankan Anda untuk tidak menggunakan replikasi multithreaded dalam produksi.

Jika Anda menggunakan replikasi multithreaded, kami sarankan Anda mengujinya secara menyeluruh.

Untuk informasi selengkapnya tentang replikasi di Amazon Aurora, lihat Replikasi dengan Amazon Aurora. Untuk informasi lebih lanjut tentang replikasi multithreaded di Aurora My, lihat. SQL Replikasi log biner multithreaded

## Memohon AWS Lambda fungsi menggunakan SQL fungsi saya asli

Sebaiknya gunakan SQL fungsi Saya asli `lambda_sync` dan `lambda_async` untuk memanggil fungsi Lambda.

Jika Anda menggunakan prosedur `mysql.lambda_async` yang sudah dihentikan, kami sarankan agar Anda menggabungkan panggilan ke prosedur `mysql.lambda_async` dalam prosedur tersimpan. Anda dapat memanggil prosedur tersimpan ini dari berbagai sumber, seperti pemicu atau kode klien. Pendekatan ini dapat membantu menghindari masalah ketidakcocokan impedansi dan memudahkan pemrogram basis data Anda untuk menginvokasi fungsi Lambda.

Untuk informasi selengkapnya tentang menginvokasi fungsi Lambda dari Amazon Aurora, lihat Memanggil fungsi Lambda dari cluster Amazon Aurora My DB SQL.

## Menghindari transaksi XA dengan Amazon Aurora My SQL

Kami menyarankan Anda untuk tidak menggunakan transaksi eXtended Architecture (XA) dengan Aurora SQL My, karena mereka dapat menyebabkan waktu pemulihan yang lama jika XA berada dalam keadaan. `PREPARED` Jika Anda harus menggunakan transaksi XA dengan Aurora SQL My, ikuti praktik terbaik berikut:

- Jangan tinggalkan transaksi XA terbuka pada status `PREPARED`.
- Pertahankan transaksi XA sekecil mungkin.

Untuk informasi selengkapnya tentang penggunaan transaksi XA dengan MySQL, lihat transaksi XA di dokumentasi SayaSQL.

## Menyimpan kunci asing dihidupkan selama DML pernyataan

Kami sangat menyarankan agar Anda tidak menjalankan pernyataan data definition language (DDL) ketika `foreign_key_checks` variabel disetel ke `0` (off).

Jika Anda perlu menyisipkan atau memperbarui baris yang memerlukan pelanggaran sementara terhadap kunci asing, ikuti langkah-langkah berikut:

1. Atur `foreign_key_checks` ke `0`.
2. Buat bahasa manipulasi data Anda (DML) berubah.
3. Pastikan bahwa perubahan yang telah Anda selesaikan tidak melanggar batasan kunci asing.
4. Atur `foreign_key_checks` ke 1 (aktif).

Selain itu, ikuti praktik terbaik lainnya untuk batasan kunci asing:

- Pastikan bahwa aplikasi klien tidak mengatur variabel `foreign_key_checks` ke `0` sebagai bagian dari variabel `init_connect`.
- Jika pemulihan dari cadangan logis seperti `mysqldump` gagal atau tidak selesai, pastikan bahwa `foreign_key_checks` diatur ke 1 sebelum memulai operasi lain pada sesi yang sama. Cadangan logis mengatur `foreign_key_checks` ke `0` saat dimulai.

## Mengonfigurasi seberapa sering buffer log di-flush

Dalam My SQL Community Edition, untuk membuat transaksi tahan lama, buffer log InnoDB harus dibilas ke penyimpanan yang tahan lama. Anda menggunakan parameter `innodb_flush_log_at_trx_commit` untuk mengonfigurasi seberapa sering buffer log di-flush ke disk.

Saat Anda mengatur parameter `innodb_flush_log_at_trx_commit` ke nilai default 1, buffer log akan di-flush pada setiap commit transaksi. Pengaturan ini membantu menjaga basis data tetap ACIDsesuai. Kami menyarankan Anda untuk tetap mempertahankan pengaturan default 1.

Mengubah `innodb_flush_log_at_trx_commit` ke nilai nondefault dapat membantu mengurangi latensi bahasa manipulasi data (DML), tetapi mengorbankan daya tahan catatan log. Kurangnya daya

tahan ini membuat database ACID tidak sesuai. Kami menyarankan agar database Anda ACID patuh untuk menghindari risiko kehilangan data jika server restart. Untuk informasi selengkapnya tentang parameter ini, lihat innodb_flush_log_at_trx_commit dalam dokumentasi Saya. SQL

Di Aurora MySQL, pemrosesan log ulang diturunkan ke lapisan penyimpanan, jadi tidak ada pembilasan ke file log yang terjadi pada instance DB. Ketika sebuah penulisan dikeluarkan, log redo akan dikirim dari instans DB penulis langsung ke volume klaster Aurora. Satu-satunya penulisan yang melintasi jaringan adalah catatan log redo. Tidak ada halaman yang akan ditulis dari tingkat basis data.

Secara default, setiap thread yang melakukan transaksi menunggu konfirmasi dari volume cluster Aurora. Konfirmasi ini menunjukkan bahwa catatan ini dan semua catatan log redo sebelumnya telah ditulis dan mencapai kuorum. Mempersistensi catatan log dan mencapai kuorum akan membuat transaksi durabel, baik melalui commit otomatis maupun commit eksplisit. Untuk informasi selengkapnya tentang arsitektur penyimpanan Aurora, lihat Amazon Aurora storage demystified.

Aurora My SQL tidak menyiram log ke file data seperti yang dilakukan Edisi SQL Komunitas Saya. Namun, Anda dapat menggunakan parameter `innodb_flush_log_at_trx_commit` untuk melonggarkan batasan durabilitas saat menulis catatan log redo ke volume klaster Aurora.

Untuk Aurora SQL Versi saya 2:

- `innodb_flush_log_at_trx_commit=` 0 atau 2 — Database tidak menunggu konfirmasi bahwa catatan log ulang ditulis ke volume cluster Aurora.
- `innodb_flush_log_at_trx_commit=` 1 — Database menunggu konfirmasi bahwa catatan redo log ditulis ke volume cluster Aurora.

Untuk Aurora SQL Versi saya 3:

- `innodb_flush_log_at_trx_commit=` 0 — Database tidak menunggu konfirmasi bahwa catatan redo log ditulis ke volume cluster Aurora.
- `innodb_flush_log_at_trx_commit=` 1 atau 2 — Database menunggu konfirmasi bahwa catatan log ulang ditulis ke volume cluster Aurora.

Oleh karena itu, untuk mendapatkan perilaku nondefault yang sama di Aurora SQL My versi 3 yang Anda lakukan dengan nilai yang disetel ke 0 atau 2 di Aurora SQL My versi 2, atur parameternya ke 0.

Meskipun pengaturan ini dapat menurunkan DML latensi ke klien, mereka juga dapat mengakibatkan kehilangan data jika terjadi failover atau restart. Oleh karena itu, sebaiknya pertahankan parameter `innodb_flush_log_at_trx_commit` yang diatur ke nilai default 1.

Meskipun kehilangan data dapat terjadi di My SQL Community Edition dan Aurora MySQL, perilaku berbeda di setiap database karena arsitekturnya yang berbeda. Perbedaan arsitektur ini dapat menyebabkan berbagai tingkat kehilangan data. Untuk memastikan bahwa database Anda ACID sesuai, selalu atur `innodb_flush_log_at_trx_commit` ke 1.

> **Note**
>
> Di Aurora My SQL versi 3, sebelum Anda dapat mengubah `innodb_flush_log_at_trx_commit` ke nilai selain 1, Anda harus terlebih dahulu mengubah nilai `innodb_trx_commit_allow_data_loss` menjadi 1. Dengan melakukannya, berarti Anda memahami adanya risiko kehilangan data.

## Meminimalkan dan memecahkan masalah Aurora Kebuntuan saya SQL

Pengguna yang menjalankan beban kerja yang secara rutin mengalami pelanggaran batasan pada indeks sekunder unik atau kunci asing, saat memodifikasi catatan pada halaman data yang sama secara konkuren, mungkin akan lebih sering mengalami deadlock dan kehabisan waktu tunggu kunci. Kebuntuan dan batas waktu ini disebabkan oleh perbaikan bug My SQL Community Edition.

Perbaikan ini disertakan dalam My SQL Community Edition versi 5.7.26 dan lebih tinggi, dan di-backport ke Aurora SQL My versi 2.10.3 dan yang lebih tinggi. Perbaikan ini diperlukan untuk menegakkan serialisasi, dengan menerapkan penguncian tambahan untuk jenis operasi bahasa manipulasi data (DML) ini, pada perubahan yang dibuat pada catatan dalam tabel InnoDB. Masalah ini terungkap sebagai bagian dari penyelidikan masalah kebuntuan yang diperkenalkan oleh perbaikan bug My SQL Community Edition sebelumnya.

Perbaikan ini mengubah penanganan internal untuk rollback sebagian pembaruan tuple (baris) di mesin penyimpanan InnoDB. Operasi yang menghasilkan pelanggaran batasan pada kunci asing atau indeks sekunder unik menyebabkan rollback sebagian. Ini termasuk, tetapi tidak terbatas pada, pernyataan `INSERT...ON DUPLICATE KEY UPDATE`, `REPLACE INTO`, dan `INSERT IGNORE` konkuren (upsert).

Dalam konteks ini, rollback sebagian tidak mengacu pada rollback transaksi tingkat aplikasi, melainkan rollback InnoDB internal terhadap perubahan ke indeks berklaster ketika pelanggaran batasan ditemui. Misalnya, nilai kunci duplikat ditemukan selama operasi upsert.

Dalam operasi penyisipan normal, InnoDB secara atomis membuat entri indeks berklaster dan sekunder untuk setiap indeks. Jika InnoDB mendeteksi nilai duplikat pada indeks sekunder unik selama operasi upsert, entri yang disisipkan dalam indeks berklaster harus dikembalikan (rollback sebagian), dan pembaruan kemudian harus diterapkan ke baris duplikat yang ada. Selama langkah rollback sebagian internal ini, InnoDB harus mengunci setiap catatan yang dilihat sebagai bagian dari operasi. Perbaikan ini memastikan kemampuan serialisasi transaksi dengan menerapkan penguncian tambahan setelah rollback sebagian.

Meminimalkan deadlock InnoDB

Anda dapat mengambil pendekatan berikut untuk mengurangi frekuensi deadlock dalam instans basis data Anda. Contoh lainnya dapat ditemukan di SQLdokumentasi Saya.

1. Untuk mengurangi kemungkinan deadlock, lakukan transaksi segera setelah melakukan serangkaian perubahan terkait. Anda dapat melakukannya dengan memecah transaksi besar (beberapa pembaruan baris di antara commit) menjadi lebih kecil. Jika Anda memasukkan baris secara batch, cobalah untuk mengurangi ukuran penyisipan batch, terutama saat menggunakan operasi upsert yang disebutkan sebelumnya.

   Untuk mengurangi jumlah kemungkinan rollback sebagian, Anda dapat mencoba beberapa pendekatan berikut:

   a. Ganti operasi penyisipan batch dengan memasukkan satu baris pada satu waktu. Hal ini dapat mengurangi jumlah waktu saat kunci ditahan oleh transaksi yang mungkin memiliki konflik.

   b. Alih-alih menggunakan`REPLACE INTO`, tulis ulang SQL pernyataan sebagai transaksi multistatement seperti berikut:

   ```
   BEGIN;
   DELETE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

   c. Alih-alih menggunakan`INSERT...ON DUPLICATE KEY UPDATE`, tulis ulang SQL pernyataan sebagai transaksi multistatement seperti berikut:

   ```
   BEGIN;
   ```

```
SELECT rows that conflict on secondary indexes;
UPDATE conflicting rows;
INSERT new rows;
COMMIT;
```

2. Hindari transaksi yang berjalan lama, aktif atau idle, yang mungkin menahan kunci. Ini termasuk sesi SQL klien saya interaktif yang mungkin terbuka untuk waktu yang lama dengan transaksi tanpa komitmen. Saat mengoptimalkan ukuran transaksi atau ukuran batch, dampaknya dapat bervariasi tergantung pada sejumlah faktor seperti konkurensi, jumlah duplikat, dan struktur tabel. Setiap perubahan harus diterapkan dan diuji berdasarkan beban kerja Anda.
3. Dalam beberapa situasi, deadlock dapat terjadi ketika dua transaksi mencoba mengakses set data yang sama, baik dalam satu maupun beberapa tabel, dalam urutan yang berbeda-beda. Untuk mencegah hal ini, Anda dapat memodifikasi transaksi untuk mengakses data dalam urutan yang sama, sehingga menserialisasi akses. Misalnya, buat antrean transaksi yang akan diselesaikan. Pendekatan ini dapat membantu menghindari deadlock ketika beberapa transaksi terjadi secara bersamaan.
4. Menambahkan indeks yang dipilih dengan cermat ke tabel Anda dapat meningkatkan selektivitas dan mengurangi kebutuhan untuk mengakses baris, sehingga mengurangi penguncian.
5. Jika Anda mengalami penguncian celah, Anda dapat memodifikasi tingkat isolasi transaksi menjadi `READ COMMITTED` untuk sesi atau transaksi agar mencegahnya. Untuk informasi selengkapnya tentang tingkat isolasi InnoDB dan perilakunya, lihat Tingkat isolasi transaksi dalam dokumentasi SayaSQL.

> **Note**
>
> Meskipun Anda dapat mengambil tindakan pencegahan untuk mengurangi kemungkinan deadlock terjadi, deadlock adalah perilaku basis data yang sudah diperkirakan dan tetap dapat terjadi. Aplikasi harus memiliki logika yang diperlukan untuk menangani deadlock ketika terjadi. Misalnya, terapkan logika percobaan ulang dan backing-off dalam aplikasi. Yang terbaik adalah mengatasi akar penyebab masalahnya, tetapi jika deadlock memang terjadi, aplikasi memiliki opsi untuk menunggu dan mencoba lagi.

### Memantau deadlock InnoDB

Kebuntuan dapat terjadi di My SQL ketika transaksi aplikasi mencoba mengambil kunci tingkat meja dan tingkat baris dengan cara yang menghasilkan penantian melingkar. Deadlock InnoDB

yang sesekali tidak selalu menjadi masalah karena mesin penyimpanan InnoDB mendeteksi kondisi ini dengan segera dan melakukan rollback terhadap salah satu transaksi secara otomatis. Jika Anda sering mengalami deadlock, kami sarankan untuk meninjau dan memodifikasi aplikasi Anda untuk mengurangi masalah performa dan menghindari deadlock. Ketika deteksi deadlock diaktifkan (default), InnoDB secara otomatis mendeteksi deadlock transaksi dan melakukan rollback transaksi untuk mengatasi deadlock. InnoDB mencoba memilih transaksi kecil untuk di-rollback. Ukuran transaksi akan ditentukan berdasarkan jumlah baris yang disisipkan, diperbarui, atau dihapus.

- Pernyataan `SHOW ENGINE` – Pernyataan `SHOW ENGINE INNODB STATUS \G` berisi detail deadlock terbaru yang ditemui pada basis data sejak pengaktifan ulang terakhir.
- Log SQL kesalahan saya - Jika Anda sering mengalami kebuntuan di mana output `SHOW ENGINE` pernyataan tidak memadai, Anda dapat mengaktifkan parameter cluster DB innodb_print_all_deadlocks.

  Ketika parameter ini dihidupkan, informasi tentang semua kebuntuan dalam transaksi pengguna InnoDB dicatat dalam Aurora SQL My error log.
- CloudWatch Metrik Amazon — Kami juga menyarankan Anda memantau kebuntuan secara proaktif menggunakan metrik. CloudWatch `Deadlocks` Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.
- CloudWatch Log Amazon — Dengan CloudWatch Log, Anda dapat melihat metrik, menganalisis data log, dan membuat alarm waktu nyata. Untuk informasi selengkapnya, lihat Memantau kesalahan di Amazon Aurora Saya dan SQL Amazon RDS untuk Saya menggunakan SQL Amazon CloudWatch dan mengirim pemberitahuan menggunakan Amazon. SNS

  Menggunakan CloudWatch Log dengan `innodb_print_all_deadlocks` dihidupkan, Anda dapat mengonfigurasi alarm untuk memberi tahu Anda ketika jumlah kebuntuan melebihi ambang batas yang diberikan. Untuk menentukan ambang batas, kami sarankan Anda mengamati tren Anda dan menggunakan nilai berdasarkan beban kerja normal Anda.
- Wawasan Performa – Saat menggunakan Wawasan Performa, Anda dapat memantau metrik `innodb_deadlocks` dan `innodb_lock_wait_timeout`. Untuk informasi selengkapnya tentang metrik ini, lihat Penghitung non-native untuk Aurora MySQL.

# Mengevaluasi penggunaan instans DB untuk Aurora My SQL dengan metrik Amazon CloudWatch

Anda dapat menggunakan CloudWatch metrik untuk memantau throughput instans DB dan menentukan apakah kelas instans DB menyediakan sumber daya yang cukup untuk aplikasi Anda. Untuk informasi tentang batas kelas instans DB Anda, lihatSpesifikasi perangkat keras kelas instans DB untuk Aurora. Temukan spesifikasi untuk kelas instans DB Anda untuk menemukan kinerja jaringan.

Jika penggunaan instans DB Anda mendekati batas kelas instans, maka performa mungkin mulai melambat. CloudWatch Metrik dapat mengonfirmasi situasi ini sehingga Anda dapat merencanakan untuk meningkatkan skala secara manual ke kelas instance yang lebih besar.

Gabungkan nilai CloudWatch metrik berikut untuk mengetahui apakah Anda mendekati batas kelas instance:

- NetworkThroughput— Jumlah throughput jaringan yang diterima dan ditransmisikan oleh klien untuk setiap instance di cluster Aurora DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- StorageNetworkThroughput— Jumlah throughput jaringan yang diterima dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.

Tambahkan ke NetworkThroughputStorageNetworkThroughputuntuk menemukan throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB Anda. Batas kelas instans untuk instans Anda harus lebih besar dari jumlah kedua metrik gabungan ini.

Anda dapat menggunakan metrik berikut untuk meninjau detail tambahan lalu lintas jaringan dari aplikasi klien Anda saat mengirim dan menerima:

- NetworkReceiveThroughput— Jumlah throughput jaringan yang diterima dari klien oleh setiap instans DB di cluster Aurora SQL My DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- NetworkTransmitThroughput— Jumlah throughput jaringan yang dikirim ke klien oleh setiap instance di cluster Aurora DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.

- StorageNetworkReceiveThroughput— Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instance di cluster DB.
- StorageNetworkTransmitThroughput— Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster DB.

Tambahkan semua metrik ini bersama-sama untuk mengevaluasi bagaimana penggunaan jaringan Anda dibandingkan dengan batas kelas instans DB. Batas kelas instans harus lebih besar dari jumlah metrik gabungan ini.

Batas jaringan dan CPU penggunaan untuk penyimpanan terkait langsung. Ketika throughput jaringan meningkat, maka CPU penggunaannya juga meningkat. Pemantauan CPU dan penggunaan jaringan memberikan informasi tentang bagaimana dan mengapa sumber daya habis.

Untuk membantu meminimalkan penggunaan jaringan, Anda dapat mempertimbangkan hal berikut:

- Menggunakan kelas instans DB yang lebih besar.
- Membagi permintaan tulis dalam batch untuk mengurangi keseluruhan transaksi.
- Mengarahkan beban kerja hanya-baca ke instans hanya-baca.
- Menghapus indeks yang tidak digunakan.

# Memecahkan masalah Amazon Aurora Kinerja database saya SQL

Topik ini berfokus pada beberapa masalah kinerja Aurora My SQL DB yang umum, dan cara memecahkan masalah atau mengumpulkan informasi untuk memperbaiki masalah ini dengan cepat. Kami membagi kinerja database menjadi dua kategori:

- Kinerja server — Seluruh server database berjalan lebih lambat.
- Kinerja kueri — Satu atau beberapa kueri membutuhkan waktu lebih lama untuk dijalankan.

## AWS opsi pemantauan

Kami menyarankan Anda menggunakan opsi AWS pemantauan berikut untuk membantu pemecahan masalah:

- Amazon CloudWatch — Amazon CloudWatch memantau AWS sumber daya Anda dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat menggunakan CloudWatch untuk mengumpulkan dan melacak metrik, yang merupakan variabel yang dapat Anda ukur untuk sumber daya dan aplikasi Anda. Untuk informasi selengkapnya, lihat Apa itu Amazon CloudWatch? .

  Anda dapat melihat semua metrik sistem dan memproses informasi untuk instans DB Anda di. AWS Management Console Anda dapat mengonfigurasi klaster Aurora My SQL DB Anda untuk mempublikasikan data log umum, lambat, audit, dan kesalahan ke grup log di Amazon CloudWatch Logs. Ini memungkinkan Anda untuk melihat tren, memelihara log jika host terpengaruh, dan membuat garis dasar untuk kinerja “normal” untuk mengidentifikasi anomali atau perubahan dengan mudah. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
- Pemantauan yang Disempurnakan — Untuk mengaktifkan CloudWatch metrik Amazon tambahan untuk database Aurora SQL My, aktifkan Pemantauan yang Ditingkatkan. Saat Anda membuat atau memodifikasi klaster Aurora DB, pilih Aktifkan Pemantauan yang Ditingkatkan. Hal ini memungkinkan Aurora untuk mempublikasikan metrik kinerja ke. CloudWatch Beberapa metrik utama yang tersedia termasuk CPU penggunaan, koneksi database, penggunaan penyimpanan, dan latensi kueri. Ini dapat membantu mengidentifikasi kemacetan kinerja.

  Jumlah informasi yang ditransfer untuk instans DB berbanding lurus dengan granularitas yang ditentukan untuk Enhanced Monitoring. Interval pemantauan yang lebih kecil menghasilkan pelaporan metrik OS yang lebih sering dan meningkatkan biaya pemantauan. Untuk mengelola biaya, tetapkan granularitas yang berbeda untuk contoh yang berbeda di Anda. Akun AWS

Granularitas default pada pembuatan instance adalah 60 detik. Untuk informasi selengkapnya, lihat Biaya Pemantauan yang Ditingkatkan.

- Performance Insights — Anda dapat melihat semua metrik panggilan database. Ini termasuk kunci DB, menunggu, dan jumlah baris yang diproses, yang semuanya dapat Anda gunakan untuk pemecahan masalah. Saat Anda membuat atau memodifikasi klaster Aurora DB, pilih Aktifkan Performance Insights. Secara default, Performance Insights memiliki periode retensi data 7 hari, tetapi dapat disesuaikan untuk menganalisis tren kinerja jangka panjang. Untuk retensi lebih dari 7 hari, Anda perlu meningkatkan ke tingkat berbayar. Untuk informasi selengkapnya, lihat harga Performance Insights. Anda dapat mengatur periode retensi data untuk setiap instans Aurora DB secara terpisah. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

## Alasan paling umum untuk masalah kinerja SQL database Aurora My

Anda dapat menggunakan langkah-langkah berikut untuk memecahkan masalah kinerja di database Aurora My Anda. SQL Kami mencantumkan langkah-langkah ini dalam urutan logis investigasi, tetapi mereka tidak dimaksudkan untuk linier. Satu penemuan bisa melompat melintasi langkah, yang memungkinkan serangkaian jalur investigasi.

1. Beban Kerja - Memahami beban kerja database Anda.
2. Logging — Tinjau semua log database.
3. Koneksi database — Pastikan bahwa koneksi antara aplikasi dan database Anda dapat diandalkan.
4. Kinerja kueri — Periksa rencana eksekusi kueri Anda untuk melihat apakah mereka telah berubah. Perubahan kode dapat menyebabkan rencana berubah.

# Memecahkan masalah beban kerja untuk database Aurora My SQL

Beban kerja database dapat dilihat sebagai membaca dan menulis. Dengan pemahaman tentang beban kerja database “normal”, Anda dapat menyetel kueri dan server database untuk memenuhi permintaan saat berubah. Ada sejumlah alasan berbeda mengapa kinerja dapat berubah, jadi langkah pertama adalah memahami apa yang telah berubah.

- Apakah ada peningkatan versi mayor atau minor?

  Peningkatan versi utama mencakup perubahan pada kode mesin, terutama di pengoptimal, yang dapat mengubah rencana eksekusi kueri. Saat memutakhirkan versi basis data, terutama versi utama, sangat penting bagi Anda untuk menganalisis beban kerja database dan menyetelnya. Tuning dapat melibatkan mengoptimalkan dan menulis ulang kueri, atau menambahkan dan memperbarui pengaturan parameter, tergantung pada hasil pengujian. Memahami apa yang menyebabkan dampak akan memungkinkan Anda untuk mulai fokus pada area spesifik itu.

  Untuk informasi selengkapnya, lihat Apa yang baru di My SQL 8.0 dan Server serta variabel status dan opsi ditambahkan, tidak digunakan lagi, atau dihapus di My SQL 8.0 dalam dokumentasi Saya, dan. SQL Membandingkan Aurora SQL Versi saya 2 dan Aurora Versi saya 3 SQL
- Apakah ada peningkatan data yang sedang diproses (jumlah baris)?
- Apakah ada lebih banyak kueri yang berjalan secara bersamaan?
- Apakah ada perubahan skema atau database?
- Apakah ada cacat atau perbaikan kode?

Daftar Isi


- Metrik host instance
  - CPUpenggunaan
  - Penggunaan memori
  - Throughput jaringan
- Metrik basis data
- Memecahkan masalah penggunaan memori untuk database Aurora MySQL
  - Contoh 1: Penggunaan memori tinggi terus menerus
  - Contoh 2: Lonjakan memori transien
  - Contoh 3: Memori yang dapat dibebaskan turun terus menerus dan tidak direklamasi

- Memecahkan out-of-memory masalah untuk database Aurora MySQL

## Metrik host instance

Pantau metrik host instans sepertiCPU, memori, dan aktivitas jaringan untuk membantu memahami apakah telah terjadi perubahan beban kerja. Ada dua konsep utama untuk memahami perubahan beban kerja:

- Pemanfaatan — Penggunaan perangkat, seperti CPU atau disk. Ini bisa berbasis waktu atau berbasis kapasitas.
  - Berbasis waktu — Jumlah waktu sumber daya sibuk selama periode pengamatan tertentu.
  - Berbasis kapasitas — Jumlah throughput yang dapat diberikan oleh sistem atau komponen, sebagai persentase dari kapasitasnya.
- Saturasi — Sejauh mana lebih banyak pekerjaan dibutuhkan dari sumber daya daripada yang dapat diproses. Ketika penggunaan berbasis kapasitas mencapai 100%, pekerjaan ekstra tidak dapat diproses dan harus antri.

### CPUpenggunaan

Anda dapat menggunakan alat berikut untuk mengidentifikasi CPU penggunaan dan saturasi:

- CloudWatch menyediakan `CPUUtilization` metrik. Jika ini mencapai 100%, maka instance jenuh. Namun, CloudWatch metrik dirata-ratakan lebih dari 1 menit, dan tidak memiliki perincian.

  Untuk informasi selengkapnya tentang CloudWatch metrik, lihatMetrik tingkat instans untuk Amazon Aurora.
- Enhanced Monitoring menyediakan metrik yang dikembalikan oleh `top` perintah sistem operasi. Ini menunjukkan rata-rata beban dan CPU status berikut, dengan granularitas 1 detik:
  - `Idle (%)`= Waktu idle
  - `IRQ (%)`= Perangkat lunak menyela
  - `Nice (%)`= Waktu yang tepat untuk proses dengan prioritas yang baik.
  - `Steal (%)`= Waktu yang dihabiskan untuk melayani penyewa lain (terkait virtualisasi)
  - `System (%)`= Waktu sistem
  - `User (%)`= Waktu pengguna
  - `Wait (%)`= I/O tunggu

Untuk informasi selengkapnya tentang metrik Pemantauan yang Ditingkatkan, lihatMetrik OS untuk Aurora.

Penggunaan memori

Jika sistem berada di bawah tekanan memori, dan konsumsi sumber daya mencapai saturasi, Anda harus mengamati pemindaian halaman, paging, pertukaran, dan kesalahan tingkat tinggi. out-of-memory

Anda dapat menggunakan alat-alat berikut untuk mengidentifikasi penggunaan memori dan saturasi:

CloudWatch menyediakan `FreeableMemory` metrik, yang menunjukkan berapa banyak memori yang dapat direklamasi dengan membilas beberapa cache OS dan memori bebas saat ini.

Untuk informasi selengkapnya tentang CloudWatch metrik, lihatMetrik tingkat instans untuk Amazon Aurora.

Enhanced Monitoring menyediakan metrik berikut yang dapat membantu Anda mengidentifikasi masalah penggunaan memori:

- `Buffers (KB)`— Jumlah memori yang digunakan untuk buffering permintaan I/O sebelum menulis ke perangkat penyimpanan, dalam kilobyte.
- `Cached (KB)`— Jumlah memori yang digunakan untuk caching file system berbasis I/O.
- `Free (KB)`— Jumlah memori yang tidak ditetapkan, dalam kilobyte.
- `Swap`— Cached, Gratis, dan Total.

Misalnya, jika Anda melihat bahwa instans DB Anda menggunakan `Swap` memori, maka jumlah total memori untuk beban kerja Anda lebih besar daripada instans Anda saat ini tersedia. Kami merekomendasikan untuk meningkatkan ukuran instans DB Anda atau menyetel beban kerja Anda untuk menggunakan lebih sedikit memori.

Untuk informasi selengkapnya tentang metrik Pemantauan yang Ditingkatkan, lihatMetrik OS untuk Aurora.

Untuk informasi lebih rinci tentang penggunaan Skema Kinerja dan `sys` skema untuk menentukan koneksi dan komponen mana yang menggunakan memori, lihat. Memecahkan masalah penggunaan memori untuk database Aurora MySQL

Throughput jaringan

CloudWatch menyediakan metrik berikut untuk total throughput jaringan, semuanya rata-rata lebih dari 1 menit:

- `NetworkReceiveThroughput`— Jumlah throughput jaringan yang diterima dari klien oleh setiap instance di cluster Aurora DB.
- `NetworkTransmitThroughput`— Jumlah throughput jaringan yang dikirim ke klien oleh setiap instance di cluster Aurora DB.
- `NetworkThroughput`— Jumlah throughput jaringan yang diterima dari dan ditransmisikan ke klien oleh setiap instance di cluster Aurora DB.
- `StorageNetworkReceiveThroughput`— Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instance di cluster DB.
- `StorageNetworkTransmitThroughput`— Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.
- `StorageNetworkThroughput`— Jumlah throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.

Untuk informasi selengkapnya tentang CloudWatch metrik, lihatMetrik tingkat instans untuk Amazon Aurora.

Enhanced Monitoring menyediakan grafik yang `network` diterima (RX) dan ditransmisikan (TX), dengan granularitas hingga 1 detik.

Untuk informasi selengkapnya tentang metrik Pemantauan yang Ditingkatkan, lihatMetrik OS untuk Aurora.

## Metrik basis data

Periksa CloudWatch metrik berikut untuk perubahan beban kerja:

- `BlockedTransactions`— Rata-rata jumlah transaksi dalam database yang diblokir per detik.
- `BufferCacheHitRatio`— Persentase permintaan yang dilayani oleh cache buffer.
- `CommitThroughput`— Jumlah rata-rata operasi komit per detik.
- `DatabaseConnections`— Jumlah koneksi jaringan klien ke instance database.
- `Deadlocks`— Jumlah rata-rata kebuntuan dalam database per detik.

- `DMLThroughput`— Jumlah rata-rata sisipan, pembaruan, dan penghapusan per detik.
- `ResultSetCacheHitRatio`— Persentase permintaan yang dilayani oleh cache kueri.
- `RollbackSegmentHistoryListLength`— Log batalkan yang mencatat transaksi yang dilakukan dengan catatan yang ditandai hapus.
- `RowLockTime`— Total waktu yang dihabiskan untuk memperoleh kunci baris untuk tabel InnoDB.
- `SelectThroughput`— Jumlah rata-rata kueri pilih per detik.

Untuk informasi selengkapnya tentang CloudWatch metrik, lihatMetrik tingkat instans untuk Amazon Aurora.

Pertimbangkan pertanyaan-pertanyaan berikut saat memeriksa beban kerja:

1. Apakah ada perubahan terbaru di kelas instans DB, misalnya mengurangi ukuran instance dari 8xlarge menjadi 4xlarge, atau mengubah dari db.r5 ke db.r6?
2. Bisakah Anda membuat klon dan mereproduksi masalah, atau apakah itu hanya terjadi pada satu contoh itu?
3. Apakah ada kelelahan sumber daya server, tinggi CPU atau kelelahan memori? Jika ya, ini bisa berarti bahwa perangkat keras tambahan diperlukan.
4. Apakah satu atau lebih pertanyaan membutuhkan waktu lebih lama?
5. Apakah perubahan disebabkan oleh peningkatan, terutama peningkatan versi utama? Jika ya, bandingkan metrik sebelum dan sesudah peningkatan.
6. Apakah ada perubahan dalam jumlah instans DB pembaca?
7. Sudahkah Anda mengaktifkan pencatatan umum, audit, atau biner? Untuk informasi selengkapnya, lihat Logging untuk database MySQL Aurora.
8. Apakah Anda mengaktifkan, menonaktifkan, atau mengubah penggunaan replikasi log biner (binlog) Anda?
9. Apakah ada transaksi jangka panjang yang memegang sejumlah besar kunci baris? Periksa panjang daftar riwayat InnoDB (HLL) untuk indikasi transaksi yang berjalan lama.

   Untuk informasi lebih lanjut, lihat Panjang daftar riwayat InnoDB meningkat secara signifikan dan posting blog Mengapa SELECT kueri saya berjalan lambat di cluster Amazon Aurora My SQL DB saya? .

   a. Jika besar HLL disebabkan oleh transaksi tulis, itu berarti UNDO log terakumulasi (tidak dibersihkan secara teratur). Dalam transaksi tulis besar, akumulasi ini dapat tumbuh dengan

cepat. Di MySQL, UNDO disimpan di SYSTEMtablespace. SYSTEMRuang meja tidak dapat menyusut. UNDOLog dapat menyebabkan SYSTEM tablespace tumbuh menjadi beberapa GB, atau bahkan TB. Setelah pembersihan, lepaskan ruang yang dialokasikan dengan mengambil cadangan logis (dump) data, lalu impor dump ke instance DB baru.

b. Jika besar HLL disebabkan oleh transaksi baca (kueri yang berjalan lama), itu bisa berarti bahwa kueri menggunakan sejumlah besar ruang sementara. Lepaskan ruang sementara dengan me-reboot. Periksa metrik DB Performance Insights untuk setiap perubahan di Temp bagian, seperti. `created_tmp_tables` Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

10 Bisakah Anda membagi transaksi yang berjalan lama menjadi transaksi yang lebih kecil yang memodifikasi lebih sedikit baris?

11 Apakah ada perubahan dalam transaksi yang diblokir atau peningkatan kebuntuan? Periksa metrik DB Performance Insights untuk setiap perubahan variabel status di Locks bagian, seperti`innodb_row_lock_time`, `innodb_row_lock_waits` dan. `innodb_dead_locks` Gunakan interval 1 menit atau 5 menit.

12 Apakah ada peningkatan acara menunggu? Periksa Performance Insights acara tunggu dan jenis tunggu menggunakan interval 1 menit atau 5 menit. Analisis peristiwa tunggu teratas dan lihat apakah mereka berkorelasi dengan perubahan beban kerja atau pertentangan database. Misalnya, `buf_pool mutex` menunjukkan pertentangan kumpulan buffer. Untuk informasi selengkapnya, lihat Menyesuaikan Aurora MySQL dengan peristiwa tunggu.

## Memecahkan masalah penggunaan memori untuk database Aurora MySQL

Sementara CloudWatch, Enhanced Monitoring, dan Performance Insights memberikan gambaran yang baik tentang penggunaan memori di tingkat sistem operasi, seperti berapa banyak memori yang digunakan oleh proses database, mereka tidak memungkinkan Anda untuk memecah koneksi atau komponen apa di dalam mesin yang mungkin menyebabkan penggunaan memori ini.

Untuk memecahkan masalah ini, Anda dapat menggunakan skema kinerja dan skema. `sys` Di Aurora MySQL versi 3, instrumentasi memori diaktifkan secara default saat Skema Kinerja diaktifkan. Di Aurora MySQL versi 2, hanya instrumentasi memori untuk penggunaan memori Skema Kinerja yang diaktifkan secara default. Untuk informasi tentang tabel yang tersedia di Skema Kinerja untuk melacak penggunaan memori dan mengaktifkan instrumentasi memori Skema Kinerja, lihat Tabel ringkasan memori dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang penggunaan Skema Kinerja dengan Performance Insights, lihat. Ikhtisar Skema Kinerja untuk Performance Insights di Aurora

Meskipun informasi terperinci tersedia di Skema Kinerja untuk melacak penggunaan memori saat ini, skema sys MySQL memiliki tampilan di atas tabel Skema Kinerja yang dapat Anda gunakan untuk menentukan dengan cepat di mana memori digunakan.

Dalam `sys` skema, tampilan berikut tersedia untuk melacak penggunaan memori dengan koneksi, komponen, dan kueri.

| Tayang | Deskripsi |
| --- | --- |
| memory_by_host_by_current_bytes | Memberikan informasi tentang penggunaan memori mesin oleh host. Ini dapat berguna untuk mengidentifikasi server aplikasi atau host klien mana yang menghabiskan memori. |
| memory_by_thread_by_current_bytes | Memberikan informasi tentang penggunaan memori mesin dengan ID utas. ID thread di MySQL dapat berupa koneksi klien atau thread latar belakang. Anda dapat memetakan thread IDs ke IDs koneksi MySQL dengan menggunakan tampilan sys.processlist atau tabel performance_schema.threads. |
| memory_by_user_by_current_bytes | Memberikan informasi tentang penggunaan memori mesin oleh pengguna. Ini dapat berguna untuk mengidentifikasi akun pengguna atau klien mana yang menghabiskan memori. |
| memory_global_by_current_bytes | Memberikan informasi tentang penggunaan memori mesin oleh komponen mesin. Ini dapat berguna untuk mengidentifikasi penggunaan memori secara global oleh buffer atau komponen mesin. Misalnya, Anda mungkin melihat `memory/innodb/buf_buf_pool` acara untuk kumpulan buffer InnoDB, atau `memory/sql/Prepared_statement::main_mem_root` acara untuk pernyataan yang disiapkan. |

| Tayang | Deskripsi |
| --- | --- |
| memory_global_total | Memberikan gambaran umum total penggunaan memori yang dilacak di mesin database. |

Di Aurora MySQL versi 3.05 dan lebih tinggi, Anda juga dapat melacak penggunaan memori maksimum dengan intisari pernyataan di tabel ringkasan pernyataan Skema Kinerja. Tabel ringkasan pernyataan berisi intisari pernyataan yang dinormalisasi dan statistik agregat pada pelaksanaannya. `MAX_TOTAL_MEMORY`Kolom dapat membantu Anda mengidentifikasi memori maksimum yang digunakan oleh intisari kueri sejak statistik terakhir disetel ulang, atau karena instance database dimulai ulang. Ini dapat berguna dalam mengidentifikasi kueri tertentu yang mungkin menghabiskan banyak memori.

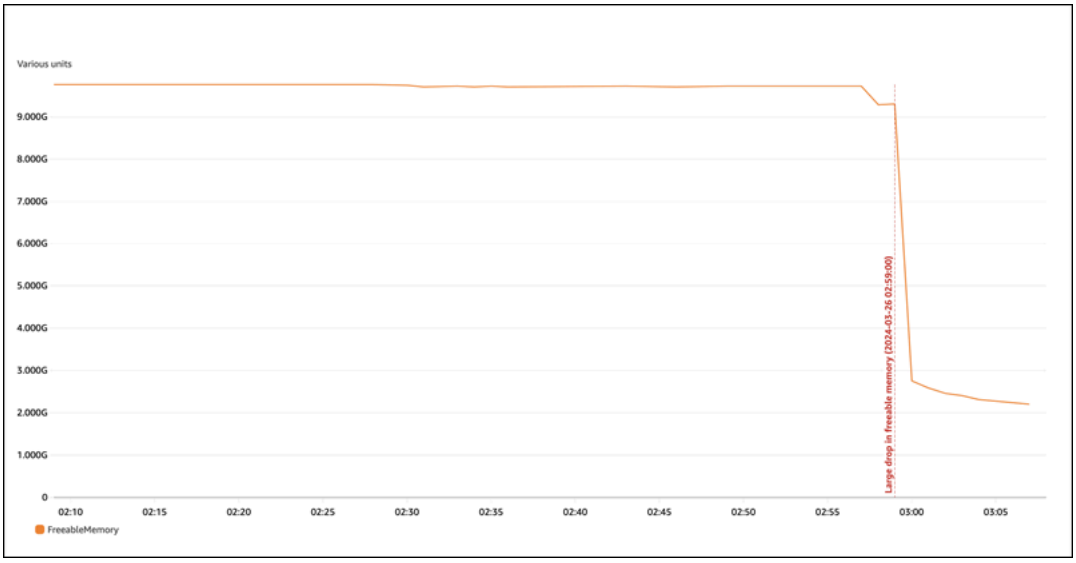
> **Note**
>
> Skema dan `sys` skema Kinerja menunjukkan kepada Anda penggunaan memori saat ini di server, dan tanda air tinggi untuk memori yang dikonsumsi per koneksi dan komponen mesin. Karena Skema Kinerja dipertahankan dalam memori, informasi diatur ulang ketika instans DB dimulai ulang. Untuk mempertahankan riwayat dari waktu ke waktu, sebaiknya Anda mengonfigurasi pengambilan dan penyimpanan data ini di luar Skema Kinerja.

Topik

- Contoh 1: Penggunaan memori tinggi terus menerus
- Contoh 2: Lonjakan memori transien
- Contoh 3: Memori yang dapat dibebaskan turun terus menerus dan tidak direklamasi

Contoh 1: Penggunaan memori tinggi terus menerus

Melihat `FreeableMemory` secara global CloudWatch, kita dapat melihat bahwa penggunaan memori sangat meningkat pada 2024-03-26 02:59 UTC.




Ini tidak memberi tahu kita keseluruhan gambar. Untuk menentukan komponen mana yang paling banyak menggunakan memori, Anda dapat masuk ke database dan melihatnya`sys.memory_global_by_current_bytes`. Tabel ini berisi daftar peristiwa memori yang dilacak MySQL, bersama dengan informasi tentang alokasi memori per peristiwa. Setiap peristiwa pelacakan memori dimulai dengan`memory/%`, diikuti oleh informasi lain tentang komponen/fitur mesin mana yang terkait dengan peristiwa tersebut.

Misalnya, `memory/performance_schema/%` adalah untuk peristiwa memori yang terkait dengan Skema Kinerja, `memory/innodb/%` adalah untuk InnoDB, dan sebagainya. Untuk informasi selengkapnya tentang konvensi penamaan acara, lihat konvensi penamaan instrumen Skema Kinerja dalam dokumentasi MySQL.

Dari kueri berikut, kita dapat menemukan kemungkinan pelakunya berdasarkan`current_alloc`, tetapi kita juga dapat melihat banyak `memory/performance_schema/%` peristiwa.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+-----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+-----------------+
```

```
| memory/sql/Prepared_statement::main_mem_root                                  |
 512817 | 4.91 GiB      | 10.04 KiB         |     512823 | 4.91 GiB   | 10.04 KiB
 |
| memory/performance_schema/prepared_statements_instances                       |
   252 | 488.25 MiB    | 1.94 MiB          |        252 | 488.25 MiB | 1.94 MiB       |
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error           |
  1028 | 52.27 MiB     | 52.06 KiB         |       1028 | 52.27 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     4 | 47.25 MiB     | 11.81 MiB         |          4 | 47.25 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/performance_schema/memory_summary_by_thread_by_event_name             |
     4 | 31.64 MiB     | 7.91 MiB          |          4 | 31.64 MiB  | 7.91 MiB       |
| memory/innodb/memory                                                          |
 15227 | 27.44 MiB     | 1.85 KiB          |      20619 | 33.33 MiB  | 1.66 KiB       |
| memory/sql/String::value                                                      |
 74411 | 21.85 MiB     |  307 bytes        |      76867 | 25.54 MiB  |  348 bytes     |
| memory/sql/TABLE                                                              |
  8381 | 21.03 MiB     | 2.57 KiB          |       8381 | 21.03 MiB  | 2.57 KiB       |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
10 rows in set (0.02 sec)
```

Kami menyebutkan sebelumnya bahwa Skema Kinerja disimpan dalam memori, yang berarti bahwa itu juga dilacak dalam instrumentasi `performance_schema` memori.

> **Note**
>
> Jika Anda menemukan bahwa Skema Kinerja menggunakan banyak memori, dan ingin membatasi penggunaan memorinya, Anda dapat menyetel parameter database berdasarkan kebutuhan Anda. Untuk informasi selengkapnya, lihat Model alokasi memori Skema Kinerja dalam dokumentasi MySQL.

Untuk keterbacaan, Anda dapat menjalankan kembali kueri yang sama tetapi mengecualikan peristiwa Skema Kinerja. Outputnya menunjukkan hal berikut:

- Konsumen memori utama adalah`memory/sql/Prepared_statement::main_mem_root`.

- `current_alloc`Kolom memberitahu kita bahwa MySQL memiliki 4,91 GiB saat ini dialokasikan untuk acara ini.
- Ini `high_alloc column` memberi tahu kita bahwa 4,91 GiB adalah tanda `current_alloc` air tinggi sejak statistik terakhir diatur ulang atau sejak server dimulai ulang. Ini berarti `memory/sql/Prepared_statement::main_mem_root` itu pada nilai tertinggi.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name NOT LIKE
 'memory/performance_schema/%' LIMIT 10;

+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root  |        512817 | 4.91 GiB      | 10.04
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
| memory/innodb/hash0hash                       |             4 | 79.07 MiB     | 19.77
 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/memory                          |         17096 | 31.68 MiB     | 1.90
 KiB          |      22498 | 37.60 MiB  | 1.71 KiB       |
| memory/sql/String::value                      |        122277 | 27.94 MiB     |  239
 bytes        |     124699 | 29.47 MiB  |  247 bytes     |
| memory/sql/TABLE                              |          9927 | 24.67 MiB     | 2.55
 KiB          |       9929 | 24.68 MiB  | 2.55 KiB       |
| memory/innodb/lock0lock                       |          8888 | 19.71 MiB     | 2.27
 KiB          |       8888 | 19.71 MiB  | 2.27 KiB       |
| memory/sql/Prepared_statement::infrastructure |        257623 | 16.24 MiB     |   66
 bytes        |     257631 | 16.24 MiB  |   66 bytes     |
| memory/mysys/KEY_CACHE                        |             3 | 16.00 MiB     | 5.33
 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/innodb/sync0arr                        |             3 | 7.03 MiB      | 2.34
 MiB          |          3 | 7.03 MiB   | 2.34 MiB       |
| memory/sql/THD::main_mem_root                 |           815 | 6.56 MiB      | 8.24
 KiB          |        849 | 7.19 MiB   | 8.67 KiB       |
+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
10 rows in set (0.06 sec)
```

Dari nama acara, kita dapat mengetahui bahwa memori ini digunakan untuk pernyataan yang disiapkan. Jika Anda ingin melihat koneksi mana yang menggunakan memori ini, Anda dapat memeriksa memory_by_thread_by_current_bytes.

Dalam contoh berikut, setiap sambungan memiliki sekitar 7 MiB yang dialokasikan, dengan tanda air tinggi sekitar 6,29 MiB (). `current_max_alloc` Ini masuk akal, karena contohnya menggunakan `sysbench` dengan 80 tabel dan 800 koneksi dengan pernyataan yang disiapkan. Jika Anda ingin mengurangi penggunaan memori dalam skenario ini, Anda dapat mengoptimalkan penggunaan aplikasi Anda dari pernyataan yang disiapkan untuk mengurangi konsumsi memori.

```
mysql> SELECT * FROM sys.memory_by_thread_by_current_bytes;

+-----------+-----------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
| thread_id | user                                    | current_count_used |
 current_allocated | current_avg_alloc | current_max_alloc | total_allocated |
+-----------+-----------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
|        46 | rdsadmin@localhost                      |                405 | 8.47 MiB
         | 21.42 KiB         | 8.00 MiB          | 155.86 MiB      |
|        61 | reinvent@10.0.4.4                       |               1749 | 6.72 MiB
         | 3.93 KiB          | 6.29 MiB          | 14.24 MiB       |
|       101 | reinvent@10.0.4.4                       |               1845 | 6.71 MiB
         | 3.72 KiB          | 6.29 MiB          | 14.50 MiB       |
|        55 | reinvent@10.0.4.4                       |               1674 | 6.68 MiB
         | 4.09 KiB          | 6.29 MiB          | 14.13 MiB       |
|        57 | reinvent@10.0.4.4                       |               1416 | 6.66 MiB
         | 4.82 KiB          | 6.29 MiB          | 13.52 MiB       |
|       112 | reinvent@10.0.4.4                       |               1759 | 6.66 MiB
         | 3.88 KiB          | 6.29 MiB          | 14.17 MiB       |
|        66 | reinvent@10.0.4.4                       |               1428 | 6.64 MiB
         | 4.76 KiB          | 6.29 MiB          | 13.47 MiB       |
|        75 | reinvent@10.0.4.4                       |               1389 | 6.62 MiB
         | 4.88 KiB          | 6.29 MiB          | 13.40 MiB       |
|       116 | reinvent@10.0.4.4                       |               1333 | 6.61 MiB
         | 5.08 KiB          | 6.29 MiB          | 13.21 MiB       |
|        90 | reinvent@10.0.4.4                       |               1448 | 6.59 MiB
         | 4.66 KiB          | 6.29 MiB          | 13.58 MiB       |
|        98 | reinvent@10.0.4.4                       |               1440 | 6.57 MiB
         | 4.67 KiB          | 6.29 MiB          | 13.52 MiB       |
|        94 | reinvent@10.0.4.4                       |               1433 | 6.57 MiB
         | 4.69 KiB          | 6.29 MiB          | 13.49 MiB       |
```

```
|        62 | reinvent@10.0.4.4                       |              1323 | 6.55 MiB
         | 5.07 KiB          | 6.29 MiB          | 13.48 MiB       |
|        87 | reinvent@10.0.4.4                       |              1323 | 6.55 MiB
         | 5.07 KiB          | 6.29 MiB          | 13.25 MiB       |
|        99 | reinvent@10.0.4.4                       |              1346 | 6.54 MiB
         | 4.98 KiB          | 6.29 MiB          | 13.24 MiB       |
|       105 | reinvent@10.0.4.4                       |              1347 | 6.54 MiB
         | 4.97 KiB          | 6.29 MiB          | 13.34 MiB       |
|        73 | reinvent@10.0.4.4                       |              1335 | 6.54 MiB
         | 5.02 KiB          | 6.29 MiB          | 13.23 MiB       |
|        54 | reinvent@10.0.4.4                       |              1510 | 6.53 MiB
         | 4.43 KiB          | 6.29 MiB          | 13.49 MiB       |
.                                                                                .
.                                                                                .
.                                                                                .
|       812 | reinvent@10.0.4.4                       |              1259 | 6.38 MiB
         | 5.19 KiB          | 6.29 MiB          | 13.05 MiB       |
|       214 | reinvent@10.0.4.4                       |              1279 | 6.38 MiB
         | 5.10 KiB          | 6.29 MiB          | 12.90 MiB       |
|       325 | reinvent@10.0.4.4                       |              1254 | 6.38 MiB
         | 5.21 KiB          | 6.29 MiB          | 12.99 MiB       |
|       705 | reinvent@10.0.4.4                       |              1273 | 6.37 MiB
         | 5.13 KiB          | 6.29 MiB          | 13.03 MiB       |
|       530 | reinvent@10.0.4.4                       |              1268 | 6.37 MiB
         | 5.15 KiB          | 6.29 MiB          | 12.92 MiB       |
|       307 | reinvent@10.0.4.4                       |              1263 | 6.37 MiB
         | 5.17 KiB          | 6.29 MiB          | 12.87 MiB       |
|       738 | reinvent@10.0.4.4                       |              1260 | 6.37 MiB
         | 5.18 KiB          | 6.29 MiB          | 13.00 MiB       |
|       819 | reinvent@10.0.4.4                       |              1252 | 6.37 MiB
         | 5.21 KiB          | 6.29 MiB          | 13.01 MiB       |
|        31 | innodb/srv_purge_thread                 |             17810 | 3.14 MiB
         |  184 bytes        | 2.40 MiB          | 205.69 MiB      |
|        38 | rdsadmin@localhost                      |               599 | 1.76 MiB
         | 3.01 KiB          | 1.00 MiB          | 25.58 MiB       |
|         1 | sql/main                                |              3756 | 1.32 MiB
         |  367 bytes        | 355.78 KiB        | 6.19 MiB        |
|       854 | rdsadmin@localhost                      |                46 | 1.08 MiB
         | 23.98 KiB         | 1.00 MiB          | 5.10 MiB        |
|        30 | innodb/clone_gtid_thread                |              1596 | 573.14
 KiB        |  367 bytes        | 254.91 KiB        | 970.69 KiB      |
```

```
|        40 | rdsadmin@localhost                          |                235 | 245.19
 KiB        | 1.04 KiB          | 128.88 KiB        | 808.64 KiB      |
|       853 | rdsadmin@localhost                          |                 96 | 94.63
 KiB         | 1009 bytes        | 29.73 KiB         | 422.45 KiB      |
|        36 | rdsadmin@localhost                          |                 33 | 36.29
 KiB         | 1.10 KiB          | 16.08 KiB         | 74.15 MiB       |
|        33 | sql/event_scheduler                         |                  3 | 16.27
 KiB         | 5.42 KiB          | 16.04 KiB         | 16.27 KiB       |
|        35 | sql/compress_gtid_table                     |                  8 | 14.20
 KiB         | 1.77 KiB          | 8.05 KiB          | 18.62 KiB       |
|        25 | innodb/fts_optimize_thread                  |                 12 | 1.86 KiB
          |  158 bytes        |  648 bytes        | 1.98 KiB        |
|        23 | innodb/srv_master_thread                    |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 24.40 KiB       |
|        24 | innodb/dict_stats_thread                    |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 1.35 KiB        |
|         5 | innodb/io_read_thread                       |                  1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         6 | innodb/io_read_thread                       |                  1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         2 | sql/aws_oscar_log_level_monitor             |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         4 | innodb/io_ibuf_thread                       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         7 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         8 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         9 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        10 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        11 | innodb/srv_lra_thread                       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        12 | innodb/srv_akp_thread                       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        18 | innodb/srv_lock_timeout_thread              |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |  248 bytes      |
|        19 | innodb/srv_error_monitor_thread             |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        20 | innodb/srv_monitor_thread                   |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        21 | innodb/buf_resize_thread                    |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
```

```
|        22 | innodb/btr_search_sys_toggle_thread       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        32 | innodb/dict_persist_metadata_table_thread |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        34 | sql/signal_handler                        |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
+-----------+-------------------------------------------+-------------------
+-------------------+-------------------+-------------------+-----------------+
831 rows in set (2.48 sec)
```

Seperti disebutkan sebelumnya, nilai thread ID (`thd_id`) di sini dapat merujuk ke thread latar belakang server atau koneksi database. Jika Anda ingin memetakan nilai ID thread ke koneksi database IDs, Anda dapat menggunakan `performance_schema.threads` tabel atau `sys.processlist` tampilan, di `conn_id` mana ID koneksi.

```
mysql> SELECT thd_id,conn_id,user,db,command,state,time,last_wait FROM sys.processlist
 WHERE user='reinvent@10.0.4.4';

+--------+---------+------------------+----------+---------+----------------+------
+-----------------------------------------------+
| thd_id | conn_id | user             | db       | command | state          | time |
 last_wait                                     |
+--------+---------+------------------+----------+---------+----------------+------
+-----------------------------------------------+
|    590 |     562 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                        |
|    578 |     550 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                          |
|    579 |     551 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                        |
|    580 |     552 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    581 |     553 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    582 |     554 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                          |
|    583 |     555 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                          |
|    584 |     556 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    585 |     557 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                        |
```

```
|    586 |     558 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                          |
|    587 |     559 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                             |
.
                                                   .
.
                                                   .
.
                                                   .
|    323 |     295 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                               |
|    324 |     296 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                          |
|    325 |     297 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                             |
|    326 |     298 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                          |
|    438 |     410 | reinvent@10.0.4.4 | sysbench | Execute | System lock    |    0 |
 wait/lock/table/sql/handler                        |
|    280 |     252 | reinvent@10.0.4.4 | sysbench | Sleep   | starting       |    0 |
 wait/io/socket/sql/client_connection               |
|     98 |      70 | reinvent@10.0.4.4 | sysbench | Query   | freeing items  |    0 |
 NULL                                               |
+--------+---------+-------------------+----------+---------+----------------+------
+-------------------------------------------------+
804 rows in set (5.51 sec)
```

Sekarang kita menghentikan `sysbench` beban kerja, yang menutup koneksi dan melepaskan memori. Memeriksa peristiwa lagi, kami dapat mengonfirmasi bahwa memori dilepaskan, tetapi `high_alloc` masih memberi tahu kami apa tanda air tinggi itu. `high_alloc`Kolom dapat sangat berguna dalam mengidentifikasi lonjakan singkat dalam penggunaan memori, di mana Anda mungkin tidak dapat segera mengidentifikasi penggunaan`current_alloc`, yang hanya menampilkan memori yang dialokasikan saat ini.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
```

```
+----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |           17 | 253.80 KiB    | 14.93
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
+----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

Jika Anda ingin mengatur ulang`high_alloc`, Anda dapat memotong tabel ringkasan `performance_schema` memori, tetapi ini mengatur ulang semua instrumentasi memori. Untuk informasi selengkapnya, lihat Karakteristik tabel umum Skema Kinerja dalam dokumentasi MySQL.

Dalam contoh berikut, kita dapat melihat `high_alloc` bahwa reset setelah pemotongan.

```
mysql> TRUNCATE `performance_schema`.`memory_summary_global_by_event_name`;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

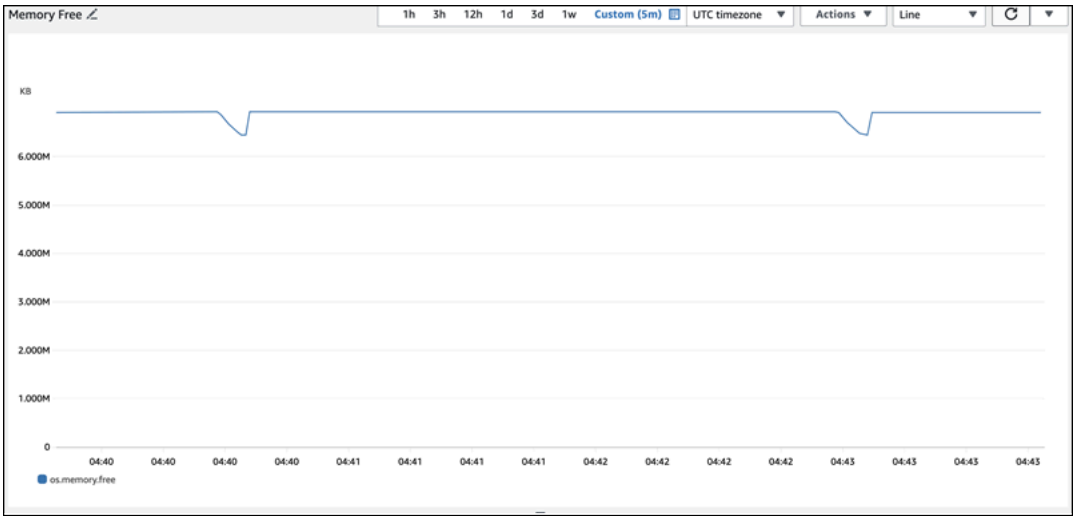
+----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
| event_name                                   | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |           17 | 253.80 KiB    | 14.93
 KiB         |         17 | 253.80 KiB | 14.93 KiB      |
+----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

Contoh 2: Lonjakan memori transien

Kejadian umum lainnya adalah lonjakan pendek dalam penggunaan memori pada server database. Ini bisa berupa penurunan periodik dalam memori yang dapat dibebaskan yang sulit dipecahkan masalah saat digunakan `current_allocsys.memory_global_by_current_bytes`, karena memori telah dibebaskan.

> **Note**
>
> Jika statistik Skema Kinerja telah disetel ulang, atau instance database telah dimulai ulang, informasi ini tidak akan tersedia di `sys` atau p. `erformance_schema` Untuk menyimpan informasi ini, sebaiknya Anda mengonfigurasi pengumpulan metrik eksternal.

Grafik `os.memory.free` metrik berikut di Enhanced Monitoring menunjukkan lonjakan singkat 7 detik dalam penggunaan memori. Enhanced Monitoring memungkinkan Anda untuk memantau pada interval sesingkat 1 detik, yang sempurna untuk menangkap lonjakan sementara seperti ini.




Untuk membantu mendiagnosis penyebab penggunaan memori di sini, kita dapat menggunakan kombinasi tampilan ringkasan `sys` memori dan tabel ringkasan pernyataan Skema Kinerja untuk mencoba mengidentifikasi sesi dan koneksi yang menyinggung. `high_alloc`

Seperti yang diharapkan, karena penggunaan memori saat ini tidak tinggi, kami tidak dapat melihat pelanggar utama dalam tampilan `sys` skema di bawah. `current_alloc`

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+-----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
```

```
+-------------------------------------------------------------------------------
+---------------+---------------+------------------+------------+------------
+----------------+
| memory/innodb/hash0hash                                                      |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/os0event                                                       |
 439372 | 60.34 MiB     |  144 bytes        |     439372 | 60.34 MiB  |  144 bytes
 |
| memory/performance_schema/events_statements_summary_by_digest               |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/mysys/KEY_CACHE                                                       |
     3 | 16.00 MiB     | 5.33 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/performance_schema/events_statements_history_long                     |
     1 | 14.34 MiB     | 14.34 MiB         |          1 | 14.34 MiB  | 14.34 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error           |
   257 | 13.07 MiB     | 52.06 KiB         |        257 | 13.07 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     1 | 11.81 MiB     | 11.81 MiB         |          1 | 11.81 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest.digest_text   |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.digest_text        |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.sql_text           |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
+-------------------------------------------------------------------------------
+---------------+---------------+------------------+------------+------------
+----------------+
10 rows in set (0.01 sec)
```

Memperluas tampilan ke order by`high_alloc`, sekarang kita dapat melihat bahwa `memory/temptable/physical_ram` komponen tersebut adalah kandidat yang sangat baik di sini. Pada level tertinggi, ia mengkonsumsi 515,00 MiB.

Seperti namanya, `memory/temptable/physical_ram` instrumen penggunaan memori untuk mesin `TEMP` penyimpanan di MySQL, yang diperkenalkan di MySQL 8.0. Untuk informasi selengkapnya tentang cara MySQL menggunakan tabel sementara, lihat Penggunaan tabel sementara internal di MySQL dalam dokumentasi MySQL.

> **Note**
>
> Kami menggunakan `sys.x$memory_global_by_current_bytes` tampilan dalam contoh ini.

```
mysql> SELECT event_name, format_bytes(current_alloc) AS "currently allocated",
 sys.format_bytes(high_alloc) AS "high-water mark"
FROM sys.x$memory_global_by_current_bytes ORDER BY high_alloc DESC LIMIT 10;

+-------------------------------------------------------------------------------
+--------------------+-----------------+
| event_name                                                                    |
 currently allocated | high-water mark |
+-------------------------------------------------------------------------------
+--------------------+-----------------+
| memory/temptable/physical_ram                                                 | 4.00
 MiB            | 515.00 MiB      |
| memory/innodb/hash0hash                                                       | 79.07
 MiB           | 79.07 MiB       |
| memory/innodb/os0event                                                        | 63.95
 MiB           | 63.95 MiB       |
| memory/performance_schema/events_statements_summary_by_digest                | 40.28
 MiB           | 40.28 MiB       |
| memory/mysys/KEY_CACHE                                                        | 16.00
 MiB           | 16.00 MiB       |
| memory/performance_schema/events_statements_history_long                      | 14.34
 MiB           | 14.34 MiB       |
| memory/performance_schema/events_errors_summary_by_thread_by_error           | 13.07
 MiB           | 13.07 MiB       |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name | 11.81
 MiB           | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest.digest_text    | 9.77
 MiB            | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.sql_text            | 9.77
 MiB            | 9.77 MiB        |
+-------------------------------------------------------------------------------
+--------------------+-----------------+
10 rows in set (0.00 sec)
```

DiContoh 1: Penggunaan memori tinggi terus menerus, kami memeriksa penggunaan memori saat ini untuk setiap koneksi untuk menentukan koneksi mana yang bertanggung jawab untuk menggunakan

memori yang dimaksud. Dalam contoh ini, memori sudah dibebaskan, jadi memeriksa penggunaan memori untuk koneksi saat ini tidak berguna.

Untuk menggali lebih dalam dan menemukan pernyataan yang menyinggung, pengguna, dan host, kami menggunakan Skema Kinerja. Skema Kinerja berisi beberapa tabel ringkasan pernyataan yang diiris oleh dimensi yang berbeda seperti nama acara, intisari pernyataan, host, utas, dan pengguna. Setiap tampilan akan memungkinkan Anda menggali lebih dalam di mana pernyataan tertentu dijalankan dan apa yang mereka lakukan. Bagian ini difokuskan`MAX_TOTAL_MEMORY`, tetapi Anda dapat menemukan informasi lebih lanjut tentang semua kolom yang tersedia dalam dokumentasi tabel ringkasan pernyataan Skema Kinerja.

```
mysql> SHOW TABLES IN performance_schema LIKE 'events_statements_summary_%';

+------------------------------------------------------------+
| Tables_in_performance_schema (events_statements_summary_%) |
+------------------------------------------------------------+
| events_statements_summary_by_account_by_event_name         |
| events_statements_summary_by_digest                        |
| events_statements_summary_by_host_by_event_name            |
| events_statements_summary_by_program                       |
| events_statements_summary_by_thread_by_event_name          |
| events_statements_summary_by_user_by_event_name            |
| events_statements_summary_global_by_event_name             |
+------------------------------------------------------------+
7 rows in set (0.00 sec)
```

Pertama kita periksa `events_statements_summary_by_digest` untuk melihat`MAX_TOTAL_MEMORY`.

Dari sini kita dapat melihat yang berikut:

- Kueri dengan intisari `20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a` tampaknya menjadi kandidat yang baik untuk penggunaan memori ini. `MAX_TOTAL_MEMORY`Itu adalah 537450710, yang cocok dengan tanda air tinggi yang kami lihat untuk acara tersebut. `memory/temptable/physical_ram` `sys.x$memory_global_by_current_bytes`
- Telah dijalankan empat kali (`COUNT_STAR`), pertama pada 2024-03-26 04:08:34.943 256, dan terakhir pada 2024-03-26 04:43:06.998 310.

```
mysql> SELECT SCHEMA_NAME,DIGEST,COUNT_STAR,MAX_TOTAL_MEMORY,FIRST_SEEN,LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest ORDER BY MAX_TOTAL_MEMORY
 DESC LIMIT 5;

+-------------+------------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
| SCHEMA_NAME | DIGEST                                                           |
 COUNT_STAR | MAX_TOTAL_MEMORY | FIRST_SEEN                | LAST_SEEN
  |
+-------------+------------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
| sysbench    | 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a |
     4 |        537450710 | 2024-03-26 04:08:34.943256 | 2024-03-26 04:43:06.998310 |
| NULL        | f158282ea0313fefd0a4778f6e9b92fc7d1e839af59ebd8c5eea35e12732c45d |
     4 |          3636413 | 2024-03-26 04:29:32.712348 | 2024-03-26 04:36:26.269329 |
| NULL        | 0046bc5f642c586b8a9afd6ce1ab70612dc5b1fd2408fa8677f370c1b0ca3213 |
     2 |          3459965 | 2024-03-26 04:31:37.674008 | 2024-03-26 04:32:09.410718 |
| NULL        | 8924f01bba3c55324701716c7b50071a60b9ceaf17108c71fd064c20c4ab14db |
     1 |          3290981 | 2024-03-26 04:31:49.751506 | 2024-03-26 04:31:49.751506 |
| NULL        | 90142bbcb50a744fcec03a1aa336b2169761597ea06d85c7f6ab03b5a4e1d841 |
     1 |          3131729 | 2024-03-26 04:15:09.719557 | 2024-03-26 04:15:09.719557 |
+-------------+------------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
5 rows in set (0.00 sec)
```

Sekarang kita tahu intisari yang menyinggung, kita bisa mendapatkan detail lebih lanjut seperti teks kueri, pengguna yang menjalankannya, dan di mana itu dijalankan. Berdasarkan teks intisari yang dikembalikan, kita dapat melihat bahwa ini adalah ekspresi tabel umum (CTE) yang membuat empat tabel sementara dan melakukan empat pemindaian tabel, yang sangat tidak efisien.

```
mysql> SELECT
 SCHEMA_NAME,DIGEST_TEXT,QUERY_SAMPLE_TEXT,MAX_TOTAL_MEMORY,SUM_ROWS_SENT,SUM_ROWS_EXAMINED,SUM
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST='20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a'\G;

*************************** 1. row ***************************
           SCHEMA_NAME: sysbench
           DIGEST_TEXT: WITH RECURSIVE `cte` ( `n` ) AS ( SELECT ? FROM `sbtest1` UNION
 ALL SELECT `id` + ? FROM `sbtest1` ) SELECT * FROM `cte`
```

```
     QUERY_SAMPLE_TEXT: WITH RECURSIVE cte (n) AS (   SELECT 1  from sbtest1 UNION ALL
  SELECT id + 1 FROM sbtest1) SELECT * FROM cte
      MAX_TOTAL_MEMORY: 537450710
         SUM_ROWS_SENT: 80000000
     SUM_ROWS_EXAMINED: 80000000
SUM_CREATED_TMP_TABLES: 4
     SUM_NO_INDEX_USED: 4
1 row in set (0.01 sec)
```

Untuk informasi selengkapnya tentang `events_statements_summary_by_digest` tabel dan tabel ringkasan pernyataan Skema Kinerja lainnya, lihat tabel ringkasan pernyataan dalam dokumentasi MySQL.

Anda juga dapat menjalankan pernyataan EXPLORE atau EXPLORE ANALYSIS untuk melihat detail selengkapnya.

> **Note**
>
> `EXPLAIN ANALYZE`dapat memberikan lebih banyak informasi daripada`EXPLAIN`, tetapi juga menjalankan kueri, jadi berhati-hatilah.

```
-- EXPLAIN
mysql> EXPLAIN WITH RECURSIVE cte (n) AS (SELECT 1  FROM sbtest1 UNION ALL SELECT id +
 1 FROM sbtest1) SELECT * FROM cte;

+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
| id | select_type | table      | partitions | type  | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra       |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL   | NULL          | NULL | NULL    |
 NULL | 19221520 |   100.00 | NULL        |
|  2 | DERIVED     | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
|  3 | UNION       | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)
```

```
-- EXPLAIN format=tree
mysql> EXPLAIN format=tree WITH RECURSIVE cte (n) AS (SELECT 1 FROM sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
1 row in set (0.00 sec)

-- EXPLAIN ANALYZE
mysql> EXPLAIN ANALYZE WITH RECURSIVE cte (n) AS (SELECT 1 from sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6) (actual
 time=6666..9201 rows=20e+6 loops=1)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6) (actual
 time=6666..6666 rows=20e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0365..2006 rows=10e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0311..2494 rows=10e+6 loops=1)
1 row in set (10.53 sec)
```

Tapi siapa yang menjalankannya? Kita dapat melihat di Skema Kinerja yang dimiliki `destructive_operator` MAX_TOTAL_MEMORY pengguna 537450710, yang lagi-lagi cocok dengan hasil sebelumnya.

> **Note**
>
> Skema Kinerja disimpan dalam memori, jadi tidak boleh diandalkan sebagai satu-satunya sumber untuk audit. Jika Anda perlu mempertahankan riwayat pernyataan yang dijalankan, dan dari mana pengguna, kami sarankan Anda mengaktifkan Audit Lanjutan Aurora. Jika Anda juga perlu menyimpan informasi tentang penggunaan memori, kami sarankan Anda mengonfigurasi pemantauan untuk mengekspor dan menyimpan nilai-nilai ini.

```
mysql> SELECT USER,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_user_by_event_name
```

```
ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+----------------------+---------------------------+------------+------------------+
| USER                 | EVENT_NAME                | COUNT_STAR | MAX_TOTAL_MEMORY |
+----------------------+---------------------------+------------+------------------+
| destructive_operator | statement/sql/select      |          4 |        537450710 |
| rdsadmin             | statement/sql/select      |       4172 |          3290981 |
| rdsadmin             | statement/sql/show_tables |          2 |          3615821 |
| rdsadmin             | statement/sql/show_fields |          2 |          3459965 |
| rdsadmin             | statement/sql/show_status |         75 |          1914976 |
+----------------------+---------------------------+------------+------------------+
5 rows in set (0.00 sec)

mysql> SELECT HOST,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_host_by_event_name
WHERE HOST != 'localhost' AND COUNT_STAR>0 ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+------------+----------------------+------------+------------------+
| HOST       | EVENT_NAME           | COUNT_STAR | MAX_TOTAL_MEMORY |
+------------+----------------------+------------+------------------+
| 10.0.8.231 | statement/sql/select |          4 |        537450710 |
+------------+----------------------+------------+------------------+
1 row in set (0.00 sec)
```

Contoh 3: Memori yang dapat dibebaskan turun terus menerus dan tidak direklamasi

Mesin database InnoDB menggunakan berbagai peristiwa pelacakan memori khusus untuk komponen yang berbeda. Peristiwa spesifik ini memungkinkan pelacakan granular penggunaan memori di subsistem InnoDB utama, misalnya:

- `memory/innodb/buf0buf`— Didedikasikan untuk memantau alokasi memori untuk kolam buffer InnoDB.
- `memory/innodb/ibuf0ibuf`— Secara khusus melacak perubahan memori yang terkait dengan buffer perubahan InnoDB.

Untuk mengidentifikasi konsumen memori teratas, kami dapat menanyakan`sys.memory_global_by_current_bytes`:

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;
```

```
+----------------------------------------------------------------+---------------+
| event_name                                                     | current_alloc |
+----------------------------------------------------------------+---------------+
| memory/innodb/memory                                           | 5.28 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                         | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                               | 388.95 MiB    |
| memory/innodb/std                                              | 226.88 MiB    |
| memory/innodb/fil0fil                                          | 198.49 MiB    |
| memory/sql/binlog_io_cache                                     | 128.00 MiB    |
| memory/innodb/mem0mem                                          | 96.82 MiB     |
| memory/innodb/dict0dict                                        | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                     | 88.00 MiB     |
+----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

Hasilnya menunjukkan bahwa itu `memory/innodb/memory` adalah konsumen teratas, menggunakan 5,28 GiB memori yang saat ini dialokasikan. Acara ini berfungsi sebagai kategori untuk alokasi memori di berbagai komponen InnoDB yang tidak terkait dengan peristiwa tunggu yang lebih spesifik, `memory/innodb/buf0buf` seperti yang disebutkan sebelumnya.

Setelah menetapkan bahwa komponen InnoDB adalah konsumen utama memori, kita dapat menyelam lebih dalam ke spesifik menggunakan perintah MySQL berikut:

```
SHOW ENGINE INNODB STATUS \G;
```

Perintah SHOW ENGINE INNODB STATUS menyediakan laporan status komprehensif untuk mesin penyimpanan InnoDB, termasuk statistik penggunaan memori terperinci untuk komponen InnoDB yang berbeda. Ini dapat membantu mengidentifikasi struktur atau operasi InnoDB spesifik mana yang paling banyak menghabiskan memori. Untuk informasi selengkapnya, lihat struktur dalam memori InnoDB dalam dokumentasi MySQL.

Menganalisis `BUFFER POOL AND MEMORY` bagian dari laporan status InnoDB, kita melihat bahwa 5.051.647.748 byte (4,7 GiB) dialokasikan ke cache objek kamus, yang menyumbang 89% dari memori yang dilacak oleh. `memory/innodb/memory`

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
```

```
Dictionary memory allocated 5051647748
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

Cache objek kamus adalah cache global bersama yang menyimpan objek kamus data yang diakses sebelumnya dalam memori untuk mengaktifkan penggunaan kembali objek dan meningkatkan kinerja. Alokasi memori yang tinggi ke cache objek kamus menunjukkan sejumlah besar objek database dalam cache kamus data.

Sekarang kita tahu bahwa cache kamus data adalah konsumen utama, kita lanjutkan untuk memeriksa cache kamus data untuk tabel terbuka. Untuk menemukan jumlah tabel dalam cache definisi tabel, kueri variabel status global open_table_definition.

```
mysql> show global status like 'open_table_definitions';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Open_table_definitions | 20000 |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Untuk informasi selengkapnya, lihat Cara MySQL membuka dan menutup tabel dalam dokumentasi MySQL.

Anda dapat membatasi jumlah definisi tabel dalam cache kamus data dengan membatasi `table_definition_cache` parameter di cluster DB atau grup parameter instans DB. Untuk Aurora MySQL, nilai ini berfungsi sebagai batas lunak untuk jumlah tabel dalam cache definisi tabel. Nilai default tergantung pada kelas instance dan diatur sebagai berikut:

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Ketika jumlah tabel melebihi `table_definition_cache` batas, mekanisme yang paling tidak baru digunakan (LRU) mengusir dan menghapus tabel dari cache. Namun, tabel yang terlibat dalam hubungan kunci asing tidak ditempatkan dalam daftar LRU, mencegah penghapusannya.

Dalam skenario kami saat ini, kami menjalankan FLUSH TABLES untuk menghapus cache definisi tabel. Tindakan ini menghasilkan penurunan signifikan dalam variabel status global Open_TABLE_DEFINITIONS, dari 20.000 menjadi 12, seperti yang ditunjukkan di sini:

```
mysql> show global status like 'open_table_definitions';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Open_table_definitions | 12    |
+------------------------+-------+
1 row in set (0.00 sec)
```

Meskipun pengurangan ini, kami mengamati bahwa alokasi memori untuk `memory/innodb/memory` tetap tinggi pada 5,18 GiB, dan memori kamus yang dialokasikan juga tetap tidak berubah. Hal ini terbukti dari hasil query berikut:

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.18 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5001599639
Buffer pool size 170512
```

```
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

Penggunaan memori yang terus-menerus tinggi ini dapat dikaitkan dengan tabel yang terlibat dalam hubungan kunci asing. Tabel ini tidak ditempatkan dalam daftar LRU untuk dihapus, menjelaskan mengapa alokasi memori tetap tinggi bahkan setelah pembilasan cache definisi tabel.

Untuk mengatasi masalah ini:

1. Tinjau dan optimalkan skema database Anda, terutama hubungan kunci asing.
2. Pertimbangkan untuk pindah ke kelas instans DB yang lebih besar yang memiliki lebih banyak memori untuk mengakomodasi objek kamus Anda.

Dengan mengikuti langkah-langkah ini dan memahami pola alokasi memori, Anda dapat mengelola penggunaan memori dengan lebih baik di instans Aurora MySQL DB Anda dan mencegah potensi masalah kinerja karena tekanan memori.

## Memecahkan out-of-memory masalah untuk database Aurora MySQL

Parameter tingkat instans `aurora_oom_response` Aurora MySQL dapat memungkinkan instans DB untuk memantau memori sistem dan memperkirakan memori yang digunakan oleh berbagai pernyataan dan koneksi. Jika sistem kehabisan memori, ia dapat melakukan daftar tindakan untuk mencoba melepaskan memori itu. Ia melakukannya dalam upaya untuk menghindari restart database karena masalah out-of-memory (OOM). Parameter tingkat instance mengambil serangkaian tindakan yang dipisahkan koma yang dilakukan instance DB saat memorinya rendah. `aurora_oom_response`Parameter ini didukung untuk Aurora MySQL versi 2 dan 3.

Nilai-nilai berikut, dan kombinasinya, dapat digunakan untuk `aurora_oom_response` parameter. String kosong berarti tidak ada tindakan yang diambil, dan secara efektif mematikan fitur, membuat database rentan terhadap restart OOM.

- `decline`— Menolak kueri baru ketika instans DB rendah pada memori.
- `kill_connect`Menutup koneksi database yang menghabiskan sejumlah besar memori, dan mengakhiri transaksi saat ini dan pernyataan Data Definition Language (DDL). Respons ini tidak didukung untuk Aurora MySQL versi 2.

Untuk informasi selengkapnya, lihat pernyataan KILL dalam dokumentasi MySQL.

- `kill_query`— Mengakhiri kueri dalam urutan konsumsi memori yang menurun hingga memori instance muncul di atas ambang batas rendah. Pernyataan DDL tidak berakhir.

  Untuk informasi selengkapnya, lihat pernyataan KILL dalam dokumentasi MySQL.
- `print`— Hanya mencetak kueri yang menghabiskan sejumlah besar memori.
- `tune` – Menyesuaikan cache tabel internal untuk melepas sebagian memori kembali ke sistem. Aurora MySQL mengurangi memori yang digunakan untuk cache seperti dan dalam kondisi memori rendah. `table_open_cache` `table_definition_cache` Akhirnya, Aurora MySQL mengatur penggunaan memorinya kembali normal ketika memori sistem tidak lagi rendah.

  Untuk informasi selengkapnya, lihat table_open_cache dan table_definition_cache dalam dokumentasi MySQL.
- `tune_buffer_pool`— Mengurangi ukuran kolam buffer untuk melepaskan beberapa memori dan membuatnya tersedia untuk server database untuk memproses koneksi. Respons ini didukung untuk Aurora MySQL versi 3.06 dan lebih tinggi.

  Anda harus memasangkan `tune_buffer_pool` dengan salah satu `kill_query` atau `kill_connect` dalam nilai `aurora_oom_response` parameter. Jika tidak, pengubahan ukuran kumpulan buffer tidak akan terjadi, bahkan ketika Anda memasukkan `tune_buffer_pool` dalam nilai parameter.

Dalam versi MySQL Aurora lebih rendah dari 3,06, untuk kelas instans DB dengan memori kurang dari atau sama dengan 4 GiB, ketika instance berada di bawah tekanan memori, tindakan default mencakup,,, dan. `print` `tune` `decline` `kill_query` Untuk kelas instance DB dengan memori lebih besar dari 4 GiB, nilai parameter kosong secara default (dinonaktifkan).

Di Aurora MySQL versi 3.06 dan lebih tinggi, untuk kelas instans DB dengan memori kurang dari atau sama dengan 4 GiB, Aurora MySQL juga menutup koneksi yang memakan memori teratas (). `kill_connect` Untuk kelas instance DB dengan memori lebih besar dari 4 GiB, nilai parameter default adalah. `print`

Di Aurora MySQL versi 3.09 dan lebih tinggi, untuk kelas instans DB dengan memori lebih besar dari 4 GiB, nilai parameter default adalah. `print,decline,kill_connect`

Jika Anda sering mengalami out-of-memory masalah, penggunaan memori dapat dipantau menggunakan tabel ringkasan memori saat `performance_schema` diaktifkan.

Untuk CloudWatch metrik Amazon yang terkait dengan OOM, lihatMetrik tingkat instans untuk Amazon Aurora. Untuk variabel status global yang terkait dengan OOM, lihatAurora Variabel status SQL global saya.

# Logging untuk database MySQL Aurora

Log MySQL Aurora memberikan informasi penting tentang aktivitas dan kesalahan database. Dengan mengaktifkan log ini, Anda dapat mengidentifikasi dan memecahkan masalah, memahami kinerja database, dan mengaudit aktivitas database. Kami menyarankan Anda mengaktifkan log ini untuk semua instans Aurora MySQL DB Anda untuk memastikan kinerja dan ketersediaan database yang optimal. Jenis logging berikut dapat diaktifkan. Setiap log berisi informasi spesifik yang dapat menyebabkan mengungkap dampak pada pemrosesan basis data.

- Kesalahan — Aurora MySQL menulis ke log kesalahan hanya pada startup, shutdown, dan ketika menemukan kesalahan. Instan DB dapat memakan waktu berjam-jam atau berhari-hari tanpa perlu menulis entri baru ke log kesalahan. Jika Anda melihat tidak ada entri terbaru, berarti server tidak mengalami kesalahan yang akan mengakibatkan entri log. Pencatatan kesalahan diaktifkan secara default. Untuk informasi selengkapnya, lihat Log kesalahan Aurora MySQL.
- Umum — Log umum memberikan informasi rinci tentang aktivitas database, termasuk semua pernyataan SQL yang dijalankan oleh mesin database. Untuk informasi selengkapnya tentang mengaktifkan logging umum dan pengaturan parameter logging, lihatLog umum dan kueri lambat Aurora MySQL, dan Log kueri umum dalam dokumentasi MySQL.

  > **Note**
  >
  > Log umum dapat tumbuh menjadi sangat besar dan menghabiskan penyimpanan Anda. Untuk informasi selengkapnya, lihat Rotasi dan retensi log untuk Aurora MySQL.

- Kueri lambat - Log kueri lambat terdiri dari pernyataan SQL yang membutuhkan waktu lebih dari long_query_time detik untuk dijalankan dan memerlukan setidaknya baris min_examined_row_limit untuk diperiksa. Anda dapat menggunakan log kueri lambat untuk menemukan kueri yang membutuhkan waktu lama untuk dijalankan dan oleh karena itu kandidat untuk pengoptimalan.

  Nilai default untuk `long_query_time` adalah 10 detik. Kami menyarankan Anda memulai dengan nilai tinggi untuk mengidentifikasi kueri paling lambat, lalu turun untuk fine tuning.

  Anda juga dapat menggunakan parameter terkait, seperti `log_slow_admin_statements` dan`log_queries_not_using_indexes`. Bandingkan `rows_examined` dengan`rows_returned`. Jika `rows_examined` jauh lebih besar dari`rows_returned`, maka kueri tersebut berpotensi memblokir.

Di Aurora MySQL versi 3, Anda dapat mengaktifkan untuk mendapatkan rincian lebih lanjut. `log_slow_extra` Untuk informasi selengkapnya, lihat Konten log kueri lambat dalam dokumentasi MySQL. Anda juga dapat memodifikasi `long_query_time` pada tingkat sesi untuk men-debug eksekusi kueri secara interaktif, yang sangat berguna jika `log_slow_extra` diaktifkan secara global.

Untuk informasi selengkapnya tentang mengaktifkan pencatatan kueri lambat dan pengaturan parameter logging, lihatLog umum dan kueri lambat Aurora MySQL, dan Log kueri lambat dalam dokumentasi MySQL.

- Audit — Log audit memantau dan mencatat aktivitas database. Pencatatan log audit untuk Aurora MySQL disebut Audit Lanjutan. Untuk mengaktifkan Audit Lanjutan, Anda menetapkan parameter cluster DB tertentu. Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.
- Biner — Log biner (binlog) berisi peristiwa yang menggambarkan perubahan database, seperti operasi pembuatan tabel dan perubahan data tabel. Ini juga berisi peristiwa untuk pernyataan yang berpotensi membuat perubahan (misalnya, DELETE yang tidak cocok dengan baris), kecuali logging berbasis baris digunakan. Log biner juga berisi informasi tentang berapa lama setiap pernyataan mengambil data yang diperbarui.

  Menjalankan server dengan logging biner diaktifkan membuat kinerja sedikit lebih lambat. Namun, manfaat log biner yang memungkinkan Anda mengatur replikasi dan untuk operasi pemulihan umumnya lebih besar daripada penurunan kinerja kecil ini.

  > **Note**
  >
  > Aurora MySQL tidak memerlukan pencatatan biner untuk operasi pemulihan.

  Untuk informasi lebih lanjut tentang mengaktifkan logging biner dan pengaturan format binlog, lihatMengkonfigurasi Aurora untuk database Single-AZ, dan Log biner dalam dokumentasi MySQL.

Anda dapat mempublikasikan log kesalahan, umum, lambat, kueri, dan audit ke Amazon CloudWatch Logs. Untuk informasi selengkapnya, lihat Menerbitkan log database ke Amazon CloudWatch Logs.

Alat lain yang berguna untuk meringkas file log lambat, umum, dan biner adalah pt-query-digest.

Memastikan konektivitas yang andal antara aplikasi dan instans RDS DB Anda sangat penting untuk kelancaran pengoperasian beban kerja Anda. Namun, masalah konektivitas dapat muncul karena berbagai faktor, seperti konfigurasi jaringan, masalah otentikasi, atau kendala sumber daya.

Daftar Isi


- 
- 
- 
  - Pemantauan tambahan untuk Aurora My SQL
- 
- 
- 
  - Contoh 1: Memecahkan masalah upaya koneksi yang gagal
  - Contoh 2: Memecahkan masalah klien abnormal terputus


Mengidentifikasi kategori spesifik dari masalah konektivitas dapat membantu mempersempit penyebab potensial dan memandu proses pemecahan masalah. Setiap kategori mungkin memerlukan pendekatan dan teknik yang berbeda untuk diagnosis dan resolusi. Masalah konektivitas database secara luas dapat diklasifikasikan ke dalam kategori berikut.

Kesalahan dan pengecualian koneksi

Kesalahan dan pengecualian koneksi dapat terjadi karena berbagai alasan, seperti string koneksi yang salah, kegagalan otentikasi, gangguan jaringan, atau masalah server database. Penyebabnya dapat mencakup parameter koneksi yang salah konfigurasi, kredensi yang tidak valid, pemadaman jaringan, atau crash atau restart server database. Grup keamanan yang salah konfigurasi, pengaturan cloud pribadi virtual (VPC), Daftar Kontrol Akses jaringan (ACLs), dan tabel rute yang terkait dengan subnet juga dapat menyebabkan masalah koneksi.

Batas koneksi tercapai

Masalah ini muncul ketika jumlah koneksi bersamaan ke server database melebihi batas maksimum yang diizinkan. Server database biasanya memiliki batas koneksi maksimum yang dapat dikonfigurasi yang ditentukan oleh parameter max_connections di cluster dan grup parameter instance. Dengan memberlakukan batas koneksi, server database memastikan bahwa ia memiliki sumber daya yang cukup (misalnya, memoriCPU, dan pegangan file) untuk menangani koneksi yang ada secara efisien dan memberikan kinerja yang dapat diterima. Penyebabnya

dapat mencakup kebocoran koneksi dalam aplikasi, penyatuan koneksi yang tidak efisien, atau lonjakan permintaan koneksi yang tidak terduga.

Batas waktu koneksi

Batas waktu koneksi terjadi ketika aplikasi klien tidak dapat membuat koneksi dengan server database dalam periode batas waktu tertentu. Penyebab umum termasuk masalah jaringan, kelebihan server, aturan firewall, dan pengaturan koneksi yang salah konfigurasi.

Batas waktu koneksi idle

Koneksi idle yang tetap tidak aktif untuk jangka waktu lama mungkin ditutup secara otomatis oleh server database untuk menghemat sumber daya. Batas waktu ini biasanya dapat dikonfigurasi menggunakan `wait_timeout` dan`interactive_timeout parameters`, dan harus disesuaikan berdasarkan pola penggunaan koneksi aplikasi. Penyebabnya dapat mencakup logika aplikasi yang membuat koneksi menganggur untuk waktu yang lama, atau manajemen koneksi yang tidak tepat.

Pemutusan sambungan intermiten dari koneksi yang ada

Kelas kesalahan ini mengacu pada skenario di mana koneksi yang dibuat antara aplikasi klien dan database secara tak terduga dihentikan atau terputus pada interval yang tidak teratur, meskipun aktif dan digunakan. Pemutusan ini terjadi sebentar-sebentar, artinya terjadi pada interval yang tidak teratur dan tidak konsisten. Penyebabnya bisa termasuk yang berikut:

- Masalah server database seperti restart atau failover
- Penanganan koneksi aplikasi yang tidak tepat
- Masalah load balancing dan proxy
- Ketidakstabilan jaringan
- Masalah dengan komponen pihak ketiga atau middleware yang terlibat dalam jalur koneksi
- Batas waktu eksekusi kueri
- Kendala sumber daya di server atau sisi klien

Mengidentifikasi akar penyebab melalui pemantauan, pencatatan, dan analisis yang komprehensif sangat penting, sementara menerapkan penanganan kesalahan yang tepat, pengumpulan koneksi, dan mekanisme coba lagi dapat membantu mengurangi dampak pemutusan intermiten ini pada fungsionalitas aplikasi dan pengalaman pengguna.

Dengan mengumpulkan log, konfigurasi, dan informasi diagnostik yang relevan, Anda mendapatkan wawasan berharga yang dapat membantu mengidentifikasi akar penyebab masalah konektivitas dan memandu Anda menuju resolusi yang tepat.

Log dan konfigurasi jaringan, seperti aturan grup keamanan, VPC pengaturan, dan tabel rute, sangat penting untuk mengidentifikasi potensi kemacetan terkait jaringan atau kesalahan konfigurasi yang dapat mencegah aplikasi membangun koneksi yang berhasil dengan database. Dengan menganalisis komponen jaringan ini, Anda dapat memastikan bahwa port yang diperlukan terbuka, alamat IP diizinkan, dan konfigurasi perutean diatur dengan benar.

Stempel waktu

Catat stempel waktu yang tepat saat masalah konektivitas terjadi. Ini dapat membantu mengidentifikasi pola atau menghubungkan masalah dengan peristiwa atau kegiatan lain.

Log mesin DB

Selain log database umum, tinjau log mesin database (misalnya, log SQL kesalahan saya dan log kueri lambat) untuk informasi atau kesalahan yang relevan yang mungkin terkait dengan masalah konektivitas intermiten. Untuk informasi selengkapnya, lihat Logging untuk database MySQL Aurora.

Log aplikasi klien

Kumpulkan log terperinci dari aplikasi klien yang terhubung ke database. Log aplikasi memberikan visibilitas ke dalam upaya koneksi, kesalahan, dan informasi yang relevan dari perspektif aplikasi, yang dapat mengungkapkan masalah yang terkait dengan string koneksi, kredensional otentikasi, atau penanganan koneksi tingkat aplikasi.

Log database, di sisi lain, menawarkan wawasan tentang kesalahan sisi database, kueri lambat, atau peristiwa yang mungkin berkontribusi pada masalah konektivitas. Untuk informasi selengkapnya, lihat Logging untuk database MySQL Aurora.

Variabel lingkungan klien

Periksa apakah ada variabel lingkungan atau pengaturan konfigurasi di sisi klien yang mungkin memengaruhi koneksi database, seperti pengaturan proxy,SSL/TLSpengaturan, atau variabel lain yang relevan.

Versi pustaka klien

Pastikan bahwa klien menggunakan versi terbaru dari driver database, pustaka, atau kerangka kerja yang digunakan untuk konektivitas database. Versi usang dapat memiliki masalah atau masalah kompatibilitas yang diketahui.

Penangkapan jaringan klien

Lakukan pengambilan jaringan di sisi klien menggunakan alat seperti Wireshark atau `tcpdump` selama masa-masa ketika masalah konektivitas terjadi. Ini dapat membantu mengidentifikasi masalah atau anomali terkait jaringan di sisi klien.

Topologi jaringan klien

Memahami topologi jaringan klien, termasuk firewall, load balancer, atau komponen lain seperti RDS Proxy atau Proxy SQL yang membuat koneksi ke database alih-alih klien langsung membuat koneksi.

Pengaturan sistem operasi klien

Periksa pengaturan sistem operasi klien yang mungkin memengaruhi konektivitas jaringan, seperti aturan firewall, pengaturan adaptor jaringan, dan pengaturan lain yang relevan.

Konfigurasi penyatuan koneksi

Jika Anda menggunakan mekanisme penyatuan koneksi di aplikasi Anda, tinjau pengaturan konfigurasi dan pantau metrik kumpulan (misalnya, koneksi aktif, koneksi idle, dan batas waktu koneksi) untuk memastikan bahwa kumpulan berfungsi dengan benar. Tinjau juga pengaturan pool, seperti ukuran kolam maksimum, ukuran kolam minimum, dan pengaturan validasi koneksi, untuk memastikan bahwa pengaturan tersebut dikonfigurasi dengan benar.

String koneksi

String koneksi biasanya mencakup parameter seperti nama host atau titik akhir, nomor port, nama database, dan kredensi otentikasi. Menganalisis string koneksi dapat membantu mengidentifikasi potensi kesalahan konfigurasi atau pengaturan yang salah yang mungkin menyebabkan masalah konektivitas. Misalnya, nama host atau nomor port yang salah dapat mencegah klien mencapai instance database, sementara kredensi otentikasi yang tidak valid dapat menyebabkan kegagalan otentikasi dan penolakan koneksi. Selain itu, string koneksi dapat mengungkapkan masalah yang terkait dengan pengumpulan koneksi, batas waktu, atau pengaturan khusus koneksi lainnya yang dapat berkontribusi pada masalah konektivitas. Menyediakan string koneksi lengkap yang digunakan oleh aplikasi klien dapat membantu menentukan kesalahan konfigurasi pada klien.

Metrik basis data

Pantau metrik database seperti CPU penggunaan, penggunaan memori, dan disk I/O selama masa-masa ketika masalah konektivitas terjadi. Ini dapat membantu mengidentifikasi apakah instans DB mengalami pertikaian sumber daya atau masalah kinerja.

Versi mesin basis data

Perhatikan versi mesin Aurora My SQL RDS DB. AWS secara teratur merilis pembaruan yang menangani masalah yang diketahui, kerentanan keamanan, dan memperkenalkan peningkatan kinerja. Oleh karena itu, kami sangat menyarankan Anda meningkatkan ke versi terbaru yang tersedia, karena pembaruan ini sering kali menyertakan perbaikan bug dan peningkatan yang secara khusus terkait dengan konektivitas, kinerja, dan stabilitas. Menyediakan informasi versi database, bersama dengan rincian lain yang dikumpulkan, dapat membantu Dukungan dalam mendiagnosis dan menyelesaikan masalah konektivitas secara efektif.

Metrik jaringan

Kumpulkan metrik jaringan seperti latensi, kehilangan paket, dan throughput selama masa-masa ketika masalah konektivitas terjadi. Alat seperti`ping`,`traceroute`, dan alat pemantauan jaringan dapat membantu mengumpulkan data ini.

Sumber dan detail klien

Tentukan alamat IP server aplikasi, penyeimbang beban, atau komponen lain yang memulai koneksi database. Ini bisa berupa alamat IP tunggal atau berbagai alamat IP (CIDRnotasi). Jika sumbernya adalah EC2 instans Amazon, ini juga membantu untuk meninjau jenis instans, Availability Zone, subnet ID, dan grup keamanan yang terkait dengan instance, dan detail antarmuka jaringan seperti alamat IP pribadi dan alamat IP publik.

Dengan menganalisis data yang dikumpulkan secara menyeluruh, Anda dapat mengidentifikasi kesalahan konfigurasi, kendala sumber daya, gangguan jaringan, atau masalah mendasar lainnya yang menyebabkan masalah konektivitas intermiten atau persisten. Informasi ini memungkinkan Anda untuk mengambil tindakan yang ditargetkan, seperti menyesuaikan konfigurasi, menyelesaikan masalah jaringan, atau menangani penanganan koneksi tingkat aplikasi.

Untuk memantau dan memecahkan masalah konektivitas, Anda dapat menggunakan metrik dan fitur berikut.

CloudWatch metrik

- `CPUUtilization`— CPU Penggunaan yang tinggi pada instans DB dapat menyebabkan eksekusi kueri yang lambat, yang dapat mengakibatkan batas waktu atau penolakan koneksi.
- `DatabaseConnections`— Pantau jumlah koneksi aktif ke instans DB. Sejumlah besar koneksi mendekati maksimum yang dikonfigurasi dapat menunjukkan potensi masalah konektivitas atau kelelahan kumpulan koneksi.
- `FreeableMemory`— Memori yang tersedia rendah dapat menyebabkan masalah kinerja dan masalah konektivitas karena kendala sumber daya.
- `NetworkReceiveThroughput`dan `NetworkTransmitThroughput` — Lonjakan atau penurunan throughput jaringan yang tidak biasa dapat mengindikasikan masalah konektivitas atau kemacetan jaringan.

Metrik-metrik Wawasan Performa

Untuk memecahkan masalah konektivitas di Aurora My SQL RDS for My Insights, analisis metrik Database seperti berikut:

- Aborted_clients
- Aborted_connects
- Koneksi
- max_connections
- Threads_connected
- Threads_created
- Threads_running

Metrik ini dapat membantu Anda mengidentifikasi kemacetan koneksi, mendeteksi masalah jaringan atau otentikasi, mengoptimalkan penyatuan koneksi, dan memastikan manajemen thread yang efisien. Untuk informasi selengkapnya, lihat Penghitung Wawasan Performa untuk Aurora MySQL.

Fitur Performance Insights

- Beban Database - Visualisasikan beban database dari waktu ke waktu dan menghubungkannya dengan masalah konektivitas atau penurunan kinerja.
- SQLStatistik — Menganalisis SQL statistik untuk mengidentifikasi kueri yang tidak efisien atau operasi basis data yang mungkin berkontribusi pada masalah konektivitas.

- Kueri Teratas — Mengidentifikasi dan menganalisis kueri yang paling intensif sumber daya, yang dapat membantu mengidentifikasi potensi kemacetan kinerja atau kueri yang berjalan lama yang mungkin menyebabkan masalah konektivitas.

Dengan memantau metrik ini dan memanfaatkan Performance Insights, Anda dapat memperoleh visibilitas ke kinerja instans database, penggunaan sumber daya, dan potensi kemacetan yang mungkin menyebabkan masalah konektivitas. Sebagai contoh:

- Tinggi `DatabaseConnections` mendekati batas maksimum dapat menunjukkan kelelahan kolam koneksi atau penanganan koneksi yang tidak tepat, yang menyebabkan masalah konektivitas.
- Tinggi `CPUUtilization` atau rendah `FreeableMemory` dapat menunjukkan kendala sumber daya, yang dapat menyebabkan eksekusi kueri lambat dan batas waktu koneksi atau penolakan.
- Menganalisis Kueri dan SQLStatistik Teratas dapat membantu mengidentifikasi kueri yang tidak efisien atau intensif sumber daya yang mungkin berkontribusi pada masalah konektivitas.

Selain itu, pemantauan CloudWatch Log dan pengaturan alarm dapat membantu Anda secara proaktif mengidentifikasi dan menanggapi masalah konektivitas sebelum meningkat.

Penting untuk dicatat bahwa meskipun metrik dan alat ini dapat memberikan wawasan yang berharga, metrik tersebut harus digunakan bersama dengan langkah pemecahan masalah lainnya.

Pemantauan tambahan untuk Aurora My SQL

CloudWatch metrik

- `AbortedClients`— Melacak jumlah koneksi klien yang belum ditutup dengan benar.
- `AuroraSlowConnectionHandleCount`— Melacak jumlah operasi penanganan koneksi yang lambat, yang menunjukkan potensi masalah konektivitas atau kemacetan kinerja.
- `AuroraSlowHandshakeCount`— Mengukur jumlah operasi jabat tangan yang lambat, yang juga bisa menjadi indikator masalah konektivitas.
- `ConnectionAttempts`— Mengukur jumlah upaya koneksi yang dilakukan ke instans Aurora My SQL DB.

Variabel status global

`Aurora_external_connection_count`— Menunjukkan jumlah koneksi database ke instans DB, tidak termasuk koneksi RDS layanan yang digunakan untuk pemeriksaan kesehatan database.

Dengan memantau metrik dan variabel status global ini, Anda dapat memperoleh visibilitas ke dalam pola koneksi, kesalahan, dan potensi kemacetan yang mungkin menyebabkan masalah konektivitas dengan instans Amazon Aurora My Anda. SQL

Misalnya, jumlah yang tinggi `AbortedClients` atau `AuroraSlowConnectionHandleCount` dapat menunjukkan masalah konektivitas.

Selain itu, menyiapkan CloudWatch alarm dan notifikasi dapat membantu Anda secara proaktif mengidentifikasi dan menanggapi masalah konektivitas sebelum meningkat dan memengaruhi kinerja aplikasi Anda.

Berikut ini adalah beberapa kesalahan konektivitas umum untuk Aurora My SQL database, bersama dengan kode kesalahan dan penjelasannya.

Kode Kesalahan 1040: Terlalu banyak koneksi

Kesalahan ini terjadi ketika klien mencoba untuk membuat lebih banyak koneksi daripada maksimum yang diizinkan oleh server database. Kemungkinan penyebabnya meliputi:

- Kesalahan konfigurasi penyatuan koneksi - Jika menggunakan mekanisme penyatuan koneksi, pastikan ukuran kolam maksimum tidak diatur terlalu tinggi, dan koneksi dilepaskan dengan benar kembali ke kolam.
- Konfigurasi instans database - Verifikasi pengaturan koneksi maksimum yang diizinkan untuk instance database dan sesuaikan jika perlu dengan menyetel `max_connections` parameter.
- Konkurensi tinggi — Jika beberapa klien atau aplikasi terhubung ke database secara bersamaan, batas koneksi maksimum yang diizinkan dapat tercapai.

Kode Kesalahan 1045: Akses ditolak untuk pengguna '...' @ '...' (menggunakan kata sandi: YES /NO)

Kesalahan ini menunjukkan kegagalan otentikasi ketika mencoba untuk terhubung ke database. Kemungkinan penyebabnya meliputi:

- Kompatibilitas plugin otentikasi - Periksa apakah plugin otentikasi yang digunakan oleh klien kompatibel dengan mekanisme otentikasi server database.
- Nama pengguna atau kata sandi salah — Verifikasi bahwa nama pengguna dan kata sandi yang benar digunakan dalam string koneksi atau mekanisme otentikasi.
- Izin pengguna — Pastikan bahwa pengguna memiliki izin yang diperlukan untuk terhubung ke instance database dari host atau jaringan yang ditentukan.

Kode Kesalahan 1049: Database tidak dikenal '...'

Kesalahan ini menunjukkan bahwa klien mencoba untuk terhubung ke database yang tidak ada di server. Kemungkinan penyebabnya meliputi:

- Database tidak dibuat — Pastikan database yang ditentukan telah dibuat di server database.
- Nama database salah — Periksa ulang nama database yang digunakan dalam string koneksi atau kueri untuk akurasi.
- Izin pengguna — Verifikasi bahwa pengguna memiliki izin yang diperlukan untuk mengakses database yang ditentukan.

Kode Kesalahan 1153: Mendapat paket yang lebih besar dari byte 'max_allowed_packet'

Kesalahan ini terjadi ketika klien mencoba mengirim atau menerima data yang melebihi ukuran paket maksimum yang diizinkan oleh server database. Kemungkinan penyebabnya meliputi:

- Kueri besar atau set hasil - Jika mengeksekusi kueri yang melibatkan sejumlah besar data, batas ukuran paket mungkin terlampaui.
- Pengaturan ukuran paket yang salah konfigurasi - Periksa `max_allowed_packet` pengaturan pada server database dan sesuaikan jika perlu.
- Masalah konfigurasi jaringan — Pastikan bahwa konfigurasi jaringan (misalnya, MTU ukuran) memungkinkan untuk ukuran paket yang diperlukan.

Kode Kesalahan 1226: Pengguna '...' telah melampaui sumber daya 'max_user_connections' (nilai saat ini:...)

Kesalahan ini menunjukkan bahwa pengguna telah melampaui jumlah maksimum koneksi bersamaan yang diizinkan oleh server database. Kemungkinan penyebabnya meliputi::

- Kesalahan konfigurasi penyatuan koneksi - Jika menggunakan mekanisme penyatuan koneksi, pastikan ukuran kolam maksimum tidak disetel terlalu tinggi untuk batas koneksi pengguna.
- Konfigurasi instance database - Verifikasi `max_user_connections` pengaturan untuk instance database dan sesuaikan jika perlu.
- Konkurensi tinggi — Jika beberapa klien atau aplikasi terhubung ke database secara bersamaan menggunakan pengguna yang sama, batas koneksi khusus pengguna dapat tercapai.

Kode Kesalahan 2003: Tidak dapat terhubung ke SQL server Saya di '...' (10061)

Kesalahan ini biasanya terjadi ketika klien tidak dapat membuat koneksi TCP /IP dengan server database. Hal ini dapat disebabkan oleh berbagai masalah, seperti berikut ini:

- Status instans database - Pastikan bahwa instance database dalam `available` keadaan, dan tidak menjalani operasi pemeliharaan atau cadangan apa pun.
- Aturan Firewall — Periksa apakah ada firewall (sistem operasi, jaringan, atau grup keamanan) yang memblokir koneksi pada port yang ditentukan (biasanya 3306 untuk My). SQL
- Nama host atau titik akhir yang salah - Pastikan bahwa nama host atau titik akhir yang digunakan dalam string koneksi sudah benar dan cocok dengan instance database.
- Masalah konektivitas jaringan — Verifikasi bahwa mesin klien dapat mencapai instance database melalui jaringan. Periksa apakah ada pemadaman jaringan, masalah perutean, VPC atau kesalahan konfigurasi subnet.

Kode Kesalahan 2005: Tidak Diketahui Host SQL server saya '...' (11001)

Kesalahan ini terjadi ketika klien tidak dapat menyelesaikan nama host atau titik akhir server database ke alamat IP. Kemungkinan penyebabnya meliputi:

- DNSmasalah resolusi — Verifikasi bahwa mesin klien dapat menyelesaikan nama host dengan benar menggunakanDNS. Periksa DNS pengaturan, DNS cache, dan coba gunakan alamat IP alih-alih nama host.
- Nama host atau titik akhir salah - Periksa ulang nama host atau titik akhir yang digunakan dalam string koneksi untuk akurasi.
- Masalah konfigurasi jaringan — Pastikan bahwa konfigurasi jaringan klien (misalnya, subnetVPC, dan tabel rute) memungkinkan DNS resolusi dan konektivitas ke instance database.

Kode Kesalahan 2026: kesalahan SSL koneksi

Kesalahan ini terjadi ketika ada masalah dengan SSL TLS konfigurasi/atau validasi sertifikat selama upaya koneksi. Kemungkinan penyebabnya meliputi:

- Kedaluwarsa sertifikat — Periksa apakah SSL TLS sertifikat/yang digunakan oleh server telah kedaluwarsa dan perlu diperpanjang.
- Masalah validasi sertifikat — Verifikasi bahwa klien dapat memvalidasi SSL TLS server/sertifikat dengan benar, dan bahwa sertifikat tersebut dipercaya.
- Masalah konfigurasi jaringan — Pastikan konfigurasi jaringan memungkinkan proses SSL/TLS connections and doesn't block or interfere with the SSL/TLS jabat tangan.
- SSL/TLS configuration mismatch – Make sure that the SSL/TLSpengaturan (misalnya, cipher suite dan versi protokol) pada klien dan server kompatibel.

Koneksi maksimum

Menyesuaikan parameter ini dapat membantu mencegah masalah koneksi yang disebabkan oleh mencapai batas koneksi maksimum yang diizinkan. Pastikan bahwa nilai-nilai ini ditetapkan dengan tepat berdasarkan persyaratan konkurensi aplikasi Anda dan kendala sumber daya.

- `max_connections`— Parameter ini menentukan jumlah maksimum koneksi bersamaan yang diizinkan untuk instans DB.
- `max_user_connections`— Parameter ini dapat ditentukan selama pembuatan dan modifikasi pengguna, dan menetapkan jumlah maksimum koneksi bersamaan yang diizinkan untuk akun pengguna tertentu.

Ukuran buffer jaringan

Meningkatkan nilai-nilai ini dapat meningkatkan kinerja jaringan, terutama untuk beban kerja yang melibatkan transfer data besar atau set hasil. Namun, berhati-hatilah karena ukuran buffer yang lebih besar dapat mengkonsumsi lebih banyak memori.

- `net_buffer_length`— Parameter ini menetapkan ukuran awal untuk koneksi klien dan buffer hasil, menyeimbangkan penggunaan memori dengan kinerja kueri.
- `max_allowed_packet`— Parameter ini menentukan ukuran maksimum paket jaringan tunggal yang dapat dikirim atau diterima oleh instans DB.

Kompresi jaringan (sisi klien)

Mengaktifkan kompresi jaringan dapat mengurangi penggunaan bandwidth jaringan, tetapi dapat meningkatkan CPU overhead pada kedua sisi klien dan server.

- `compress`— Parameter ini memungkinkan atau menonaktifkan kompresi jaringan untuk komunikasi klien/server.
- `compress_protocol`— Parameter ini menentukan protokol kompresi yang akan digunakan untuk komunikasi jaringan.

Penyetelan kinerja jaringan

Menyesuaikan batas waktu ini dapat membantu mengelola koneksi idle dan mencegah kehabisan sumber daya, tetapi berhati-hatilah karena nilai yang rendah dapat menyebabkan penghentian koneksi prematur.

- `interactive_timeout`— Parameter ini menentukan jumlah detik server menunggu aktivitas pada koneksi interaktif sebelum menutupnya.

- `wait_timeout`— Parameter ini menentukan jumlah detik server menunggu aktivitas pada koneksi noninteraktif sebelum menutupnya.

Pengaturan batas waktu jaringan

Menyesuaikan batas waktu ini dapat membantu mengatasi masalah yang terkait dengan koneksi yang lambat atau tidak responsif. Tetapi berhati-hatilah untuk tidak mengaturnya terlalu rendah, karena dapat menyebabkan kegagalan koneksi prematur.

- `net_read_timeout`— Parameter ini menentukan jumlah detik untuk menunggu lebih banyak data dari koneksi sebelum mengakhiri operasi baca.
- `net_write_timeout`— Parameter ini menentukan jumlah detik untuk menunggu blok ditulis ke koneksi sebelum mengakhiri operasi tulis.

Contoh 1: Memecahkan masalah upaya koneksi yang gagal

Upaya koneksi dapat gagal karena beberapa alasan, termasuk kegagalan otentikasi, kegagalan SSL TLS /jabat tangan, `max_connections` batas yang dicapai, dan kendala sumber daya pada instans DB.

Anda dapat melacak jumlah koneksi yang gagal baik dari Performance Insights atau dengan menggunakan perintah berikut.

```
mysql> show global status like 'aborted_connects';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Aborted_connects | 7     |
+------------------+-------+
1 row in set (0.00 sec)
```

Jika jumlah `Aborted_connects` meningkat dari waktu ke waktu, maka aplikasi bisa mengalami masalah konektivitas intermiten.

Anda dapat menggunakan Audit Lanjutan Aurora untuk mencatat koneksi dan memutuskan sambungan dari koneksi klien. Anda dapat melakukan ini dengan mengatur parameter berikut di grup parameter cluster DB:

- `server_audit_logging` = `1`
- `server_audit_events` = `CONNECT`

Berikut ini adalah ekstrak dari log audit untuk login yang gagal.

```
1728498527380921,auora-mysql-node1,user_1,172.31.49.222,147189,0,FAILED_CONNECT,,,1045
1728498527380940,auora-mysql-node1,user_1,172.31.49.222,147189,0,DISCONNECT,,,0
```

Di mana:

- `1728498527380921`— Stempel waktu zaman saat login gagal terjadi
- `aurora-mysql-node1`— Instance identifier dari node Aurora SQL My cluster di mana koneksi gagal
- `user_1`— Nama pengguna database yang login gagal
- `172.31.49.222`— Alamat IP pribadi klien dari mana koneksi dibuat
- `147189`— ID koneksi dari login yang gagal
- `FAILED_CONNECT`— Menunjukkan bahwa koneksi gagal.
- `1045`— Kode pengembalian. Nilai bukan nol menunjukkan kesalahan. Dalam hal ini, `1045` sesuai dengan akses ditolak.

Untuk informasi selengkapnya, lihat Kode kesalahan server dan kode kesalahan Klien dalam SQL dokumentasi Saya.

Anda juga dapat memeriksa log SQL kesalahan Aurora My untuk setiap pesan kesalahan terkait, misalnya:

```
2024-10-09T19:26:59.310443Z 220 [Note] [MY-010926] [Server] Access denied for user
 'user_1'@'172.31.49.222' (using password: YES) (sql_authentication.cc:1502)
```

Contoh 2: Memecahkan masalah klien abnormal terputus

Anda dapat melacak jumlah pemutusan klien abnormal baik dari Performance Insights atau dengan menggunakan perintah berikut.

```
mysql> show global status like 'aborted_clients';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| Aborted_clients | 9     |
+-----------------+-------+
1 row in set (0.01 sec)
```

Jika jumlah `Aborted_clients` meningkat dari waktu ke waktu, maka aplikasi tidak menutup koneksi ke database dengan benar. Jika koneksi tidak ditutup dengan benar, itu dapat menyebabkan kebocoran sumber daya dan potensi masalah kinerja. Membiarkan koneksi terbuka secara tidak perlu dapat mengkonsumsi sumber daya sistem, seperti memori dan deskriptor file, yang pada akhirnya dapat menyebabkan aplikasi atau server menjadi tidak responsif atau restart.

Anda dapat menggunakan kueri berikut untuk mengidentifikasi akun yang tidak menutup koneksi dengan benar. Ini mengambil nama akun pengguna, host dari mana pengguna terhubung, jumlah koneksi tidak ditutup, dan persentase koneksi tidak ditutup.

```
SELECT
    ess.user,
    ess.host,
    (a.total_connections - a.current_connections) - ess.count_star AS not_closed,
    (((a.total_connections - a.current_connections) - ess.count_star) * 100) /
 (a.total_connections - a.current_connections) AS pct_not_closed
FROM
    performance_schema.events_statements_summary_by_account_by_event_name AS ess
    JOIN performance_schema.accounts AS a ON (ess.user = a.user AND ess.host = a.host)
WHERE
    ess.event_name = 'statement/com/quit'
    AND (a.total_connections - a.current_connections) > ess.count_star;

+----------+---------------+------------+----------------+
| user     | host          | not_closed | pct_not_closed |
+----------+---------------+------------+----------------+
| user1    | 172.31.49.222 |          1 |        33.3333 |
| user1    | 172.31.93.250 |       1024 |        12.1021 |
| user2    | 172.31.93.250 |         10 |        12.8551 |
+----------+---------------+------------+----------------+
3 rows in set (0.00 sec)
```

Setelah Anda mengidentifikasi akun pengguna dan host dari mana koneksi tidak ditutup, Anda dapat melanjutkan untuk memeriksa kode yang tidak menutup koneksi dengan anggun.

Misalnya, dengan SQL konektor Saya di Python, gunakan `close()` metode objek koneksi untuk menutup koneksi. Berikut adalah contoh fungsi yang membuat koneksi ke database, melakukan query, dan menutup koneksi:

```
import mysql.connector
```

```
def execute_query(query):
    # Establish a connection to the database
    connection = mysql.connector.connect(
        host="your_host",
        user="your_username",
        password="your_password",
        database="your_database"
    )

    try:
        # Create a cursor object
        cursor = connection.cursor()

        # Execute the query
        cursor.execute(query)

        # Fetch and process the results
        results = cursor.fetchall()
        for row in results:
            print(row)

    finally:
        # Close the cursor and connection
        cursor.close()
        connection.close()
```

Dalam contoh ini, `connection.close()` metode ini dipanggil di `finally` blok untuk memastikan bahwa koneksi ditutup, apakah pengecualian terjadi atau tidak.

# Memecahkan masalah kinerja kueri untuk database Aurora MySQL

MySQL menyediakan kontrol pengoptimal kueri melalui variabel sistem yang memengaruhi cara rencana kueri dievaluasi, pengoptimalan yang dapat dialihkan, petunjuk pengoptimal dan indeks, dan model biaya pengoptimal. Titik data ini dapat membantu tidak hanya saat membandingkan lingkungan MySQL yang berbeda, tetapi juga untuk membandingkan rencana eksekusi kueri sebelumnya dengan rencana eksekusi saat ini, dan untuk memahami eksekusi keseluruhan kueri MySQL kapan saja.

Kinerja kueri tergantung pada banyak faktor, termasuk rencana eksekusi, skema tabel dan ukuran, statistik, sumber daya, indeks, dan konfigurasi parameter. Penyetelan kueri memerlukan identifikasi kemacetan dan mengoptimalkan jalur eksekusi.

- Temukan rencana eksekusi untuk kueri dan periksa apakah kueri menggunakan indeks yang sesuai. Anda dapat mengoptimalkan kueri Anda dengan menggunakan `EXPLAIN` dan meninjau detail setiap paket.
- Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0 Community Edition) menggunakan pernyataan. `EXPLAIN ANALYZE` `EXPLAIN ANALYZE`Pernyataan ini adalah alat profil yang menunjukkan di mana MySQL menghabiskan waktu pada kueri Anda dan mengapa. Dengan`EXPLAIN ANALYZE`, Aurora MySQL merencanakan, menyiapkan, dan menjalankan kueri sambil menghitung baris dan mengukur waktu yang dihabiskan di berbagai titik rencana eksekusi. Saat kueri selesai, `EXPLAIN ANALYZE` mencetak rencana dan pengukurannya alih-alih hasil kueri.
- Perbarui statistik skema Anda dengan menggunakan `ANALYZE` pernyataan. Pengoptimal kueri terkadang dapat memilih rencana eksekusi yang buruk karena statistik yang sudah ketinggalan zaman. Hal ini dapat menyebabkan kinerja kueri yang buruk karena perkiraan kardinalitas yang tidak akurat dari tabel dan indeks. `last_update`Kolom tabel innodb_table_stats menunjukkan terakhir kali statistik skema Anda diperbarui, yang merupakan indikator “kebuntuan” yang baik.
- Masalah lain dapat terjadi, seperti kemiringan distribusi data, yang tidak diperhitungkan untuk kardinalitas tabel. Untuk informasi selengkapnya, lihat Memperkirakan kompleksitas TABEL ANALISIS untuk tabel InnoDB dan statistik Histogram di MySQL dalam dokumentasi MySQL.

## Memahami waktu yang dihabiskan oleh pertanyaan

Berikut ini adalah cara untuk menentukan waktu yang dihabiskan oleh kueri:

- Profil
- Skema Kinerja

- Pengoptimal kueri

Profil

Secara default, pembuatan profil dinonaktifkan. Aktifkan pembuatan profil, lalu jalankan kueri lambat dan tinjau profilnya.

```
SET profiling = 1;
Run your query.
SHOW PROFILE;
```

1. Identifikasi tahap di mana waktu paling banyak dihabiskan. Menurut status utas Umum dalam dokumentasi MySQL, membaca dan memproses baris untuk pernyataan seringkali merupakan status `SELECT` yang paling lama berjalan selama masa kueri yang diberikan. Anda dapat menggunakan `EXPLAIN` pernyataan untuk memahami bagaimana MySQL menjalankan kueri ini.
2. Tinjau log kueri lambat `rows_sent` untuk mengevaluasi `rows_examined` dan memastikan bahwa beban kerja serupa di setiap lingkungan. Untuk informasi selengkapnya, lihat Logging untuk database MySQL Aurora.
3. Jalankan perintah berikut untuk tabel yang merupakan bagian dari kueri yang diidentifikasi:

   ```
   SHOW TABLE STATUS\G;
   ```

4. Tangkap output berikut sebelum dan sesudah menjalankan kueri di setiap lingkungan:

   ```
   SHOW GLOBAL STATUS;
   ```

5. Jalankan perintah berikut di setiap lingkungan untuk melihat apakah ada kueri/sesi lain yang memengaruhi kinerja kueri sampel ini.

   ```
   SHOW FULL PROCESSLIST;

   SHOW ENGINE INNODB STATUS\G;
   ```

   Terkadang, ketika sumber daya di server sibuk, itu berdampak pada setiap operasi lain di server, termasuk kueri. Anda juga dapat menangkap informasi secara berkala saat kueri dijalankan atau menyiapkan `cron` pekerjaan untuk menangkap informasi pada interval yang berguna.

Skema Performa

Skema Kinerja memberikan informasi yang berguna tentang kinerja runtime server, sementara memiliki dampak minimal pada kinerja itu. Ini berbeda dari`information_schema`, yang menyediakan informasi skema tentang instance DB. Untuk informasi selengkapnya, lihat Ikhtisar Skema Kinerja untuk Performance Insights di Aurora.

Jejak pengoptimal kueri

Untuk memahami mengapa rencana kueri tertentu dipilih untuk dieksekusi, Anda dapat mengatur `optimizer_trace` untuk mengakses pengoptimal kueri MySQL.

Jalankan jejak pengoptimal untuk menampilkan informasi ekstensif tentang semua jalur yang tersedia untuk pengoptimal dan pilihannya.

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

## Meninjau pengaturan pengoptimal kueri

Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0 Community Edition) memiliki banyak perubahan terkait pengoptimalan dibandingkan dengan Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7 Community Edition). Jika Anda memiliki beberapa nilai kustom untuk`optimizer_switch`, kami sarankan Anda meninjau perbedaan default dan menetapkan `optimizer_switch` nilai yang paling sesuai untuk beban kerja Anda. Kami juga menyarankan Anda menguji opsi yang tersedia untuk Aurora MySQL versi 3 untuk memeriksa kinerja kueri Anda.

> **Note**
>
> Aurora MySQL versi 3 menggunakan nilai default komunitas 20 untuk parameter innodb_stats_persistent_sample_pages.

Anda dapat menggunakan perintah berikut untuk menunjukkan `optimizer_switch` nilai-nilai:

```
SELECT @@optimizer_switch\G;
```

Tabel berikut menunjukkan `optimizer_switch` nilai default untuk Aurora MySQL versi 2 dan 3.

| Pengaturan | Aurora MySQL versi 2 | Aurora MySQL versi 3 |
|---|---|---|
| `batched_key_access` | off | off |
| `block_nested_loop` | on | on |
| `condition_fanout_filter` | on | on |
| `derived_condition_pushdown` | – | on |
| `derived_merge` | on | on |
| `duplicateweedout` | on | on |
| `engine_condition_pushdown` | on | on |
| `firstmatch` | on | on |
| `hash_join` | off | on |
| `hash_join_cost_based` | on | – |
| `hypergraph_optimizer` | – | off |
| `index_condition_pushdown` | on | on |
| `index_merge` | on | on |
| `index_merge_intersection` | on | on |
| `index_merge_sort_union` | on | on |
| `index_merge_union` | on | on |
| `loosescan` | on | on |

| Pengaturan | Aurora MySQL versi 2 | Aurora MySQL versi 3 |
|---|---|---|
| `materialization` | on | on |
| `mrr` | on | on |
| `mrr_cost_based` | on | on |
| `prefer_ordering_index` | on | on |
| `semijoin` | on | on |
| `skip_scan` | – | on |
| `subquery_materialization_cost_based` | on | on |
| `subquery_to_derived` | – | off |
| `use_index_extensions` | on | on |
| `use_invisible_indexes` | – | off |

Untuk informasi selengkapnya, lihat Pengoptimalan yang dapat dialihkan (MySQL 5.7) dan pengoptimalan Switchable (MySQL 8.0) dalam dokumentasi MySQL.

# Amazon Aurora Referensi saya SQL

Referensi ini mencakup informasi tentang SQL parameter Aurora My, variabel status, dan SQL ekstensi umum atau perbedaan dari komunitas My SQL database engine.

Topik


- Parameter konfigurasi Aurora MySQL
- Aurora Variabel status SQL global saya
- Aurora Acara tunggu saya SQL
- Aurora Utas saya menyatakan SQL
- Tingkat isolasi Aurora MySQL
- Aurora Petunjuk saya SQL
- Aurora Referensi prosedur SQL tersimpan saya
- Aurora My SQL —tabel information_schema spesifik


## Parameter konfigurasi Aurora MySQL

Anda mengelola klaster DB Amazon Aurora MySQL Anda dengan cara yang sama seperti Anda mengelola instans DB Amazon RDS lain, dengan menggunakan parameter dalam grup parameter DB. Amazon Aurora berbeda dari mesin DB lainnya karena Anda memiliki klaster DB yang berisi beberapa instans DB. Akibatnya, beberapa parameter yang Anda gunakan untuk mengelola klaster DB Aurora MySQL berlaku untuk seluruh klaster. Parameter lain hanya berlaku untuk instans DB tertentu dalam klaster DB.

Untuk mengelola parameter tingkat klaster, gunakan grup parameter klaster DB. Untuk mengelola parameter tingkat instans, gunakan grup parameter DB. Setiap instans DB di klaster DB Aurora MySQL kompatibel dengan mesin basis data MySQL. Namun, Anda menerapkan beberapa parameter mesin basis data MySQL di tingkat klaster, dan Anda mengelola parameter ini menggunakan grup parameter klaster DB. Anda tidak dapat menemukan parameter tingkat klaster dalam grup parameter DB untuk instans dalam klaster Aurora DB. Daftar parameter tingkat klaster akan muncul nanti dalam topik ini.

Anda dapat mengelola parameter tingkat cluster dan tingkat instans menggunakan, API, AWS CLI atau Amazon AWS Management Console RDS. Anda menggunakan perintah terpisah untuk mengelola parameter tingkat klaster dan parameter tingkat instans. Misalnya, Anda dapat menggunakan perintah modify-db-cluster-parameter-group CLI untuk mengelola parameter tingkat

cluster dalam grup parameter cluster DB. Anda dapat menggunakan perintah modify-db-parameter-groupCLI untuk mengelola parameter tingkat instance dalam grup parameter DB untuk instance DB di cluster DB.

Anda dapat melihat parameter tingkat klaster dan tingkat instans di konsol, atau dengan menggunakan CLI atau API RDS. Misalnya, Anda dapat menggunakan describe-db-cluster-parameters AWS CLI perintah untuk melihat parameter tingkat cluster dalam grup parameter cluster DB. Anda dapat menggunakan perintah describe-db-parametersCLI untuk melihat parameter tingkat instance dalam grup parameter DB untuk instance DB di cluster DB.

> **Note**
>
> Setiap grup parameter default berisi nilai default untuk semua parameter di grup parameter. Jika parameter memiliki "mesin default" untuk nilai ini, lihat dokumentasi MySQL atau PostgreSQL khusus versi untuk nilai default sebenarnya.
> Kecuali jika dinyatakan lain, parameter yang tercantum dalam tabel berikut berlaku untuk Aurora MySQL versi 2 dan 3.

Untuk informasi selengkapnya tentang grup parameter DB, lihat . Untuk aturan dan batasan klaster Aurora Serverless v1, lihat Kelompok parameter untuk Aurora Serverless v1.

Topik

- Parameter tingkat klaster
- Parameter tingkat instans
- Parameter MySQL yang tidak berlaku untuk Aurora MySQL

## Parameter tingkat klaster

Tabel berikut menunjukkan semua parameter yang berlaku untuk seluruh klaster DB Aurora MySQL.

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `aurora_binlog_read_buffer_size` | Ya | Hanya memengaruhi klaster yang menggunakan replikasi biner log (binlog). Untuk informasi tentang |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| | | replikasi binlog, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner). Dihapus dari Aurora MySQL versi 3. |
| `aurora_binlog_replication_max_yield_seconds` | Ya | Hanya memengaruhi klaster yang menggunakan replikasi biner log (binlog). Untuk informasi tentang replikasi binlog, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner). |
| `aurora_binlog_replication_sec_index_parallel_workers` | Ya | Menetapkan jumlah thread paralel yang tersedia untuk menerapkan perubahan indeks sekunder saat mereplikasi transaksi untuk tabel besar dengan lebih dari satu indeks sekunder. Parameter diatur ke `0` (dinonaktifkan) secara default.<br><br>Parameter ini tersedia di Aurora MySQL versi 306 dan lebih tinggi. Untuk informasi selengkapnya, lihat Mengoptimalkan replikasi log biner untuk Aurora MySQL. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| aurora_binlog_use_large_read_buffer | Ya | Hanya memengaruhi klaster yang menggunakan replikasi biner log (binlog). Untuk informasi tentang replikasi binlog, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner). Dihapus dari Aurora MySQL versi 3. |
| aurora_disable_hash_join | Ya | Atur parameter ini ke 0N untuk menonaktifkan pengoptimalan hash join di Aurora MySQL versi 2.09 atau lebih tinggi. Parameter ini tidak didukung untuk versi 3. Untuk informasi selengkapnya, lihat Kueri paralel untuk Amazon Aurora My SQL. |
| aurora_enable_replica_log_compression | Ya | Untuk informasi selengkapnya, lihat Pertimbangan performa untuk replikasi Amazon Aurora MySQL. Tidak berlaku untuk klaster yang merupakan bagian dari basis data global Aurora. Dihapus dari Aurora MySQL versi 3. |
| aurora_enable_repl_bin_log_filtering | Ya | Untuk informasi selengkapnya, lihat Pertimbangan performa untuk replikasi Amazon Aurora MySQL. Tidak berlaku untuk klaster yang merupakan bagian dari basis data global Aurora. Dihapus dari Aurora MySQL versi 3. |
| aurora_enable_staggered_replica_restart | Ya | Pengaturan ini tersedia di Aurora MySQL versi 3, tetapi tidak digunakan. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `aurora_enable_zdr` | Ya | Pengaturan ini diaktifkan secara default di Aurora MySQL 2.10 dan yang lebih tinggi. Untuk informasi selengkapnya, lihat Restart nol-downtime () untuk ZDR Amazon Aurora My SQL. |
| `aurora_enhanced_binlog` | Ya | Tetapkan nilai parameter ini ke 1 untuk mengaktifkan binlog yang ditingkatkan di Aurora MySQL versi 3.03.1 dan yang lebih tinggi. Untuk informasi selengkap nya, lihat Menyiapkan binlog yang disempurnakan untuk Aurora MySQL. |
| `aurora_full_double_precisio n_in_json` | Ya | Tetapkan nilai parameter ini untuk mengaktifkan penguraian nomor floating point dalam dokumen JSON dengan presisi penuh. |
| `aurora_jemalloc_background_ thread` | Ya | Gunakan parameter ini untuk mengaktif kan thread latar belakang untuk melakukan operasi pemeliharaan memori. Nilai yang diizinkan adalah `0` (dinonaktifkan) dan `1` (diaktifkan). Nilai default-nya adalah `0`.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `aurora_jemalloc_dirty_decay_ms` | Ya | Gunakan parameter ini untuk mempertahankan memori yang dibebaskan untuk jangka waktu tertentu (dalam milidetik). Mempertahankan memori memungkinkan penggunaan kembali lebih cepat. Nilai yang diizinkan adalah `0`–`18446744073709551615`. Nilai defaultnya adalah `10000` (10 detik).<br><br>Anda dapat menggunakan penundaan yang lebih pendek untuk membantu menghindari out-of-memory masalah, dengan mengorbankan kinerja yang lebih lambat.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi. |
| `aurora_jemalloc_tcache_enabled` | Ya | Gunakan parameter ini untuk melayani permintaan memori kecil (hingga 32 KiB) dalam cache lokal thread, melewati arena memori. Nilai yang diizinkan adalah `0` (dinonaktifkan) dan `1` (diaktifkan). Nilai default-nya adalah `1`.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi. |
| `aurora_load_from_s3_role` | Ya | Untuk informasi selengkapnya, lihat Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3. Saat ini tidak tersedia di Aurora MySQL versi 3. Gunakan `aws_default_s3_role`. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `aurora_mask_password_hashes_type` | Ya | Pengaturan ini diaktifkan secara default di Aurora MySQL 2.11 dan yang lebih tinggi.<br><br>Gunakan pengaturan ini untuk me-masking hash kata sandi MySQL Aurora di log kueri dan audit yang lambat. Nilai yang diizinkan adalah `0` dan `1` (default) . Saat diatur ke `1`, kata sandi dicatat sebagai `<secret>`. Saat diatur ke `0`, kata sandi dicatat sebagai nilai hash (#). |
| `aurora_select_into_s3_role` | Ya | Untuk informasi selengkapnya, lihat Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3. Saat ini tidak tersedia di Aurora MySQL versi 3. Gunakan `aws_default_s3_role` . |
| `authentication_kerberos_caseins_cmp` | Ya | Mengontrol perbandingan nama pengguna yang tidak peka huruf besar/kecil untuk plugin `authentication_kerberos` . Atur ke `true` untuk perbandingan yang tidak peka huruf besar/kecil. Secara default, perbandingan peka huruf besar/kecil digunakan (`false`). Untuk informasi selengkapnya, lihat Menggunakan otentikasi Kerberos untuk Aurora My SQL.<br><br>Parameter ini tersedia di Aurora MySQL versi 3.03 dan lebih tinggi. |
| `auto_increment_increment` | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `auto_increment_offset` | Ya | Tidak ada |
| `aws_default_lambda_role` | Ya | Untuk informasi selengkapnya, lihat Memanggil fungsi Lambda dari cluster Amazon Aurora My DB SQL. |
| `aws_default_s3_role` | Ya | Digunakan saat menginvokasi pernyataan `LOAD DATA FROM S3`, `LOAD XML FROM S3`, atau `SELECT INTO OUTFILE S3` dari klaster DB Anda.<br><br>Di Aurora MySQL versi 2, peran IAM yang ditentukan dalam parameter ini akan digunakan jika peran IAM tidak ditentukan untuk `aurora_load_from_s3_role` atau `aurora_select_into_s3_role` untuk pernyataan yang sesuai.<br><br>Di Aurora MySQL versi 3, peran IAM yang ditentukan untuk parameter ini selalu digunakan.<br><br>Untuk informasi selengkapnya, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| binlog_backup | Ya | Tetapkan nilai parameter ini ke 0 untuk mengaktifkan binlog yang ditingkatkan di Aurora MySQL versi 3.03.1 dan yang lebih tinggi. Anda dapat menonaktifkan parameter ini hanya ketika Anda menggunakan binlog yang ditingkatkan. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang disempurnakan untuk Aurora MySQL. |
| binlog_checksum | Ya | API AWS CLI dan RDS melaporkan nilai None jika parameter ini tidak disetel. Dalam hal ini, Aurora MySQL menggunakan nilai default mesin, yaitu CRC32. Ini berbeda dari pengaturan eksplisit NONE, yang menonaktifkan checksum. |
| binlog-do-db | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog_format | Ya | Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner). |
| binlog_group_commit_sync_delay | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog_group_commit_sync_no_delay_count | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog-ignore-db | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| binlog_replication_globaldb | Ya | Tetapkan nilai parameter ini ke 0 untuk mengaktifkan binlog yang ditingkatkan di Aurora MySQL versi 3.03.1 dan yang lebih tinggi. Anda dapat menonaktifkan parameter ini hanya ketika Anda menggunakan binlog yang ditingkatkan. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang disempurnakan untuk Aurora MySQL. |
| binlog_row_image | Tidak | Tidak ada |
| binlog_row_metadata | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog_row_value_options | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog_rows_query_log_events | Ya | Tidak ada |
| binlog_transaction_compression | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog_transaction_compression_level_zstd | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| binlog_transaction_dependency_history_size | Ya | Parameter ini menetapkan batas atas jumlah hash baris yang dipertahankan dalam memori dan digunakan untuk mencari transaksi yang terakhir memodifikasi baris tertentu. Setelah jumlah hash ini tercapai, riwayatnya dibersihkan.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3. |
| binlog_transaction_dependency_tracking | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| character-set-client-handshake | Ya | Tidak ada |
| character_set_client | Ya | Tidak ada |
| character_set_connection | Ya | Tidak ada |
| character_set_database | Ya | Set karakter yang digunakan oleh database default. Nilai default-nya adalah utf8mb4. |
| character_set_filesystem | Ya | Tidak ada |
| character_set_results | Ya | Tidak ada |
| character_set_server | Ya | Tidak ada |
| collation_connection | Ya | Tidak ada |
| collation_server | Ya | Tidak ada |
| completion_type | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `default_storage_engine` | Tidak | Klaster Aurora MySQL menggunakan mesin penyimpanan InnoDB untuk semua data Anda. |
| `enforce_gtid_consistency` | Terkadang | Dapat dimodifikasi di Aurora MySQL versi 2 dan lebih tinggi. |
| `event_scheduler` | Ya | Menunjukkan status Penjadwal Peristiwa.<br><br>Dapat dimodifikasi hanya pada tingkat klaster di Aurora MySQL versi 3. |
| `gtid-mode` | Terkadang | Dapat dimodifikasi di Aurora MySQL versi 2 dan lebih tinggi. |
| `information_schema_stats_expiry` | Ya | Jumlah detik setelah server basis data MySQL mengambil data dari mesin penyimpanan dan mengganti data dalam cache. Nilai yang diizinkan adalah `0`–`31536000`.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `init_connect` | Ya | Perintah yang akan dijalankan oleh server untuk setiap klien yang terhubung. Gunakan tanda kutip ganda (“) untuk pengaturan agar menghindari kegagalan koneksi, misalnya:<br>`SET optimizer_switch="hash_join =off"`<br>Di Aurora MySQL versi 3, parameter ini tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTIO N_ADMIN` . Ini termasuk pengguna master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran. |
| `innodb_adaptive_hash_index` | Ya | Anda dapat memodifikasi parameter ini pada tingkat klaster DB di Aurora MySQL versi 2 dan 3.<br>Indeks Hash Adaptif tidak didukung pada instans DB pembaca. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_aurora_instant_alter_column_allowed | Ya | Mengontrol apakah algoritma INSTANT dapat digunakan untuk operasi ALTER COLUMN di tingkat global. Nilai yang diizinkan adalah sebagai berikut:<br>• 0— INSTANT Algoritma tidak diizinkan untuk ALTER COLUMN operasi (OFF). Kembali ke algoritma lain.<br>• 1— INSTANT Algoritma diperbolehkan untuk ALTER COLUMN operasi (ON). Ini adalah nilai default.<br><br>Untuk informasi selengkapnya, lihat Column Operations dalam dokumentasi MySQL.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi. |
| innodb_autoinc_lock_mode | Ya | Tidak ada |
| innodb_checksums | Tidak | Dihapus dari Aurora MySQL versi 3. |
| innodb_cmp_per_index_enabled | Ya | Tidak ada |
| innodb_commit_concurrency | Ya | Tidak ada |
| innodb_data_home_dir | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `innodb_deadlock_detect` | Ya | Opsi ini digunakan untuk menonaktifkan deteksi deadlock di Aurora MySQL versi 2.11 dan lebih tinggi serta versi 3.<br><br>Pada sistem konkurensi tinggi, deteksi deadlock dapat menyebabkan perlambatan ketika banyak thread menunggu kunci yang sama. Baca dokumentasi MySQL untuk informasi selengkapnya tentang parameter ini. |
| `innodb_default_row_format` | Ya | Parameter ini mendefinisikan format baris default untuk tabel InnoDB (termasuk tabel sementara InnoDB yang dibuat pengguna). Ini berlaku untuk Aurora MySQL versi 2 dan 3.<br><br>Nilainya bisa `DYNAMIC`, `COMPACT`, atau `REDUNDANT`. |
| `innodb_file_per_table` | Ya | Parameter ini memengaruhi cara penyimpanan tabel disusun. Untuk informasi selengkapnya, lihat Penskalaan penyimpanan. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_flush_log_at_trx_commit | Ya | Kami sangat menyarankan agar Anda menggunakan nilai default 1.<br><br>Di Aurora MySQL versi 3, sebelum Anda dapat mengatur parameter ini ke nilai selain 1, Anda harus menetapkan nilai innodb_trx_commit_allow_data_loss ke 1.<br><br>Untuk informasi selengkapnya, lihat Mengonfigurasi seberapa sering buffer log di-flush. |
| innodb_ft_max_token_size | Ya | Tidak ada |
| innodb_ft_min_token_size | Ya | Tidak ada |
| innodb_ft_num_word_optimize | Ya | Tidak ada |
| innodb_ft_sort_pll_degree | Ya | Tidak ada |
| innodb_online_alter_log_max_size | Ya | Tidak ada |
| innodb_optimize_fulltext_only | Ya | Tidak ada |
| innodb_page_size | Tidak | Tidak ada |
| innodb_print_all_deadlocks | Ya | Saat diaktifkan, mencatat informasi tentang semua deadlock InnoDB di log kesalahan Aurora MySQL. Untuk informasi selengkapnya, lihat Meminimalkan dan memecahkan masalah Aurora Kebuntuan saya SQL. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_purge_batch_size | Ya | Tidak ada |
| innodb_purge_threads | Ya | Tidak ada |
| innodb_rollback_on_timeout | Ya | Tidak ada |
| innodb_rollback_segments | Ya | Tidak ada |
| innodb_spin_wait_delay | Ya | Tidak ada |
| innodb_strict_mode | Ya | Tidak ada |
| innodb_support_xa | Ya | Dihapus dari Aurora MySQL versi 3. |
| innodb_sync_array_size | Ya | Tidak ada |
| innodb_sync_spin_loops | Ya | Tidak ada |
| innodb_stats_include_delete_marked | Ya | Ketika parameter ini diaktifkan, InnoDB menyertakan catatan yang ditandai hapus saat menghitung statistik pengoptimisasi persisten.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3. |
| innodb_table_locks | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_trx_commit_allow_data_loss | Ya | Di Aurora MySQL versi 3, atur nilai parameter ini ke 1 sehingga Anda dapat mengubah nilai innodb_flush_log_at_trx_commit .<br><br>Nilai default innodb_trx_commit_allow_data_loss adalah 0.<br><br>Untuk informasi selengkapnya, lihat Mengonfigurasi seberapa sering buffer log di-flush. |
| innodb_undo_directory | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| internal_tmp_disk_storage_engine | Ya | Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah INNODB dan MYISAM.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 2. |
| internal_tmp_mem_storage_engine | Ya | Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah MEMORY dan TempTable .<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| key_buffer_size | Ya | Cache kunci untuk tabel MyISAM. Untuk informasi selengkapnya, lihat mutex keycache->cache_lock. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `lc_time_names` | Ya | Tidak ada |
| `log_error_suppression_list` | Ya | Menentukan daftar kode kesalahan yang tidak login di log kesalahan MySQL. Ini memungkinkan Anda untuk mengabaikan kondisi kesalahan nonkritis tertentu untuk membantu menjaga log kesalahan Anda tetap bersih. Untuk informasi selengkap nya, lihat log_error_suppression_list di dokumentasi MySQL.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3.03 dan lebih tinggi. |
| `low_priority_updates` | Ya | Operasi `INSERT`, `UPDATE`, `DELETE`, dan `LOCK TABLE WRITE` menunggu sampai tidak ada operasi `SELECT` yang tertunda. Parameter ini hanya memengaruhi mesin penyimpanan yang hanya menggunakan penguncia n tingkat tabel (MyISAM, MEMORY, MERGE).<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `lower_case_table_names` | Ya (Aurora MySQL versi 2)<br><br>Hanya pada waktu pembuatan klaster (Aurora MySQL versi 3) | Di Aurora MySQL versi 2.10 dan versi 2.x yang lebih tinggi, pastikan untuk mem-boot ulang semua instans pembaca setelah mengubah pengaturan ini dan mem-boot ulang instans penulis. Untuk detailnya, lihat Mem-boot ulang klaster Aurora dengan ketersediaan baca.<br><br>Di Aurora MySQL versi 3, nilai parameter ini diatur secara permanen pada saat klaster dibuat. Jika Anda menggunakan nilai nondefault untuk opsi ini, siapkan grup parameter kustom Aurora MySQL versi 3 Anda sebelum meningkatkan, dan tentukan grup parameter selama operasi pemulihan snapshot yang membuat klaster versi 3.<br><br>Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan di tempat dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan. Untuk informasi selengkapnya tentang metode yang dapat Anda gunakan, lihat Peningkatan versi utama. |
| `master-info-repository` | Ya | Dihapus dari Aurora MySQL versi 3. |
| `master_verify_checksum` | Ya | Aurora MySQL versi 2. Gunakan `source_verify_checksum` di Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| max_delayed_threads | Ya | Mengatur jumlah maksimum thread untuk menangani pernyataan INSERT DELAYED.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| max_error_count | Ya | Jumlah maksimum pesan kesalahan , peringatan, dan catatan yang akan disimpan untuk ditampilkan.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| max_execution_time | Ya | Batas waktu untuk menjalankan pernyataan SELECT, dalam milidetik. Nilainya bisa berkisar 0–18446744073709551615 . Ketika diatur ke0, tidak ada batas waktu.<br><br>Untuk informasi selengkapnya, lihat max_execution_time dalam dokumentasi MySQL. |
| min_examined_row_limit | Ya | Gunakan parameter ini agar kueri yang memeriksa lebih sedikit jumlah baris dari yang ditentukan tidak dicatat. |
| partial_revokes | Tidak | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| preload_buffer_size | Ya | Ukuran buffer yang dialokasikan saat melakukan pra-pemuatan indeks.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `query_cache_type` | Ya | Dihapus dari Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `read_only` | Ya | Ketika parameter ini diaktifkan, server tidak mengizinkan pembaruan kecuali dari yang dilakukan oleh thread replika.<br><br>Untuk Aurora MySQL versi 2, nilai yang valid adalah sebagai berikut:<br><br>• `0` – `OFF`<br>• `1` – `ON`<br>• `{TrueIfReplica}` — `ON` untuk membaca replika. Ini adalah nilai default.<br>• `{TrueIfClusterReplica}` — `ON` untuk klaster replika seperti replika baca lintas wilayah, cluster sekunder dalam database global Aurora, dan penerapan biru/hijau.<br><br>Untuk Aurora MySQL versi 3, nilai yang valid adalah sebagai berikut:<br><br>• `0`—`OFF`. Ini adalah nilai default.<br>• `1` – `ON`<br>• `{TrueIfClusterReplica}` — `ON` untuk klaster replika seperti replika baca lintas wilayah, cluster sekunder dalam database global Aurora, dan penerapan biru/hijau.<br><br>Di Aurora MySQL versi 3, parameter ini tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTION_ADMIN` . Ini termasuk pengguna |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| | | master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran. |
| `relay-log-space-limit` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replica_parallel_type` | Ya | Parameter ini memungkinkan eksekusi paralel pada replika dari semua thread yang tidak di-commit yang sudah ada dalam fase persiapan, tanpa melanggar konsistensi. Parameter ini berlaku untuk Aurora MySQL versi 3.<br><br>Di Aurora MySQL versi 3.03.* dan lebih rendah, nilai default-nya adalah DATABASE. Di Aurora MySQL versi 3.04 dan lebih tinggi, nilai default-nya adalah LOGICAL_CLOCK. |
| `replica_preserve_commit_order` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replica_transaction_retries` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `replica_type_conversions` | Ya | Parameter ini menentukan jenis konversi yang digunakan pada replika. Nilai yang diizinkan adalah: `ALL_LOSSY` , `ALL_NON_LOSSY` , `ALL_SIGNED` , dan `ALL_UNSIGNED` . Untuk informasi selengkapnya, lihat Replication with differing table definitions on source and replica dalam dokumentasi MySQL.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replicate-do-db` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replicate-do-table` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replicate-ignore-db` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replicate-ignore-table` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replicate-wild-do-table` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `replicate-wild-ignore-table` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `require_secure_transport` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 2 dan 3. Untuk informasi selengkapnya, lihat Koneksi TLS ke cluster DB MySQL Aurora. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `rpl_read_size` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `server_audit_cw_upload` | Tidak | Parameter ini telah usang di Aurora MySQL. Gunakan `server_audit_logs_upload` .<br><br>Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch . |
| `server_audit_events` | Ya | Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL. |
| `server_audit_excl_users` | Ya | Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL. |
| `server_audit_incl_users` | Ya | Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL. |
| `server_audit_logging` | Ya | Untuk petunjuk tentang mengungga h log ke Amazon CloudWatch Logs, lihatMenerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch . |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `server_audit_logs_upload` | Ya | Anda dapat memublikasikan log audit ke CloudWatch Log dengan mengaktif kan Audit Lanjutan dan menyetel parameter ini. 1 Default untuk parameter `server_audit_logs_upload` adalah `0`.<br><br>Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch . |
| `server_id` | Tidak | Tidak ada |
| `skip-character-set-client-handshake` | Ya | Tidak ada |
| `skip_name_resolve` | Tidak | Tidak ada |
| `slave-skip-errors` | Ya | Hanya berlaku untuk klaster Aurora MySQL versi 2, dengan kompatibilitas MySQL 5.7. |
| `source_verify_checksum` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `sync_frm` | Ya | Dihapus dari Aurora MySQL versi 3. |
| `thread_cache_size` | Ya | Jumlah thread yang akan di-cache. Parameter ini berlaku untuk Aurora MySQL versi 2 dan 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `time_zone` | Ya | Secara default, zona waktu untuk cluster Aurora DB adalah Universal Time Coordinated (UTC). Anda dapat mengatur zona waktu untuk instans di klaster DB ke zona waktu lokal untuk aplikasi Anda. Untuk informasi selengkapnya, lihat Zona waktu lokal untuk klaster DB Amazon Aurora. |
| `tls_version` | Ya | Untuk informasi selengkapnya, lihat Versi TLS untuk Aurora MySQL. |

## Parameter tingkat instans

Tabel berikut menunjukkan semua parameter yang berlaku untuk instans DB tertentu di klaster DB Aurora MySQL.

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `activate_all_roles_on_login` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `allow-suspicious-udfs` | Tidak | Tidak ada |
| `aurora_disable_hash_join` | Ya | Atur parameter ini ke `0N` untuk menonaktifkan pengoptimalan hash join di Aurora MySQL versi 2.09 atau lebih tinggi. Parameter ini tidak didukung untuk versi 3. Untuk informasi selengkapnya, lihat Kueri paralel untuk Amazon Aurora My SQL. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| aurora_lab_mode | Ya | Untuk informasi selengkapnya, lihat Mode lab Amazon Aurora MySQL. Dihapus dari Aurora MySQL versi 3. |
| aurora_oom_response | Ya | Parameter ini didukung untuk Aurora MySQL versi 2 dan 3. Untuk informasi selengkapnya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL. |
| aurora_parallel_query | Ya | Atur ke ON untuk mengaktifkan kueri paralel di Aurora MySQL versi 2.09 atau lebih tinggi. Parameter aurora_pq lama tidak digunakan dalam versi ini. Untuk informasi selengkapnya, lihat Kueri paralel untuk Amazon Aurora My SQL. |
| aurora_pq | Ya | Atur ke OFF untuk menonaktifkan kueri paralel untuk instans DB tertentu di versi Aurora MySQL sebelum 2.09. Di versi 2.09 atau lebih tinggi, aktifkan dan nonaktifkan kueri paralel dengan aurora_parallel_query . Untuk informasi selengkapnya, lihat Kueri paralel untuk Amazon Aurora My SQL. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `aurora_read_replica_read_committed` | Ya | Mengaktifkan tingkat isolasi `READ COMMITTED` untuk Replika Aurora dan mengubah perilaku isolasi untuk mengurangi lag pembersihan selama kueri yang berjalan lama. Aktifkan pengaturan ini hanya jika Anda memahami perubahan perilaku dan pengaruhnya terhadap hasil kueri. Misalnya, pengaturan ini menggunakan isolasi yang kurang ketat daripada default MySQL. Ketika diaktifkan, kueri yang berjalan lama mungkin mengamati lebih dari satu salinan dari baris yang sama karena Aurora menyusun ulang data tabel saat kueri sedang berjalan. Untuk informasi selengkapnya, lihat Tingkat isolasi Aurora MySQL. |
| `aurora_tmptable_enable_per_table_limit` | Ya | Menentukan apakah parameter `tmp_table_size` mengontrol ukuran maksimum tabel sementara dalam memori yang dibuat oleh mesin penyimpanan `TempTable` di Aurora MySQL versi 3.04 dan lebih tinggi.<br><br>Untuk informasi selengkapnya, lihat Membatasi ukuran tabel sementara internal dalam memori. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| aurora_use_vector_instructions | Ya | Ketika parameter ini diaktifkan, Aurora MySQL menggunakan instruksi pemrosesan vektor yang dioptimalkan yang disediakan oleh modern CPUs untuk meningkatkan kinerja pada beban kerja intensif I/O.<br><br>Pengaturan ini diaktifkan secara default di Aurora MySQL versi 3.05 dan lebih tinggi. |
| autocommit | Ya | Tidak ada |
| automatic_sp_privileges | Ya | Tidak ada |
| back_log | Ya | Tidak ada |
| basedir | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| binlog_cache_size | Ya | Tidak ada |
| binlog_max_flush_queue_time | Ya | Tidak ada |
| binlog_order_commits | Ya | Tidak ada |
| binlog_stmt_cache_size | Ya | Tidak ada |
| binlog_transaction_compression | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| binlog_transaction_compression_level_zstd | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| bulk_insert_buffer_size | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| concurrent_insert | Ya | Tidak ada |
| connect_timeout | Ya | Tidak ada |
| core-file | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| datadir | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| default_authentication_plugin | Tidak | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| default_time_zone | Tidak | Tidak ada |
| default_tmp_storage_engine | Ya | Mesin penyimpanan default untuk tabel sementara. |
| default_week_format | Ya | Tidak ada |
| delay_key_write | Ya | Tidak ada |
| delayed_insert_limit | Ya | Tidak ada |
| delayed_insert_timeout | Ya | Tidak ada |
| delayed_queue_size | Ya | Tidak ada |
| div_precision_increment | Ya | Tidak ada |
| end_markers_in_json | Ya | Tidak ada |
| eq_range_index_dive_limit | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| event_scheduler | Terkadang | Menunjukkan status Penjadwal Peristiwa.<br><br>Dapat dimodifikasi hanya pada tingkat klaster di Aurora MySQL versi 3. |
| explicit_defaults_for_timestamp | Ya | Tidak ada |
| flush | Tidak | Tidak ada |
| flush_time | Ya | Tidak ada |
| ft_boolean_syntax | Tidak | Tidak ada |
| ft_max_word_len | Ya | Tidak ada |
| ft_min_word_len | Ya | Tidak ada |
| ft_query_expansion_limit | Ya | Tidak ada |
| ft_stopword_file | Ya | Tidak ada |
| general_log | Ya | Untuk petunjuk tentang mengunggah log ke CloudWatch Log, lihatMenerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch . |
| general_log_file | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| group_concat_max_len | Ya | Tidak ada |
| host_cache_size | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `init_connect` | Ya | Perintah yang akan dijalankan oleh server untuk setiap klien yang terhubung. Gunakan tanda kutip ganda (“) untuk pengaturan agar menghindari kegagalan koneksi, misalnya:<br>`SET optimizer_switch="hash_join =off"`<br>Di Aurora MySQL versi 3, parameter ini tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTIO N_ADMIN` , termasuk pengguna master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran. |
| `innodb_adaptive_hash_index` | Ya | Anda dapat memodifikasi parameter ini di tingkat instans DB di Aurora MySQL versi 2. Ini hanya dapat dimodifikasi pada tingkat klaster DB di Aurora MySQL versi 3.<br>Indeks Hash Adaptif tidak didukung pada instans DB pembaca. |
| `innodb_adaptive_max_sleep_d elay` | Ya | Memodifikasi parameter ini tidak berdampak karena `innodb_th read_concurrency` selalu 0 untuk Aurora. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_aurora_max_partitions_for_range | Ya | Dalam beberapa kasus saat statistik tetap tidak tersedia, Anda dapat menggunakan parameter ini untuk meningkatkan performa estimasi jumlah baris pada tabel yang dipartisi.<br><br>Anda dapat mengaturnya ke nilai dalam rentang 0–8192, yang menentukan jumlah partisi yang akan diperiksa selama estimasi jumlah baris. Nilai default adalah 0, yang menentukan estimasi menggunakan semua partisi, sesuai dengan perilaku MySQL default.<br><br>Parameter ini tersedia untuk Aurora MySQL versi 3.03.1 dan lebih tinggi. |
| innodb_autoextend_increment | Ya | Tidak ada |
| innodb_buffer_pool_dump_at_shutdown | Tidak | Tidak ada |
| innodb_buffer_pool_dump_now | Tidak | Tidak ada |
| innodb_buffer_pool_filename | Tidak | Tidak ada |
| innodb_buffer_pool_load_abort | Tidak | Tidak ada |
| innodb_buffer_pool_load_at_startup | Tidak | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_buffer_pool_load_now | Tidak | Tidak ada |
| innodb_buffer_pool_size | Ya | Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai DBInstanceClassMemory dalam rumus dihitung, lihat Variabel formula parameter DB. |
| innodb_change_buffer_max_size | Tidak | Aurora MySQL tidak menggunakan buffer perubahan InnoDB sama sekali. |
| innodb_compression_failure_threshold_pct | Ya | Tidak ada |
| innodb_compression_level | Ya | Tidak ada |
| innodb_compression_pad_pct_max | Ya | Tidak ada |
| innodb_concurrency_tickets | Ya | Memodifikasi parameter ini tidak berdampak karena innodb_thread_concurrency selalu 0 untuk Aurora. |
| innodb_deadlock_detect | Ya | Opsi ini digunakan untuk menonaktifkan deteksi deadlock di Aurora MySQL versi 2.11 dan lebih tinggi serta versi 3.<br><br>Pada sistem konkurensi tinggi, deteksi deadlock dapat menyebabkan perlambatan ketika banyak thread menunggu kunci yang sama. Baca dokumentasi MySQL untuk informasi selengkapnya tentang parameter ini. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_file_format | Ya | Dihapus dari Aurora MySQL versi 3. |
| innodb_flushing_avg_loops | Tidak | Tidak ada |
| innodb_force_load_corrupted | Tidak | Tidak ada |
| innodb_ft_aux_table | Ya | Tidak ada |
| innodb_ft_cache_size | Ya | Tidak ada |
| innodb_ft_enable_stopword | Ya | Tidak ada |
| innodb_ft_server_stopword_table | Ya | Tidak ada |
| innodb_ft_user_stopword_table | Ya | Tidak ada |
| innodb_large_prefix | Ya | Dihapus dari Aurora MySQL versi 3. |
| innodb_lock_wait_timeout | Ya | Tidak ada |
| innodb_log_compressed_pages | Tidak | Tidak ada |
| innodb_lru_scan_depth | Ya | Tidak ada |
| innodb_max_purge_lag | Ya | Tidak ada |
| innodb_max_purge_lag_delay | Ya | Tidak ada |
| innodb_monitor_disable | Ya | Tidak ada |
| innodb_monitor_enable | Ya | Tidak ada |
| innodb_monitor_reset | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_monitor_reset_all | Ya | Tidak ada |
| innodb_old_blocks_pct | Ya | Tidak ada |
| innodb_old_blocks_time | Ya | Tidak ada |
| innodb_open_files | Ya | Tidak ada |
| innodb_print_all_deadlocks | Ya | Saat diaktifkan, mencatat informasi tentang semua deadlock InnoDB di log kesalahan Aurora MySQL. Untuk informasi selengkapnya, lihat Meminimalkan dan memecahkan masalah Aurora Kebuntuan saya SQL. |
| innodb_random_read_ahead | Ya | Tidak ada |
| innodb_read_ahead_threshold | Ya | Tidak ada |
| innodb_read_io_threads | Tidak | Tidak ada |
| innodb_read_only | Tidak | Aurora MySQL mengelola instans read-only dan read/write state of DB instances based on the type of cluster. For example, a provisioned cluster has one read/write DB (instance utama) dan instance lainnya di cluster adalah read-only (Aurora Replicas). |
| innodb_replication_delay | Ya | Tidak ada |
| innodb_sort_buffer_size | Ya | Tidak ada |
| innodb_stats_auto_recalc | Ya | Tidak ada |
| innodb_stats_method | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| innodb_stats_on_metadata | Ya | Tidak ada |
| innodb_stats_persistent | Ya | Tidak ada |
| innodb_stats_persistent_sample_pages | Ya | Tidak ada |
| innodb_stats_transient_sample_pages | Ya | Tidak ada |
| innodb_thread_concurrency | Tidak | Tidak ada |
| innodb_thread_sleep_delay | Ya | Memodifikasi parameter ini tidak berdampak karena innodb_thread_concurrency selalu 0 untuk Aurora. |
| interactive_timeout | Ya | Aurora mengevaluasi nilai minimum interactive_timeout dan wait_timeout . Kemudian, layanan ini menggunakan nilai minimum tersebut sebagai batas waktu untuk mengakhiri semua sesi idle, interaktif, dan non-interaktif. |
| internal_tmp_disk_storage_engine | Ya | Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah INNODB dan MYISAM.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 2. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `internal_tmp_mem_storage_engine` | Terkadang | Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan untuk instance DB penulis adalah `MEMORY` dan`TempTable` .<br><br>Untuk instance DB pembaca, parameter ini diatur ke `TempTable` dan tidak dapat dimodifikasi.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| `join_buffer_size` | Ya | Tidak ada |
| `keep_files_on_create` | Ya | Tidak ada |
| `key_buffer_size` | Ya | Cache kunci untuk tabel MyISAM. Untuk informasi selengkapnya, lihat mutex keycache->cache_lock. |
| `key_cache_age_threshold` | Ya | Tidak ada |
| `key_cache_block_size` | Ya | Tidak ada |
| `key_cache_division_limit` | Ya | Tidak ada |
| `local_infile` | Ya | Tidak ada |
| `lock_wait_timeout` | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| log-bin | Tidak | Mengatur binlog_format ke STATEMENT , MIXED, atau ROW akan secara otomatis mengatur log-bin ke ON. Mengatur binlog_format ke OFF akan secara otomatis mengatur log-bin ke OFF. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner). |
| log_bin_trust_function_creators | Ya | Tidak ada |
| log_bin_use_v1_row_events | Ya | Dihapus dari Aurora MySQL versi 3. |
| log_error | Tidak | Tidak ada |
| log_error_suppression_list | Ya | Menentukan daftar kode kesalahan yang tidak login di log kesalahan MySQL. Ini memungkinkan Anda untuk mengabaikan kondisi kesalahan nonkritis tertentu untuk membantu menjaga log kesalahan Anda tetap bersih. Untuk informasi selengkap nya, lihat log_error_suppression_list di dokumentasi MySQL.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3.03 dan lebih tinggi. |
| log_output | Ya | Tidak ada |
| log_queries_not_using_indexes | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| log_slave_updates | Tidak | Aurora MySQL versi 2. Gunakan log_replica_updates di Aurora MySQL versi 3. |
| log_replica_updates | Tidak | Aurora MySQL versi 3 |
| log_throttle_queries_not_using_indexes | Ya | Tidak ada |
| log_warnings | Ya | Dihapus dari Aurora MySQL versi 3. |
| long_query_time | Ya | Tidak ada |
| low_priority_updates | Ya | Operasi INSERT, UPDATE, DELETE, dan LOCK TABLE WRITE menunggu sampai tidak ada operasi SELECT yang tertunda. Parameter ini hanya memengaruhi mesin penyimpanan yang hanya menggunakan penguncian tingkat tabel (MyISAM, MEMORY, MERGE).<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| max_allowed_packet | Ya | Tidak ada |
| max_binlog_cache_size | Ya | Tidak ada |
| max_binlog_size | Tidak | Tidak ada |
| max_binlog_stmt_cache_size | Ya | Tidak ada |
| max_connect_errors | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| max_connections | Ya | Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai DBInstanceClassMemory dalam rumus dihitung, lihat Variabel formula parameter DB. Untuk nilai default yang tergantung pada kelas instans, lihat Koneksi maksimum ke instans DB Aurora MySQL. |
| max_delayed_threads | Ya | Mengatur jumlah maksimum thread untuk menangani pernyataan INSERT DELAYED.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| max_error_count | Ya | Jumlah maksimum pesan kesalahan, peringatan, dan catatan yang akan disimpan untuk ditampilkan.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| max_execution_time | Ya | Batas waktu untuk menjalankan pernyataan SELECT, dalam milidetik. Nilainya bisa berkisar 0–18446744073709551615. Ketika diatur ke0, tidak ada batas waktu.<br><br>Untuk informasi selengkapnya, lihat max_execution_time dalam dokumentasi MySQL. |
| max_heap_table_size | Ya | Tidak ada |
| max_insert_delayed_threads | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| max_join_size | Ya | Tidak ada |
| max_length_for_sort_data | Ya | Dihapus dari Aurora MySQL versi 3. |
| max_prepared_stmt_count | Ya | Tidak ada |
| max_seeks_for_key | Ya | Tidak ada |
| max_sort_length | Ya | Tidak ada |
| max_sp_recursion_depth | Ya | Tidak ada |
| max_tmp_tables | Ya | Dihapus dari Aurora MySQL versi 3. |
| max_user_connections | Ya | Tidak ada |
| max_write_lock_count | Ya | Tidak ada |
| metadata_locks_cache_size | Ya | Dihapus dari Aurora MySQL versi 3. |
| min_examined_row_limit | Ya | Gunakan parameter ini agar kueri yang memeriksa lebih sedikit jumlah baris dari yang ditentukan tidak dicatat.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| myisam_data_pointer_size | Ya | Tidak ada |
| myisam_max_sort_file_size | Ya | Tidak ada |
| myisam_mmap_size | Ya | Tidak ada |
| myisam_sort_buffer_size | Ya | Tidak ada |
| myisam_stats_method | Ya | Tidak ada |
| myisam_use_mmap | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| net_buffer_length | Ya | Tidak ada |
| net_read_timeout | Ya | Tidak ada |
| net_retry_count | Ya | Tidak ada |
| net_write_timeout | Ya | Tidak ada |
| old-style-user-limits | Ya | Tidak ada |
| old_passwords | Ya | Dihapus dari Aurora MySQL versi 3. |
| optimizer_prune_level | Ya | Tidak ada |
| optimizer_search_depth | Ya | Tidak ada |
| optimizer_switch | Ya | Untuk informasi tentang fitur Aurora MySQL yang menggunakan switch ini, lihat Praktik terbaik dengan Amazon Aurora My SQL. |
| optimizer_trace | Ya | Tidak ada |
| optimizer_trace_features | Ya | Tidak ada |
| optimizer_trace_limit | Ya | Tidak ada |
| optimizer_trace_max_mem_size | Ya | Tidak ada |
| optimizer_trace_offset | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `performance-schema-consumer-events-waits-current` | Ya | Jika kolom Sumber untuk parameter `performance_schema` diatur ke`Modified`, skema kinerja menggunakan parameter`performance-schema-consumer-events-waits-current`. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| `performance-schema-instrument` | Ya | Jika kolom Sumber untuk parameter `performance_schema` diatur ke`Modified`, skema kinerja menggunakan parameter`performance-schema-instrument`. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| `performance_schema` | Ya | Jika kolom Sumber disetel ke`Modified`, Performance Insights mengelola Skema Kinerja. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `performance_schema_accounts_size` | Ya | Jika kolom Sumber untuk parameter `performance_schema` diatur ke`Modified`, skema kinerja menggunakan parameter`performance_schema_accounts_size`. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| `performance_schema_consumer_global_instrumentation` | Ya | Jika kolom Sumber untuk parameter `performance_schema` diatur ke`Modified`, skema kinerja menggunakan parameter`performance_schema_consumer_global_instrumentation`. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| `performance_schema_consumer_thread_instrumentation` | Ya | Jika kolom Sumber untuk parameter `performance_schema` diatur ke`Modified`, skema kinerja menggunakan parameter`performance_schema_consumer_thread_instrumentation`. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_consumer_events_stages_current | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_stages_current . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_consumer_events_stages_history | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_stages_history . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_consumer_events_stages_history_long | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_stages_history_long . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_consumer_events_statements_current | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_statements_current. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_consumer_events_statements_history | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_statements_history. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_consumer_events_statements_history_long | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_statements_history_long. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_consumer_events_waits_history | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_waits_history. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_consumer_events_waits_history_long | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_events_waits_history_long. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_consumer_statements_digest | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_consumer_statements_digest. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_digests_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_digests_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_events_stages_history_long_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_stages_history_long_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_events_stages_history_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_stages_history_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_events_statements_history_long_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_statements_history_long_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_events_statements_history_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_statements_history_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_events_transactions_history_long_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_transactions_history_long_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_events_transactions_history_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_transactions_history_size . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_events_waits_history_long_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_waits_history_long_size . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_events_waits_history_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_events_waits_history_size . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_hosts_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_hosts_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_cond_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_cond_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_cond_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_cond_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_digest_length | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_digest_length. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_file_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_file_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_file_handles | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_file_handles. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_file_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_file_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_index_stat | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_index_stat. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_memory_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_memory_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_metadata_locks | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_metadata_locks. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_mutex_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_mutex_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_mutex_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_mutex_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_prepared_statements_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_prepared_statements_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_program_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_program_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_rwlock_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_rwlock_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_rwlock_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_rwlock_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_socket_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_socket_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_socket_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_socket_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_sql_text_length | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_sql_text_length. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_stage_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_stage_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_statement_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_statement_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_statement_stack | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_statement_stack. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_table_handles | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_table_handles. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_table_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_table_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_max_table_lock_stat | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_table_lock_stat. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_thread_classes | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_thread_classes. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_max_thread_instances | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_max_thread_instances. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_session_connect_attrs_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_session_connect_attrs_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_setup_actors_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_setup_actors_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| performance_schema_setup_objects_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_setup_objects_size. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `performance_schema_show_processlist` | Ya | Parameter ini menentukan implementasi `SHOW PROCESSLIST` mana yang akan digunakan:<br>• Implementasi default diiterasi ke seluruh thread aktif dari dalam pengelola thread sambil mempertahankan mutex global. Hal ini dapat menyebabkan performa lambat, terutama pada sistem yang sibuk.<br>• Implementasi `SHOW PROCESSLIST` alternatif didasarkan pada tabel Skema Performa `processlist`. Implementasi ini mengueri data thread aktif dari Skema Performa, bukan pengelola thread, dan tidak memerlukan mutex.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3.<br><br>Jika kolom Sumber untuk parameter `performance_schema` diatur ke`Modified`, skema kinerja menggunakan parameter`performance_schema_show_processlist`. Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| performance_schema_users_size | Ya | Jika kolom Sumber untuk parameter performance_schema diatur keModified, skema kinerja menggunakan parameterperformance_schema_users_size . Untuk informasi selengkapnya tentang mengaktifkan Skema Kinerja, lihat. Menentukan apakah Wawasan Performa mengelola Skema Performa |
| pid_file | Tidak | Tidak ada |
| plugin_dir | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| port | Tidak | Aurora MySQL mengelola properti koneksi dan memberlakukan pengaturan yang konsisten untuk semua instans DB dalam klaster. |
| preload_buffer_size | Ya | Ukuran buffer yang dialokasikan saat melakukan pra-pemuatan indeks.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 3. |
| profiling_history_size | Ya | Tidak ada |
| query_alloc_block_size | Ya | Tidak ada |
| query_cache_limit | Ya | Dihapus dari Aurora MySQL versi 3. |
| query_cache_min_res_unit | Ya | Dihapus dari Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| query_cache_size | Ya | Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai DBInstanceClassMemory dalam rumus dihitung, lihat Variabel formula parameter DB.<br><br>Dihapus dari Aurora MySQL versi 3. |
| query_cache_type | Ya | Dihapus dari Aurora MySQL versi 3. |
| query_cache_wlock_invalidate | Ya | Dihapus dari Aurora MySQL versi 3. |
| query_prealloc_size | Ya | Tidak ada |
| range_alloc_block_size | Ya | Tidak ada |
| read_buffer_size | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `read_only` | Ya | Ketika parameter ini diaktifkan, server tidak mengizinkan pembaruan kecuali dari yang dilakukan oleh thread replika.<br><br>Untuk Aurora MySQL versi 2, nilai yang valid adalah sebagai berikut:<br><br>• `0` – `OFF`<br>• `1` – `ON`<br>• `{TrueIfReplica}` — `ON` untuk membaca replika. Ini adalah nilai default.<br>• `{TrueIfClusterReplica}` — `ON` untuk contoh di cluster replika seperti replika baca lintas wilayah, cluster sekunder dalam database global Aurora, dan penerapan biru/hijau.<br><br>Kami menyarankan Anda menggunakan grup parameter klaster DB di Aurora MySQL versi 2 untuk memastikan bahwa parameter `read_only` diterapkan ke instans penulis baru pada saat failover.<br><br>**Note**<br>Instans pembaca selalu berstatus hanya baca karena Aurora MySQL mengatur `innodb_read_only` ke 1 di semua pembaca. Oleh karena |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| | | itu, `read_only` menjadi redundan di instans pembaca.<br><br>Dihapus pada tingkat instans dari Aurora MySQL versi 3. |
| `read_rnd_buffer_size` | Ya | Tidak ada |
| `relay-log` | Tidak | Tidak ada |
| `relay_log_info_repository` | Ya | Dihapus dari Aurora MySQL versi 3. |
| `relay_log_recovery` | Tidak | Tidak ada |
| `replica_checkpoint_group` | Ya | Aurora MySQL versi 3 |
| `replica_checkpoint_period` | Ya | Aurora MySQL versi 3 |
| `replica_parallel_workers` | Ya | Aurora MySQL versi 3 |
| `replica_pending_jobs_size_max` | Ya | Aurora MySQL versi 3 |
| `replica_skip_errors` | Ya | Aurora MySQL versi 3 |
| `replica_sql_verify_checksum` | Ya | Aurora MySQL versi 3 |
| `safe-user-create` | Ya | Tidak ada |
| `secure_auth` | Ya | Parameter ini selalu diaktifkan di Aurora MySQL versi 2. Percobaan untuk menonaktifkannya akan menghasilkan kesalahan.<br><br>Dihapus dari Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| secure_file_priv | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| show_create_table_verbosity | Ya | Pengaktifan variabel ini menyebabkan SHOW_CREATE_TABLE menampilkan ROW_FORMAT terlepas dari apakah format ini merupakan format default.<br><br>Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3. |
| skip-slave-start | Tidak | Tidak ada |
| skip_external_locking | Tidak | Tidak ada |
| skip_show_database | Ya | Tidak ada |
| slave_checkpoint_group | Ya | Aurora MySQL versi 2. Gunakan replica_checkpoint_group di Aurora MySQL versi 3. |
| slave_checkpoint_period | Ya | Aurora MySQL versi 2. Gunakan replica_checkpoint_period di Aurora MySQL versi 3. |
| slave_parallel_workers | Ya | Aurora MySQL versi 2. Gunakan replica_parallel_workers di Aurora MySQL versi 3. |
| slave_pending_jobs_size_max | Ya | Aurora MySQL versi 2. Gunakan replica_pending_jobs_size_max di Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| slave_sql_verify_checksum | Ya | Aurora MySQL versi 2. Gunakan replica_sql_verify_checksum di Aurora MySQL versi 3. |
| slow_launch_time | Ya | Tidak ada |
| slow_query_log | Ya | Untuk petunjuk tentang mengunggah log ke CloudWatch Log, lihatMenerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch . |
| slow_query_log_file | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| socket | Tidak | Tidak ada |
| sort_buffer_size | Ya | Tidak ada |
| sql_mode | Ya | Tidak ada |
| sql_select_limit | Ya | Tidak ada |
| stored_program_cache | Ya | Tidak ada |
| sync_binlog | Tidak | Tidak ada |
| sync_master_info | Ya | Tidak ada |
| sync_source_info | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. |
| sync_relay_log | Ya | Dihapus dari Aurora MySQL versi 3. |
| sync_relay_log_info | Ya | Tidak ada |
| sysdate-is-now | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `table_cache_element_entry_ttl` | Tidak | Tidak ada |
| `table_definition_cache` | Ya | Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai `DBInstanceClassMemory` dalam rumus dihitung, lihat Variabel formula parameter DB. |
| `table_open_cache` | Ya | Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai `DBInstanceClassMemory` dalam rumus dihitung, lihat Variabel formula parameter DB. |
| `table_open_cache_instances` | Ya | Tidak ada |
| `temp-pool` | Ya | Dihapus dari Aurora MySQL versi 3. |
| `temptable_max_mmap` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. Untuk detailnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3. |
| `temptable_max_ram` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. Untuk detailnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3. |
| `temptable_use_mmap` | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. Untuk detailnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3. |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| thread_cache_size | Ya | Jumlah thread yang akan di-cache. Parameter ini berlaku untuk Aurora MySQL versi 2 dan 3. |
| thread_handling | Tidak | Tidak ada |
| thread_stack | Ya | Tidak ada |
| timed_mutexes | Ya | Tidak ada |
| tmp_table_size | Ya | Mendefinisikan ukuran maksimum tabel sementara dalam memori internal yang dibuat oleh mesin penyimpanan MEMORY di Aurora MySQL versi 3.<br><br>Di Aurora MySQL versi 3.04 dan yang lebih tinggi, mendefinisikan ukuran maksimum tabel sementara dalam memori internal yang dibuat oleh mesin penyimpanan TempTable saat aurora_tmptable_enable_per_table_limit adalah ON.<br><br>Untuk informasi selengkapnya, lihat Membatasi ukuran tabel sementara internal dalam memori. |
| tmpdir | Tidak | Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung. |
| transaction_alloc_block_size | Ya | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| transaction_isolation | Ya | Parameter ini berlaku untuk Aurora MySQL versi 3. Parameter ini menggantikan tx_isolation . |
| transaction_prealloc_size | Ya | Tidak ada |
| tx_isolation | Ya | Dihapus dari Aurora MySQL versi 3. Parameter ini digantikan oleh transaction_isolation . |
| updatable_views_with_limit | Ya | Tidak ada |
| validate-password | Tidak | Tidak ada |
| validate_password_dictionary_file | Tidak | Tidak ada |
| validate_password_length | Tidak | Tidak ada |
| validate_password_mixed_case_count | Tidak | Tidak ada |
| validate_password_number_count | Tidak | Tidak ada |
| validate_password_policy | Tidak | Tidak ada |
| validate_password_special_char_count | Tidak | Tidak ada |

| Nama parameter | Dapat diubah | Catatan |
| --- | --- | --- |
| `wait_timeout` | Ya | Aurora mengevaluasi nilai minimum `interactive_timeout` dan `wait_timeout` . Kemudian, layanan ini menggunakan nilai minimum tersebut sebagai batas waktu untuk mengakhiri semua sesi idle, interaktif, dan non-inter aktif. |

## Parameter MySQL yang tidak berlaku untuk Aurora MySQL

Karena perbedaan arsitektur antara Aurora MySQL dan MySQL, sebagian parameter MySQL tidak berlaku untuk Aurora MySQL.

Parameter MySQL berikut tidak berlaku untuk Aurora MySQL: Daftar ini tidak lengkap.

- `activate_all_roles_on_login` – Parameter ini tidak berlaku untuk Aurora MySQL versi 2. Parameter ini tersedia di Aurora MySQL versi 3.
- `big_tables`
- `bind_address`
- `character_sets_dir`
- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_buffer_pool_chunk_size`
- `innodb_buffer_pool_instances`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_data_file_path`
- `innodb_dedicated_server`
- `innodb_doublewrite`
- `innodb_flush_log_at_timeout` – Parameter ini tidak berlaku untuk Aurora MySQL. Lihat informasi yang lebih lengkap di Mengonfigurasi seberapa sering buffer log di-flush.

- innodb_flush_method
- innodb_flush_neighbors
- innodb_io_capacity
- innodb_io_capacity_max
- innodb_log_buffer_size
- innodb_log_file_size
- innodb_log_files_in_group
- innodb_log_spin_cpu_abs_lwm
- innodb_log_spin_cpu_pct_hwm
- innodb_log_writer_threads
- innodb_max_dirty_pages_pct
- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

## Aurora Variabel status SQL global saya

Aurora My SQL mencakup variabel status dari komunitas My SQL dan variabel yang unik untuk Aurora. Anda dapat memeriksa variabel-variabel ini untuk mempelajari tentang apa yang terjadi di dalam mesin database. Untuk informasi selengkapnya tentang variabel status di komunitas SayaSQL, lihat Variabel Status Server di komunitas dokumentasi SQL 8.0 Saya.

Anda dapat menemukan nilai saat ini untuk variabel status SQL global Aurora My dengan menggunakan pernyataan seperti berikut:

```
show global status like '%aurora%';
```

> **Note**
>
> Variabel status global dihapus saat mesin DB reboot.

Tabel berikut menjelaskan variabel status global yang digunakan Aurora MySQL.

| Nama | Penjelasan |
| --- | --- |
| `AuroraDb_commits` | Jumlah total commit sejak pengaktifan ulang terakhir. |
| `AuroraDb_commit_latency` | Latensi commit agregat sejak pengaktifan ulang terakhir. |
| `AuroraDb_ddl_stmt_duration` | DDLLatensi agregat sejak restart terakhir. |
| `AuroraDb_select_stmt_duration` | Latensi pernyataan `SELECT` agregat sejak pengaktifan ulang terakhir. |
| `AuroraDb_insert_stmt_duration` | Latensi pernyataan `INSERT` agregat sejak pengaktifan ulang terakhir. |
| `AuroraDb_update_stmt_duration` | Latensi pernyataan `UPDATE` agregat sejak pengaktifan ulang terakhir. |
| `AuroraDb_delete_stmt_duration` | Latensi pernyataan `DELETE` agregat sejak pengaktifan ulang terakhir. |
| `Aurora_binlog_io_cache_allocated` | Jumlah byte yang dialokasikan ke binlog I/O cache. |
| `Aurora_binlog_io_cache_read _requests` | Jumlah permintaan baca yang dibuat ke cache I/O binlog. |
| `Aurora_binlog_io_cache_reads` | Jumlah permintaan baca yang dilayani dari cache binlog I/O. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_enhanced_binlog` | Menunjukkan apakah binlog yang ditingkat kan diaktifkan atau dinonaktifkan untuk instans DB ini. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang disempurnakan untuk Aurora MySQL. |
| `Aurora_external_connection_count` | Jumlah koneksi database ke instans DB, tidak termasuk koneksi RDS layanan yang digunakan untuk pemeriksaan kesehatan database. |
| `Aurora_fast_insert_cache_hits` | Penghitung yang bertambah saat kursor yang di-cache berhasil diambil dan diverifikasi. Untuk informasi selengkapnya tentang cache penyisipan cepat, lihat Peningkatan performa Amazon Aurora MySQL. |
| `Aurora_fast_insert_cache_misses` | Penghitung yang bertambah saat kursor yang di-cache tidak lagi valid dan Aurora melakukan penjelajahan indeks normal. Untuk informasi selengkapnya tentang cache penyisipan cepat, lihat Peningkatan performa Amazon Aurora MySQL. |
| `Aurora_fts_cache_memory_used` | Jumlah memori dalam byte yang digunakan sistem pencarian teks lengkap InnoDB. Variabel ini berlaku untuk Aurora My SQL versi 3.07 dan lebih tinggi. |
| `Aurora_fwd_master_dml_stmt_count` | Total jumlah pernyataan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 2. |
| `Aurora_fwd_master_dml_stmt_duration` | Total durasi pernyataan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 2. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_fwd_master_errors_rpc_timeout` | Berapa kali koneksi yang diteruskan gagal dibuat pada penulis. |
| `Aurora_fwd_master_errors_session_limit` | Jumlah kueri yang diteruskan yang ditolak karena `session full` pada penulis. |
| `Aurora_fwd_master_errors_session_timeout` | Berapa kali sesi penerusan berakhir karena batas waktu pada penulis. |
| `Aurora_fwd_master_open_sessions` | Jumlah sesi yang diteruskan pada instans DB penulis. Variabel ini berlaku untuk Aurora My SQL versi 2. |
| `Aurora_fwd_master_select_stmt_count` | Total jumlah pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 2. |
| `Aurora_fwd_master_select_stmt_duration` | Total durasi pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 2. |
| `Aurora_fwd_writer_dml_stmt_count` | Total jumlah pernyataan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 3. |
| `Aurora_fwd_writer_dml_stmt_duration` | Total durasi pernyataan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 3. |
| `Aurora_fwd_writer_errors_rpc_timeout` | Berapa kali koneksi yang diteruskan gagal dibuat pada penulis. |
| `Aurora_fwd_writer_errors_session_limit` | Jumlah kueri yang diteruskan yang ditolak karena `session full` pada penulis. |
| `Aurora_fwd_writer_errors_session_timeout` | Berapa kali sesi penerusan berakhir karena batas waktu pada penulis. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_fwd_writer_open_sessions` | Jumlah sesi yang diteruskan pada instans DB penulis. Variabel ini berlaku untuk Aurora My SQL versi 3. |
| `Aurora_fwd_writer_select_stmt_count` | Total jumlah pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 3. |
| `Aurora_fwd_writer_select_stmt_duration` | Total durasi pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora My SQL versi 3. |
| `Aurora_lockmgr_buffer_pool_memory_used` | Jumlah memori kumpulan buffer dalam byte yang digunakan oleh Aurora My SQL lock manager. |
| `Aurora_lockmgr_memory_used` | Jumlah memori dalam byte yang digunakan oleh Aurora SQL My lock manager. |
| `Aurora_ml_actual_request_cnt` | Permintaan agregat menghitung bahwa Aurora ySQLmakes M ke layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |
| `Aurora_ml_actual_response_cnt` | Jumlah respons agregat yang diterima Aurora SQL My dari layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_ml_cache_hit_cnt` | Jumlah cache internal agregat yang diterima Aurora SQL My dari layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |
| `Aurora_ml_logical_request_cnt` | Jumlah permintaan logis yang telah dievaluasi instans DB untuk dikirim ke layanan machine learning Aurora sejak reset status terakhir. Tergantung pada apakah batching telah digunakan, nilai ini dapat lebih tinggi dari `Aurora_ml_actual_request_cnt` . Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |
| `Aurora_ml_logical_response_cnt` | Jumlah respons agregat yang diterima Aurora SQL My dari layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |
| `Aurora_ml_retry_request_cnt` | Jumlah permintaan yang dicoba ulang yang dikirim instans DB ke layanan machine learning Aurora sejak reset status terakhir. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_ml_single_request_cnt` | Jumlah agregat fungsi machine learning Aurora yang dievaluasi oleh mode non-batch di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL. |
| `aurora_oom_avoidance_recovery_state` | Menunjukkan apakah pemulihan penghindaran Aurora out-of-memory (OOM) berada dalam `INACTIVE` status `ACTIVE` atau untuk instans DB ini.<br><br>Variabel ini berlaku untuk Aurora My SQL versi 3.06.0 dan lebih tinggi. |
| `aurora_oom_reserved_mem_enter_kb` | Merupakan ambang batas untuk memasuki `RESERVED` negara bagian dalam mekanisme OOM penanganan Aurora.<br><br>Ketika memori yang tersedia di server jatuh di bawah ambang batas ini`RESERVED`, `aurora_oom_status` perubahan ke, menunjukkan bahwa server mendekati tingkat kritis penggunaan memori.<br><br>Variabel ini berlaku untuk Aurora My SQL versi 3.06.0 dan lebih tinggi. |

| Nama | Penjelasan |
| --- | --- |
| `aurora_oom_reserved_mem_exit_kb` | Merupakan ambang batas untuk keluar dari `RESERVED` negara bagian dalam mekanisme OOM penanganan Aurora.<br><br>Ketika memori yang tersedia di server naik di atas ambang batas ini, `aurora_oom_status` kembali ke`NORMAL`, menunjukkan bahwa server telah kembali ke keadaan yang lebih stabil dengan sumber daya memori yang cukup.<br><br>Variabel ini berlaku untuk Aurora My SQL versi 3.06.0 dan lebih tinggi. |
| `aurora_oom_status` | Merupakan OOM status saat ini dari instans DB ini. Ketika nilainya`NORMAL`, ini menunjukkan bahwa ada sumber daya memori yang cukup.<br><br>Jika nilainya berubah`RESERVED`, ini menunjukkan bahwa server memiliki memori yang tersedia rendah. Tindakan diambil berdasarkan konfigurasi `aurora_oom_response` parameter.<br><br>Untuk informasi selengkapnya, lihat Memecahkan out-of-memory masalah untuk database Aurora MySQL.<br><br>Variabel ini berlaku untuk Aurora My SQL versi 3.06.0 dan lebih tinggi. |
| `Aurora_pq_bytes_returned` | Jumlah byte untuk struktur data tuple yang ditransmisikan ke simpul head selama kueri paralel. Bagi dengan 16.384 untuk membandingkan dengan `Aurora_pq_pages_pushed_down` . |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_max_concurrent_requests` | Jumlah maksimum sesi kueri paralel yang dapat berjalan secara konkuren di instans Aurora DB ini. Ini adalah nomor tetap yang tergantung pada kelas instans AWS DB. |
| `Aurora_pq_pages_pushed_down` | Jumlah halaman data (masing-masing dengan ukuran tetap 16 KiB) tempat kueri paralel menghindari transmisi jaringan ke simpul head. |
| `Aurora_pq_request_attempted` | Jumlah sesi kueri paralel yang diminta. Nilai ini mungkin mewakili lebih dari satu sesi per kueri, tergantung pada SQL konstruksi seperti subkueri dan gabungan. |
| `Aurora_pq_request_executed` | Jumlah sesi kueri paralel yang berhasil berjalan. |
| `Aurora_pq_request_failed` | Jumlah sesi kueri paralel yang menampilkan kesalahan kepada klien. Dalam beberapa kasus, permintaan untuk kueri paralel mungkin gagal, misalnya karena masalah dalam lapisan penyimpanan. Dalam kasus ini, bagian kueri yang gagal dicoba kembali menggunakan mekanisme kueri nonparalel. Jika kueri yang dicoba kembali juga gagal, kesalahan ditampilkan ke klien dan penghitung ini ditambah. |
| `Aurora_pq_request_in_progress` | Jumlah sesi kueri paralel yang sedang berlangsung. Angka ini berlaku untuk instans Aurora DB tertentu yang Anda hubungkan, bukan seluruh klaster Aurora DB. Untuk melihat apakah instans DB mendekati batas konkurensinya, bandingkan nilai ini dengan `Aurora_pq_max_concurrent_requests` . |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen` | Berapa kali kueri paralel tidak dipilih untuk memenuhi suatu kueri. Nilai ini adalah jumlah dari beberapa penghitung granular lainnya. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-ben ar dilakukan. |
| `Aurora_pq_request_not_chose n_below_min_rows` | Berapa kali kueri paralel tidak dipilih karena jumlah baris dalam tabel. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |
| `Aurora_pq_request_not_chose n_column_bit` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena jenis data yang tidak didukung dalam daftar kolom yang diproyeks ikan. |
| `Aurora_pq_request_not_chose n_column_geometry` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `GEOMETRY`. Untuk informasi tentang Aurora SQL versi Saya yang menghapus batasan ini, lihat. Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL |
| `Aurora_pq_request_not_chose n_column_lob` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `LOB`, atau kolom `VARCHAR` yang disimpan secara eksternal karena panjang yang dinyatakan. Untuk informasi tentang Aurora SQL versi Saya yang menghapus batasan ini, lihat. Memutakhirkan cluster kueri paralel ke Aurora Versi saya 3 SQL |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_column_virtual` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel berisi kolom virtual. |
| `Aurora_pq_request_not_chosen_custom_charset` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan set karakter kustom. |
| `Aurora_pq_request_not_chosen_fast_ddl` | Jumlah permintaan query paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel saat ini sedang diubah oleh pernyataan cepat DDL`ALTER`. |
| `Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool` | Berapa kali kueri paralel tidak dipilih, meskipun kurang dari 95 persen data tabel berada di dalam pool buffer, karena data tabel yang tidak di-buffer tidak cukup untuk membuat kueri paralel layak dijalankan. |
| `Aurora_pq_request_not_chosen_full_text_index` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki indeks teks penuh. |
| `Aurora_pq_request_not_chosen_high_buffer_pool_pct` | Berapa kali kueri paralel tidak dipilih karena persentase tinggi data tabel (saat ini, lebih dari 95 persen) sudah berada di dalam pool buffer. Dalam kasus ini, pengoptimal menentukan bahwa membaca data dari pool buffer akan lebih efisien. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_index_hint` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mencakup petunjuk indeks. |
| `Aurora_pq_request_not_chosen_innodb_table_format` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel menggunakan format baris InnoDB yang tidak didukung. Kueri paralel Aurora hanya berlaku untuk format baris `COMPACT`, `REDUNDANT` , dan `DYNAMIC`. |
| `Aurora_pq_request_not_chosen_long_trx` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri dimulai di dalam transaksi yang berjalan lama. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |
| `Aurora_pq_request_not_chosen_no_where_clause` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri tidak mencakup klausa `WHERE` apa pun. |
| `Aurora_pq_request_not_chosen_range_scan` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan pemindaian rentang pada indeks. |
| `Aurora_pq_request_not_chosen_row_length_too_long` | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena total panjang gabungan semua kolom terlalu panjang. |

| Nama | Penjelasan |
| --- | --- |
| Aurora_pq_request_not_chosen_small_table | Berapa kali kueri paralel tidak dipilih karena ukuran keseluruhan tabel, sebagaimana ditentukan oleh jumlah baris dan rata-rata panjang baris. Pernyataan EXPLAIN dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan. |
| Aurora_pq_request_not_chosen_temporary_table | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mengacu pada tabel sementara yang menggunakan jenis tabel MyISAM atau memory yang tidak didukung. |
| Aurora_pq_request_not_chosen_tx_isolation | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan tingkat isolasi transaksi yang tidak didukung. Pada instans pembaca DB, kueri paralel hanya berlaku untuk tingkat isolasi REPEATABLE READ dan READ COMMITTED . |
| Aurora_pq_request_not_chosen_update_delete_stmts | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri adalah bagian dari pernyataan UPDATE atau DELETE. |
| Aurora_pq_request_not_chosen_unsupported_access | Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klausa WHERE tidak memenuhi kriteria untuk kueri paralel. Hasil ini dapat muncul jika kueri tidak memerlukan pemindaian sarat data, atau jika kueri adalah pernyataan DELETE atau UPDATE. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_pq_request_not_chosen_unsupported_storage_type` | Jumlah permintaan query paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klaster Aurora SQL My DB tidak menggunakan konfigurasi penyimpanan klaster Aurora yang didukung. Untuk informasi selengkapnya, lihat Pembatasan.<br><br>Parameter ini berlaku untuk Aurora My SQL versi 3.04 dan lebih tinggi. |
| `Aurora_pq_request_throttled` | Berapa kali kueri paralel tidak dipilih karena jumlah maksimum kueri paralel konkuren sudah berjalan pada instans Aurora DB tertentu. |
| `Aurora_repl_bytes_received` | Jumlah byte direplikasi ke instance database Aurora My SQL reader sejak restart terakhir. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora MySQL. |
| `Aurora_reserved_mem_exceeded_incidents` | Berapa kali sejak pengaktifan ulang terakhir bahwa mesin telah melampaui batas memori yang digunakan sistem. Jika `aurora_oom_response` dikonfigurasi, ambang batas ini menentukan kapan out-of-memory (OOM) aktivitas penghindaran dipicu. Untuk informasi lebih lanjut tentang Aurora SQL OOM Tanggapan saya, lihat. Memecahkan out-of-memory masalah untuk database Aurora MySQL |
| `aurora_temptable_max_ram_allocation` | Jumlah maksimum memori, dalam byte, digunakan pada setiap titik oleh tabel sementara internal sejak restart terakhir. |
| `aurora_temptable_ram_allocation` | Jumlah memori saat ini, dalam byte, digunakan oleh tabel sementara internal. |

| Nama | Penjelasan |
| --- | --- |
| `Aurora_thread_pool_thread_count` | Jumlah thread saat ini di pool thread Aurora. Untuk informasi lebih lanjut tentang kumpulan utas di Aurora MySQL, lihat. Pool thread |
| `Aurora_tmz_version` | Menunjukkan versi informasi zona waktu saat ini yang digunakan oleh klaster DB. Nilai mengikuti format Internet Assigned Numbers Authority (IANA):YYYY`suffix` , misalnya `2022a` dan`2023c`.<br><br>Parameter ini berlaku untuk Aurora My SQL versi 2.12 dan lebih tinggi, dan versi 3.04 dan lebih tinggi. |
| `Aurora_zdr_oom_threshold` | Merupakan ambang memori, dalam kilobyte (KB), untuk instance Aurora DB untuk memulai restart waktu henti nol (ZDR) untuk memulihkan dari potensi masalah terkait memori. |
| `server_aurora_das_running` | Menunjukkan apakah Database Activity Streams (DAS) diaktifkan atau dinonakti fkan pada instance DB ini. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data. |

## Variabel SQL status saya yang tidak berlaku untuk Aurora My SQL

Karena perbedaan arsitektur antara Aurora My SQL dan MySQL, beberapa variabel SQL status Saya tidak berlaku untuk Aurora My. SQL

Variabel SQL status Saya berikut tidak berlaku untuk Aurora My. SQL Daftar ini tidak lengkap.

- `innodb_buffer_pool_bytes_dirty`
- `innodb_buffer_pool_pages_dirty`
- `innodb_buffer_pool_pages_flushed`

Aurora My SQL version 3 menghapus variabel status berikut yang ada di Aurora My version 2: SQL

- AuroraDb_lockmgr_bitmaps0_in_use
- AuroraDb_lockmgr_bitmaps1_in_use
- AuroraDb_lockmgr_bitmaps_mem_used
- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated
- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period
- Slave_last_heartbeat
- Slave_received_heartbeats
- Slave_retried_transactions
- Slave_running
- Time_since_zero_connections

Variabel SQL status Saya ini tersedia di Aurora My SQL versi 2, tetapi tidak tersedia di Aurora My versi 3: SQL

- `Innodb_redo_log_enabled`
- `Innodb_undo_tablespaces_total`
- `Innodb_undo_tablespaces_implicit`
- `Innodb_undo_tablespaces_explicit`
- `Innodb_undo_tablespaces_active`

## Aurora Acara tunggu saya SQL

Berikut ini adalah beberapa acara menunggu umum untuk Aurora My. SQL

> **Note**
>
> Untuk informasi tentang menyetel Aurora Performa SQL saya menggunakan acara tunggu, lihat. Menyesuaikan Aurora MySQL dengan peristiwa tunggu
> Untuk informasi tentang konvensi penamaan yang digunakan dalam acara SQL tunggu Saya, lihat konvensi penamaan instrumen Skema Kinerja dalam dokumentasi Saya. SQL

cpu

Jumlah koneksi aktif yang siap dijalankan secara konsisten lebih tinggi daripada jumlahvCPUs. Untuk informasi selengkapnya, lihat cpu.

io/aurora_redo_log_flush

Sesi mempersistensi data ke penyimpanan Aurora. Biasanya, acara tunggu ini adalah untuk operasi tulis I/O di SQL Aurora My. Untuk informasi selengkapnya, lihat io/aurora_redo_log_flush.

io/aurora_respond_to_client

Pemrosesan kueri telah selesai dan hasilnya dikembalikan ke klien aplikasi untuk versi Aurora My berikut: 2.10.2 dan versi 2.10 yang lebih tinggi, SQL versi 2.09.3 dan versi 2.09 yang lebih tinggi, dan versi 2.07.7 dan versi 2.07 yang lebih tinggi. Bandingkan bandwidth jaringan kelas instans DB dengan ukuran set hasil yang ditampilkan. Selain itu, periksa waktu respons sisi klien. Jika klien tidak responsif dan tidak dapat memproses paket, TCP paket drop dan transmisi TCP ulang dapat terjadi. Situasi ini berdampak negatif pada bandwidth jaringan. Dalam versi yang lebih rendah

dari 2.10.2, 2.09.3, dan 2.07.7, peristiwa tunggu secara keliru menyertakan waktu idle. Untuk mempelajari cara menyesuaikan basis data Anda saat peristiwa waktu tunggu ini sering muncul, lihat io/aurora_respond_to_client.

io/file/csv/data

Thread menulis ke tabel dalam format value (CSV) dipisahkan koma. Periksa penggunaan CSV tabel Anda. Penyebab umum peristiwa ini adalah mengatur `log_output` pada tabel.

io/file/sql/binlog

Thread sedang menunggu file log biner (binlog) yang sedang ditulis ke disk.

io/redo_log_flush

Sesi mempersistensi data ke penyimpanan Aurora. Biasanya, acara tunggu ini adalah untuk operasi tulis I/O di SQL Aurora My. Untuk informasi selengkapnya, lihat io/redo_log_flush.

io/socket/sql/client_connection

Program `mysqld` sibuk membuat thread untuk menangani koneksi klien baru yang masuk. Untuk informasi selengkapnya, lihat io/socket/sql/client_koneksi.

io/table/sql/handler

Mesin sedang menunggu akses ke tabel. Peristiwa ini terjadi terlepas dari apakah data di-cache di pool buffer atau diakses pada disk. Untuk informasi selengkapnya, lihat io/table/sql/handler.

lock/table/sql/handler

Peristiwa tunggu ini adalah handler peristiwa tunggu kunci tabel. Untuk informasi lebih lanjut tentang peristiwa atom dan molekul dalam Skema Kinerja, lihat peristiwa atom dan molekul Skema Kinerja dalam dokumentasi SayaSQL.

synch/cond/innodb/row_lock_wait

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris database yang sama pada saat yang sama. Untuk informasi selengkapnya, lihat synch/cond/innodb/row_lock_tunggu.

synch/cond/innodb/row_lock_wait_cond

Beberapa DML pernyataan mengakses baris database yang sama pada saat yang sama. Untuk informasi selengkapnya, lihat synch/cond/innodb/row_lock_wait_cond.

MDLsinkroniskan/cond/sql/ _konteks:: _wait_status COND

Thread sedang menunggu kunci metadata tabel. Mesin menggunakan jenis kunci ini untuk mengelola akses konkuren ke skema basis data dan untuk memastikan konsistensi data. Untuk

informasi selengkapnya, lihat Mengoptimalkan operasi penguncian di SQL dokumentasi Saya. Untuk mempelajari cara menyesuaikan basis data Anda saat peristiwa ini sering muncul, lihat synch/cond/sql/MDL_konteks:: COND _wait_status.

sinkronisasi/cond/sql/ MYSQL _ _:: _selesai BIN LOG COND

Anda telah mengaktifkan pencatatan log biner. Mungkin ada throughput commit yang tinggi, transaksi dalam jumlah besar yang di-commit, atau replika yang membaca binlog. Pertimbangkan untuk menggunakan pernyataan multibaris atau menggabungkan pernyataan ke dalam satu transaksi. Di Aurora, gunakan basis data global alih-alih replikasi log biner, atau gunakan parameter `aurora_binlog_*`.

synch/mutex/innodb/aurora_lock_thread_slot_futex

Beberapa DML pernyataan mengakses baris database yang sama pada saat yang sama. Untuk informasi selengkapnya, lihat synch/mutex/innodb/aurora_lock_thread_slot_futex.

synch/mutex/innodb/buf_pool_mutex

Pool buffer tidak cukup besar untuk menampung set data kerja. Atau, beban kerja mengakses halaman dari tabel tertentu, yang mengakibatkan pertentangan di pool buffer. Untuk informasi selengkapnya, lihat synch/mutex/innodb/buf_pool_mutex.

synch/mutex/innodb/fil_system_mutex

Proses sedang menunggu akses ke cache memori ruang tabel. Untuk informasi selengkapnya, lihat synch/mutex/innodb/fil_system_mutex.

synch/mutex/innodb/trx_sys_mutex

Operasi memeriksa, memperbarui, menghapus, atau menambahkan transaksi IDs di InnoDB secara konsisten atau terkontrol. Operasi ini memerlukan panggilan mutex `trx_sys`, yang dilacak oleh instrumentasi Skema Performa. Operasi mencakup manajemen sistem transaksi ketika basis data dimulai atau dinonaktifkan, rollback, pembersihan undo, akses baca baris, dan beban pool buffer. Beban basis data yang tinggi dengan sejumlah besar transaksi mengakibatkan seringnya kemunculan peristiwa tunggu ini. Untuk informasi selengkapnya, lihat synch/mutex/innodb/trx_sys_mutex.

sinkronisasi/mutex/mysys/ _KEY: :cache_lock CACHE

`keycache->cache_lock`Mutex mengontrol akses ke cache kunci untuk ISAM tabel Saya. Meskipun Aurora My SQL tidak mengizinkan penggunaan ISAM tabel Saya untuk menyimpan data persisten, mereka digunakan untuk menyimpan tabel sementara internal. Pertimbangkan

untuk memeriksa penghitung status `created_tmp_disk_tables` atau `created_tmp_tables` karena dalam situasi tertentu, tabel sementara ditulis ke disk ketika tidak lagi muat dalam memori.

FILEsinkronisasi/mutex/sql/ _AS_:: _offset TABLE LOCK

Mesin memperoleh mutex ini saat membuka atau membuat file metadata tabel. Ketika peristiwa tunggu ini terjadi dengan frekuensi yang berlebihan, jumlah tabel yang dibuat atau dibuka telah melonjak.

sinkronisasi/mutex/sql/ _AS_:: _shim_lists FILE TABLE LOCK

Mesin memperoleh mutex ini saat melakukan operasi seperti `reset_size`, `detach_contents`, atau `add_contents` pada struktur internal yang melacak tabel yang dibuka. Mutex menyinkronkan akses ke konten daftar. Ketika peristiwa tunggu ini terjadi dengan frekuensi tinggi, ini menunjukkan perubahan mendadak dalam kumpulan tabel yang sebelumnya diakses. Mesin perlu mengakses tabel baru atau melepaskan konteks yang terkait dengan tabel yang diakses sebelumnya.

sinkroniskan/mutex/sql/ _buka LOCK

Jumlah tabel yang dibuka sesi Anda melebihi ukuran cache definisi tabel atau cache pembukaan tabel. Tingkatkan ukuran cache ini. Untuk informasi selengkapnya, lihat Cara Saya SQL membuka dan menutup tabel.

sinkroniskan/mutex/sql/ _table_cache LOCK

Jumlah tabel yang dibuka sesi Anda melebihi ukuran cache definisi tabel atau cache pembukaan tabel. Tingkatkan ukuran cache ini. Untuk informasi selengkapnya, lihat Cara Saya SQL membuka dan menutup tabel.

sinkronisasi/mutex/sql/LOG

Dalam peristiwa tunggu ini, terdapat thread yang menunggu kunci log. Misalnya, thread mungkin akan menunggu kunci untuk menulis ke file log kueri lambat.

MYSQLsinkronisasi/mutex/sql/ _ _:: _komit BIN LOG LOCK

Dalam peristiwa tunggu ini, ada thread yang menunggu untuk mendapatkan kunci dengan tujuan melakukan commit ke log biner. Pertentangan log biner dapat terjadi pada basis data dengan tingkat perubahan yang sangat tinggi. Tergantung pada versi My AndaSQL, ada kunci tertentu yang digunakan untuk melindungi konsistensi dan daya tahan log biner. Di RDS for MySQL, log biner digunakan untuk replikasi dan proses pencadangan otomatis. Di Aurora MySQL, log biner tidak diperlukan untuk replikasi atau cadangan asli. Opsi ini dinonaktifkan secara default, tetapi

dapat diaktifkan dan digunakan untuk replikasi eksternal atau penangkapan data perubahan. Untuk informasi selengkapnya, lihat Log biner di SQL dokumentasi Saya.

MYSQLsinkronisasi/mutex/sql/ _ _:: _dump_thread_metrics_collection BIN LOG LOCK

Jika pencatatan log biner diaktifkan, mesin memperoleh mutex ini saat mencetak metrik thread dump aktif ke log kesalahan mesin dan ke peta operasi internal.

MYSQLsinkronisasi/mutex/sql/ _ _:: _inactive_binlogs_map BIN LOG LOCK

Jika pencatatan log biner diaktifkan, mesin memperoleh mutex ini ketika menambahkan, menghapus dari, atau mencari melalui daftar file binlog di belakang yang terbaru.

MYSQLsinkronisasi/mutex/sql/ _ _:: _io_cache BIN LOG LOCK

Jika pencatatan log biner diaktifkan, mesin memperoleh mutex ini selama operasi cache Aurora IO binlog: mengalokasikan, mengubah ukuran, membebaskan, menulis, membaca, membersihkan, dan mengakses info cache. Jika peristiwa ini sering terjadi, mesin mengakses cache tempat peristiwa binlog disimpan. Untuk mengurangi waktu tunggu, kurangi commit. Cobalah mengelompokkan beberapa pernyataan ke dalam satu transaksi.

MYSQLsinkronisasi/mutex/sql/ _ _:: _log BIN LOG LOCK

Anda telah mengaktifkan pencatatan log biner. Mungkin ada throughput commit yang tinggi, banyak transaksi yang dilakukan, atau replika yang membaca binlog. Pertimbangkan untuk menggunakan pernyataan multibaris atau menggabungkan pernyataan ke dalam satu transaksi. Di Aurora, gunakan basis data global alih-alih replikasi log biner atau gunakan parameter `aurora_binlog_*`.

SERVERsinkronisasi/mutex/sql/ _:: _sinkronisasi THREAD LOCK

Mutex `SERVER_THREAD::LOCK_sync` diperoleh selama penjadwalan, pemrosesan, atau peluncuran thread untuk penulisan file. Terjadinya peristiwa tunggu yang berlebihan ini menunjukkan peningkatan aktivitas penulisan dalam basis data.

TABLESPACESsinkroniskan/mutex/sql/:kunci

Mesin memperoleh mutex `TABLESPACES:lock` selama operasi ruang tabel berikut: membuat, menghapus, memotong, dan memperpanjang. Terjadinya peristiwa tunggu yang berlebihan ini menunjukkan frekuensi operasi ruang tabel yang tinggi. Contohnya adalah memuat sejumlah besar data ke dalam basis data.

synch/rwlock/innodb/dict

Dalam peristiwa tunggu ini, ada thread yang menunggu rwlock yang dipertahankan di kamus data InnoDB.

synch/rwlock/innodb/dict_operation_lock

Dalam peristiwa tunggu ini, ada thread yang menahan kunci di operasi kamus data InnoDB.

synch/rwlock/innodb/dict sys RW lock

Sejumlah besar pernyataan bahasa kontrol data bersamaan (DCLs) dalam kode bahasa definisi data (DDLs) dipicu pada saat yang bersamaan. Kurangi ketergantungan aplikasi DDLs selama aktivitas aplikasi reguler.

synch/rwlock/innodb/index_tree_rw_lock

Sejumlah besar pernyataan bahasa manipulasi data (DML) yang sama mengakses objek database yang sama pada saat yang sama. Coba gunakan pernyataan multibaris. Selain itu, sebarkan beban kerja ke objek basis data yang berbeda-beda. Misalnya, dengan menerapkan partisi.

synch/sxlock/innodb/dict_operation_lock

Sejumlah besar pernyataan bahasa kontrol data bersamaan (DCLs) dalam kode bahasa definisi data (DDLs) dipicu pada saat yang bersamaan. Kurangi ketergantungan aplikasi DDLs selama aktivitas aplikasi reguler.

synch/sxlock/innodb/dict_sys_lock

Sejumlah besar pernyataan bahasa kontrol data bersamaan (DCLs) dalam kode bahasa definisi data (DDLs) dipicu pada saat yang bersamaan. Kurangi ketergantungan aplikasi DDLs selama aktivitas aplikasi reguler.

synch/sxlock/innodb/hash_table_locks

Sesi tidak dapat menemukan halaman dalam pool buffer. Mesin perlu membaca file atau memodifikasi daftar (LRU) yang paling tidak baru digunakan untuk kumpulan buffer. Pertimbangkan untuk meningkatkan ukuran cache buffer dan meningkatkan jalur akses untuk kueri yang relevan.

synch/sxlock/innodb/index_tree_rw_lock

Banyak pernyataan bahasa manipulasi data (DML) yang sama mengakses objek database yang sama pada saat yang sama. Coba gunakan pernyataan multibaris. Selain itu, sebarkan beban kerja ke objek basis data yang berbeda-beda. Misalnya, dengan menerapkan partisi.

Untuk informasi selengkapnya tentang pemecahan masalah peristiwa penantian sinkronisasi, lihat Mengapa instans SQL DB Saya menampilkan banyak sesi aktif yang menunggu acara SYNCH tunggu di Performance Insights? .

## Aurora Utas saya menyatakan SQL

Berikut ini adalah beberapa status benang merah untuk Aurora My. SQL

memeriksa izin

Thread memeriksa apakah server memiliki hak akses yang diperlukan untuk menjalankan pernyataan.

memeriksa cache kueri untuk kueri

Server memeriksa apakah kueri saat ini ada di cache kueri.

dibersihkan

Ini adalah status akhir dari koneksi yang pekerjaannya selesai, tetapi belum ditutup oleh klien. Solusi terbaik adalah secara eksplisit menutup koneksi dalam kode. Atau, Anda dapat menetapkan nilai yang lebih rendah untuk `wait_timeout` di grup parameter Anda.

menutup tabel

Thread melakukan flushing data tabel yang diubah ke disk dan menutup tabel yang digunakan. Jika operasi ini tidak cepat, periksa metrik konsumsi bandwidth jaringan berdasarkan bandwidth jaringan kelas instans. Selain itu, periksa apakah nilai untuk parameter `table_open_cache` dan `table_definition_cache` memungkinkan tabel yang cukup dibuka secara bersamaan sehingga mesin tidak perlu sering membuka dan menutup tabel. Parameter ini memengaruhi konsumsi memori pada instans.

konversi HEAP ke My ISAM

Kueri mengonversi tabel sementara dari dalam memori ke di disk. Konversi ini diperlukan karena tabel sementara yang dibuat oleh My SQL dalam langkah perantara pemrosesan kueri tumbuh terlalu besar untuk memori. Periksa nilai `tmp_table_size` dan `max_heap_table_size`. Di versi yang lebih baru, nama status thread ini adalah `converting HEAP to ondisk`.

mengkonversi HEAP ke ondisk

Thread mengonversi tabel sementara internal dari tabel dalam memori ke tabel di disk.

salin ke tabel sementara

Thread sedang memproses pernyataan `ALTER TABLE`. Status ini terjadi setelah tabel dengan struktur baru telah dibuat, tetapi sebelum baris disalin ke dalamnya. Untuk thread dalam status ini, Anda dapat menggunakan Skema Performa untuk mendapatkan informasi tentang progres operasi penyalinan.

membuat indeks pengurutan

Aurora My SQL melakukan semacam karena tidak dapat menggunakan indeks yang ada untuk memenuhi `ORDER BY` atau `GROUP BY` klausa kueri. Untuk informasi selengkapnya, lihat creating sort index.

membuat tabel

Thread membuat tabel permanen atau sementara.

commit tertunda ok selesai

Komit asinkron di Aurora My SQL telah menerima pengakuan dan selesai.

commit tertunda ok diinisiasi

SQLUtas Aurora My telah memulai proses komit async tetapi sedang menunggu pengakuan. Status ini biasanya menunjukkan waktu commit yang sebenarnya dari suatu transaksi.

pengiriman tertunda ok selesai

Utas Aurora SQL Pekerja saya yang terikat ke koneksi dapat dibebaskan saat respons dikirim ke klien. Utas dapat memulai pekerjaan lain. Status `delayed send ok` berarti bahwa konfirmasi asinkron ke klien selesai.

pengiriman tertunda ok diinisiasi

Utas Aurora SQL Pekerja saya telah mengirim respons secara asinkron ke klien dan sekarang bebas melakukan pekerjaan untuk koneksi lain. Transaksi telah memulai proses commit asinkron yang belum dikonfirmasi.

mengeksekusi

Thread telah mulai menjalankan pernyataan.

membebaskan item

Thread telah menjalankan perintah. Beberapa pembebasan item yang dilakukan selama status ini memerlukan cache kueri. Status ini biasanya diikuti dengan pembersihan.

inisialisasi

Status ini terjadi sebelum inisialisasi pernyataan `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, atau `UPDATE`. Tindakan dalam status ini termasuk flushing log biner atau log InnoDB, dan beberapa pembersihan cache kueri.

Sumber telah mengirim semua binlog ke replika; menunggu pembaruan lebih lanjut

Simpul primer telah menyelesaikan bagiannya dalam replikasi. Thread sedang menunggu lebih banyak kueri untuk dijalankan sehingga dapat menulis ke log biner (binlog).

membuka tabel

Thread mencoba membuka tabel. Operasi ini cepat kecuali jika pernyataan `ALTER TABLE` atau `LOCK TABLE` perlu diselesaikan, atau melebihi nilai `table_open_cache`.

mengoptimalkan

Server melakukan optimisasi awal untuk kueri.

mempersiapkan

Status ini terjadi selama optimisasi kueri.

akhir kueri

Status ini terjadi setelah memproses kueri, tetapi sebelum status membebaskan item.

menghapus duplikat

Aurora My SQL tidak dapat mengoptimalkan `DISTINCT` operasi pada tahap awal kueri. Aurora My SQL harus menghapus semua baris duplikat sebelum mengirim hasilnya ke klien.

mencari baris untuk pembaruan

Thread mencari semua baris yang cocok sebelum memperbaruinya. Tahap ini diperlukan jika `UPDATE` mengubah indeks yang digunakan mesin untuk menemukan baris.

mengirim peristiwa binlog ke slave

Thread membaca peristiwa dari log biner dan mengirimkannya ke replika.

mengirim hasil yang di-cache ke klien

Server mengambil hasil kueri dari cache kueri dan mengirimkannya ke klien.

mengirim data

Thread sedang membaca dan memproses baris untuk pernyataan `SELECT`, tetapi belum mulai mengirim data ke klien. Proses ini mengidentifikasi halaman mana yang berisi hasil yang diperlukan untuk memenuhi kueri. Untuk informasi selengkapnya, lihat sending data.

mengirim ke klien

Server menulis paket ke klien. Dalam SQL versi saya sebelumnya, acara tunggu ini diberi label`writing to net`.

memulai

Ini adalah tahap pertama di awal eksekusi pernyataan.

statistik

Server menghitung statistik untuk mengembangkan rencana eksekusi kueri. Jika thread dalam status ini dalam waktu yang lama, server mungkin terikat pada disk sambil melakukan pekerjaan lain.

menyimpan hasil dalam cache kueri

Server menyimpan hasil kueri dalam cache kueri.

kunci sistem

Thread telah memanggil `mysql_lock_tables`, tetapi status thread belum diperbarui sejak panggilan ini. Status umum ini terjadi karena berbagai alasan.

perbarui

Thread sedang bersiap untuk mulai memperbarui tabel.

memperbarui

Thread sedang mencari baris dan memperbaruinya.

kunci pengguna

Thread mengeluarkan panggilan `GET_LOCK`. Thread meminta kunci advisory dan sedang menunggunya, atau berencana untuk memintanya.

menunggu pembaruan lainnya

Simpul primer telah menyelesaikan bagiannya dalam replikasi. Thread sedang menunggu lebih banyak kueri untuk dijalankan sehingga dapat menulis ke log biner (binlog).

menunggu kunci metadata skema

Ini adalah peristiwa tunggu untuk kunci metadata.

menunggu kunci metadata fungsi tersimpan

Ini adalah peristiwa tunggu untuk kunci metadata.

menunggu kunci metadata prosedur tersimpan

Ini adalah peristiwa tunggu untuk kunci metadata.

menunggu flushing tabel

Thread sedang menjalankan `FLUSH TABLES` dan sedang menunggu semua thread untuk menutup tabelnya. Atau, thread menerima notifikasi bahwa struktur dasar untuk tabel berubah, sehingga harus membuka kembali tabel untuk mendapatkan struktur baru. Untuk membuka kembali tabel, thread harus menunggu sampai semua thread lainnya menutup tabel. Notifikasi ini terjadi jika thread lain telah menggunakan salah satu pernyataan berikut di tabel: `FLUSH TABLES`, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, atau `OPTIMIZE TABLE`.

menunggu kunci tingkat tabel

Satu sesi mempertahankan kunci di tabel sementara sesi lain mencoba mendapatkan kunci yang sama di tabel yang sama.

menunggu kunci metadata tabel

Aurora My SQL menggunakan penguncian metadata untuk mengelola akses bersamaan ke objek database dan untuk memastikan konsistensi data. Dalam peristiwa tunggu ini, satu sesi mempertahankan kunci metadata di tabel sementara sesi lain mencoba mendapatkan kunci yang sama di tabel yang sama. Saat Skema Performa diaktifkan, status thread ini dilaporkan sebagai peristiwa tunggu `synch/cond/sql/MDL_context::COND_wait_status`.

menulis ke net

Server sedang menulis paket ke jaringan. Di SQL versi saya yang lebih baru, acara tunggu ini diberi label`Sending to client`.

## Tingkat isolasi Aurora MySQL

Pelajari bagaimana instans DB di klaster Aurora MySQL menerapkan properti basis data isolasi. Topik ini menjelaskan bagaimana perilaku default Aurora MySQL menyeimbangkan antara konsistensi yang ketat dan performa yang tinggi. Anda dapat menggunakan informasi ini untuk

membantu memutuskan kapan akan mengubah pengaturan default berdasarkan karakteristik beban kerja Anda.

### Tingkat isolasi yang tersedia untuk instans penulis

Anda dapat menggunakan tingkat isolasi `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED`, dan `SERIALIZABLE` pada instans primer klaster DB Aurora MySQL. Tingkat isolasi ini berfungsi di Aurora MySQL sama seperti di RDS for MySQL.

### Tingkat isolasi REPEATABLE READ untuk instans pembaca

Secara default, instans DB Aurora MySQL yang dikonfigurasi sebagai Replika Aurora hanya baca selalu menggunakan tingkat isolasi `REPEATABLE READ`. Instans DB ini mengabaikan pernyataan `SET TRANSACTION ISOLATION LEVEL` dan terus menggunakan tingkat isolasi `REPEATABLE READ`.

### Tingkat isolasi READ COMMITTED untuk instans pembaca

Jika aplikasi Anda mencakup beban kerja sarat penulisan pada instans primer dan kueri yang berjalan lama pada Replika Aurora, Anda mungkin mengalami lag pembuangan yang substansial. Lag pembuangan terjadi jika pengumpulan sampah internal diblokir oleh kueri yang berjalan lama. Gejala yang Anda lihat adalah nilai yang tinggi untuk `history list length` dalam output dari perintah `SHOW ENGINE INNODB STATUS`. Anda dapat memantau nilai ini menggunakan `RollbackSegmentHistoryListLength` metrik di CloudWatch. Lag pembuangan yang substansial dapat mengurangi efektivitas indeks sekunder, menurunkan performa kueri secara keseluruhan, dan menyebabkan pemborosan ruang penyimpanan.

Jika Anda mengalami masalah seperti itu, Anda dapat mengatur pengaturan konfigurasi tingkat sesi Aurora MySQL, `aurora_read_replica_read_committed`, agar menggunakan tingkat isolasi `READ COMMITTED` pada Replika Aurora. Saat menerapkan pengaturan ini, Anda dapat membantu mengurangi pelambatan dan ruang terbuang yang dapat timbul karena melakukan kueri berjalan lama pada saat yang sama seperti transaksi yang mengubah tabel Anda.

Kami menyarankan Anda untuk memahami perilaku tertentu Aurora MySQL karena isolasi `READ COMMITTED` sebelum menggunakan pengaturan ini. Perilaku Replika Aurora `READ COMMITTED` sesuai dengan standar ANSI SQL. Namun, isolasinya tidak seketat perilaku `READ COMMITTED` MySQL biasa yang mungkin Anda kenal. Dengan demikian, Anda mungkin melihat hasil kueri yang berbeda berdasarkan `READ COMMITTED` di replika baca Aurora MySQL daripada kueri yang sama berdasarkan `READ COMMITTED` di instans primer Aurora MySQL atau di RDS for MySQL. Anda dapat mempertimbangkan untuk menggunakan pengaturan

`aurora_read_replica_read_committed` untuk kasus seperti laporan komprehensif yang memindai basis data yang sangat besar. Sebaliknya, Anda dapat menghindarinya untuk kueri pendek dengan set hasil kecil yang mementingkan presisi dan keterulangan.

Tingkat isolasi `READ COMMITTED` tidak tersedia untuk sesi dengan klaster sekunder dalam basis data global Aurora yang menggunakan fitur penerusan tulis. Untuk informasi tentang penerusan tulis, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Menggunakan READ COMMITTED untuk pembaca

Untuk menggunakan tingkat isolasi `READ COMMITTED` untuk Replika Aurora, atur pengaturan konfigurasi `aurora_read_replica_read_committed` ke `ON`. Gunakan pengaturan ini di tingkat sesi saat terhubung ke Replika Aurora tertentu. Untuk melakukannya, jalankan perintah SQL berikut.

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

Anda dapat menggunakan pengaturan konfigurasi ini sementara waktu untuk melakukan kueri interaktif satu kali. Anda mungkin juga ingin menjalankan aplikasi pelaporan atau analisis data yang mendapatkan manfaat dari tingkat isolasi `READ COMMITTED`, sementara membiarkan pengaturan default tidak berubah untuk aplikasi lain.

Saat pengaturan `aurora_read_replica_read_committed` diaktifkan, gunakan perintah `SET TRANSACTION ISOLATION LEVEL` untuk menentukan tingkat isolasi untuk transaksi yang sesuai.

```
set transaction isolation level read committed;
```

Perbedaan dalam perilaku READ COMMITTED pada replika Aurora

Pengaturan `aurora_read_replica_read_committed` membuat tingkat isolasi `READ COMMITTED` tersedia untuk Replika Aurora, dengan perilaku konsistensi yang dioptimalkan untuk transaksi yang berjalan lama. Tingkat isolasi `READ COMMITTED` pada Replika Aurora memiliki isolasi yang tidak terlalu ketat dibandingkan pada instans primer Aurora. Oleh karena itu, aktifkan pengaturan ini hanya di Replika Aurora, yang memungkinkan Anda mengetahui bahwa kueri Anda dapat menerima kemungkinan jenis hasil tertentu yang tidak konsisten.

Kueri Anda dapat mengalami beberapa jenis anomali baca ketika pengaturan `aurora_read_replica_read_committed` diaktifkan. Dua jenis anomali ini sangat penting untuk dipahami dan ditangani dalam kode aplikasi Anda. Pembacaan yang tidak dapat diulang terjadi saat transaksi lain di-commit saat kueri Anda sedang berjalan. Kueri yang berjalan lama dapat melihat

data yang berbeda pada awal kueri dibandingkan dengan yang dilihat pada akhir kueri. Pembacaan phantom terjadi ketika transaksi lain menyebabkan baris yang ada disusun ulang saat kueri Anda berjalan, dan satu atau beberapa baris dibaca dua kali oleh kueri Anda.

Kueri Anda mungkin mengalami jumlah baris yang tidak konsisten sebagai hasil dari pembacaan phantom. Kueri Anda juga dapat memberikan hasil yang tidak lengkap atau tidak konsisten karena pembacaan yang tidak dapat diulang. Misalnya, anggaplah sebuah operasi join merujuk ke tabel yang secara bersamaan dimodifikasi oleh pernyataan SQL seperti `INSERT` atau `DELETE`. Dalam kasus ini, kueri join mungkin membaca baris dari satu tabel, tetapi bukan baris yang sesuai dari tabel lain.

Standar ANSI SQL memungkinkan kedua perilaku ini untuk tingkat isolasi `READ COMMITTED`. Namun, perilaku tersebut berbeda dari implementasi `READ COMMITTED` MySQL biasa. Jadi, sebelum mengaktifkan pengaturan `aurora_read_replica_read_committed`, periksa kode SQL yang ada untuk memverifikasi apakah kode ini beroperasi seperti yang diharapkan berdasarkan model konsistensi yang lebih longgar.

Jumlah baris dan hasil lainnya mungkin tidak terlalu konsisten berdasarkan tingkat isolasi `READ COMMITTED` sementara pengaturan ini diaktifkan. Dengan demikian, Anda biasanya mengaktifkan pengaturan hanya saat menjalankan kueri analitik yang mengumpulkan data dalam jumlah besar dan tidak memerlukan presisi absolut. Jika Anda tidak memiliki jenis kueri yang berjalan lama ini bersama dengan beban kerja rata penulisan, Anda mungkin tidak memerlukan pengaturan `aurora_read_replica_read_committed`. Tanpa kombinasi kueri yang berjalan lama dan beban kerja yang sarat penulisan, Anda cenderung tidak akan menghadapi masalah dengan panjang daftar riwayat.

Example Kueri yang menunjukkan perilaku isolasi untuk READ COMMITTED pada Replika Aurora

Contoh berikut menunjukkan bagaimana kueri `READ COMMITTED` pada Replika Aurora dapat memberikan hasil yang tidak dapat diulang jika transaksi mengubah tabel terkait pada saat yang sama. Tabel `BIG_TABLE` berisi 1 juta baris sebelum kueri dimulai. Pernyataan bahasa manipulasi data (DML) lainnya menambahkan, menghapus, atau mengubah baris saat sedang berjalan.

Kueri pada instans primer Aurora berdasarkan tingkat isolasi `READ COMMITTED` memberikan hasil yang dapat diprediksi. Namun demikian, overhead dalam mempertahankan tampilan baca yang konsisten selama masa pakai setiap kueri yang berjalan lama dapat menyebabkan pengumpulan sampah yang menimbulkan biaya mahal pada lain waktu.

Kueri pada Replika Aurora berdasarkan tingkat isolasi `READ COMMITTED` akan dioptimalkan untuk meminimalkan overhead pengumpulan sampah. Komprominya adalah bahwa hasilnya

mungkin berbeda-beda bergantung pada apakah kueri memuat baris yang ditambahkan, dihapus, atau disusun ulang oleh transaksi yang di-commit saat kueri berjalan. Kueri diizinkan untuk mempertimbangkan baris-baris ini, tetapi tidak diwajibkan. Untuk tujuan demonstrasi, kueri hanya memeriksa jumlah baris dalam tabel dengan menggunakan fungsi `COUNT(*)`.

| Waktu | Pernyataan DML di instans primer Aurora | Kueri di instans primer Aurora dengan READ COMMITTED | Kueri di replika Aurora dengan READ COMMITTED |
|---|---|---|---|
| T1 | `INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;` | | |
| T2 | | Q1: `SELECT COUNT(*) FROM big_table;` | Q2: `SELECT COUNT(*) FROM big_table;` |
| T3 | `INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;` | | |
| T4 | | Jika Q1 selesai sekarang, hasilnya adalah 1.000.000. | Jika Q2 selesai sekarang, hasilnya adalah 1.000.000 atau 1.000.001. |
| T5 | `DELETE FROM big_table LIMIT 2; COMMIT;` | | |
| T6 | | Jika Q1 selesai sekarang, hasilnya adalah 1.000.000. | Jika Q2 selesai sekarang, hasilnya adalah 1.000.000 atau 1.000.001 |

| Waktu | Pernyataan DML di instans primer Aurora | Kueri di instans primer Aurora dengan READ COMMITTED | Kueri di replika Aurora dengan READ COMMITTED |
|---|---|---|---|
| | | | atau 999.999 atau 999.998. |
| T7 | `UPDATE big_table SET c2 = CONCAT(c2 ,c2,c2); COMMIT;` | | |
| T8 | | Jika Q1 selesai sekarang, hasilnya adalah 1.000.000. | Jika Q2 selesai sekarang, hasilnya adalah 1.000.000 atau 1.000.001 atau 999.999, atau mungkin angka tertentu yang lebih tinggi. |
| T9 | | Q3: `SELECT COUNT(*) FROM big_table;` | Q4: `SELECT COUNT(*) FROM big_table;` |
| T10 | | Jika Q3 selesai sekarang, hasilnya adalah 999.999. | Jika Q4 selesai sekarang, hasilnya adalah 999.999. |
| T11 | | Q5: `SELECT COUNT(*) FROM parent_table p JOIN child_tab le c ON (p.id = c.id) WHERE p.id = 1000;` | Q6: `SELECT COUNT(*) FROM parent_table p JOIN child_tab le c ON (p.id = c.id) WHERE p.id = 1000;` |

| Waktu | Pernyataan DML di instans primer Aurora | Kueri di instans primer Aurora dengan READ COMMITTED | Kueri di replika Aurora dengan READ COMMITTED |
|---|---|---|---|
| T12 | `INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;` | | |
| T13 | | Jika Q5 selesai sekarang, hasilnya adalah 0. | Jika Q6 selesai sekarang, hasilnya adalah 0 atau 1. |

Jika kueri selesai dengan cepat, sebelum transaksi lain melakukan pernyataan DML dan di-commit, hasilnya dapat diprediksi dan sama antara instans primer dan Replika Aurora. Mari kita periksa perbedaan perilaku secara mendetail, dimulai dengan kueri pertama.

Hasil untuk Q1 sangat dapat diprediksi, karena `READ COMMITTED` pada instans primer menggunakan model konsistensi kuat yang serupa dengan tingkat isolasi `REPEATABLE READ`.

Hasil untuk Q2 dapat bervariasi bergantung pada transaksi apa yang dilakukan saat kueri berjalan. Misalnya, transaksi lain melakukan pernyataan DML dan di-commit saat kueri sedang berjalan. Dalam hal ini, kueri di Replika Aurora dengan tingkat isolasi `READ COMMITTED` mungkin atau mungkin tidak memperhitungkan perubahan tersebut. Hitungan baris tidak dapat diprediksi dengan cara yang sama seperti berdasarkan tingkat isolasi `REPEATABLE READ`. Hitungan baris juga tidak dapat diprediksi seperti kueri yang berjalan berdasarkan tingkat isolasi `READ COMMITTED` pada instans primer, atau pada instans RDS for MySQL.

Pernyataan `UPDATE` di T7 tidak benar-benar mengubah jumlah baris dalam tabel. Namun, dengan mengubah panjang kolom panjang variabel, pernyataan ini dapat menyebabkan baris-baris tersebut disusun ulang secara internal. Transaksi `READ COMMITTED` yang berjalan lama dapat melihat baris

versi lama, lalu dalam kueri yang sama, melihat versi baru dari baris yang sama. Kueri ini juga dapat melewati versi baris lama dan baru, sehingga jumlah baris mungkin berbeda dari yang diharapkan.

Hasil Q5 dan Q6 mungkin identik atau sedikit berbeda. Kueri Q6 pada Replika Aurora berdasarkan `READ COMMITTED` dapat melihat, tetapi tidak wajib untuk melihat, baris baru yang di-commit saat kueri berjalan. Kueri ini juga dapat melihat baris dari satu tabel, tetapi tidak dari tabel lainnya. Jika kueri join tidak menemukan baris yang cocok di kedua tabel, kueri ini akan menghasilkan angka nol. Jika kueri ini menemukan baris baru di `PARENT_TABLE` dan `CHILD_TABLE`, kueri tersebut akan menampilkan satu angka. Dalam kueri yang berjalan lama, pencarian dari tabel gabungan dapat terjadi pada waktu yang terpisah secara luas.

> **Note**
>
> Perbedaan perilaku ini bergantung pada waktu transaksi di-commit dan kueri memproses baris tabel yang mendasarinya. Oleh karena itu, kemungkinan besar Anda akan melihat perbedaan tersebut dalam kueri laporan yang memakan waktu beberapa menit atau jam dan yang berjalan di klaster Aurora yang memproses transaksi OLTP pada saat yang sama. Ini adalah jenis beban kerja campuran yang mendapatkan manfaat maksimal dari tingkat isolasi `READ COMMITTED` pada Replika Aurora.

## Aurora Petunjuk saya SQL

Anda dapat menggunakan SQL petunjuk dengan kueri Aurora SQL My untuk menyempurnakan kinerja. Anda juga dapat menggunakan petunjuk agar rencana eksekusi untuk kueri penting tidak berubah karena kondisi yang tidak dapat diprediksi.

> **Tip**
>
> Untuk memverifikasi efek petunjuk pada kueri, periksa rencana kueri yang dihasilkan oleh pernyataan `EXPLAIN`. Bandingkan rencana kueri dengan atau tanpa petunjuk.

Di Aurora My SQL versi 3, Anda dapat menggunakan semua petunjuk yang tersedia di My SQL Community Edition 8.0. Untuk informasi selengkapnya tentang petunjuk ini, lihat Petunjuk Pengoptimal di Manual Referensi Saya SQL.

Petunjuk berikut tersedia di Aurora SQL My versi 2. Petunjuk ini berlaku untuk kueri yang menggunakan fitur gabungan hash di Aurora My SQL versi 2, terutama kueri yang menggunakan optimasi kueri paralel.

PQ, NO_PQ

Menentukan apakah akan memaksa pengoptimisasi untuk menggunakan kueri paralel berdasarkan per tabel atau per kueri.

PQ memaksa pengoptimisasi untuk menggunakan kueri paralel untuk tabel tertentu atau seluruh kueri (blok). NO_PQ mencegah pengoptimisasi menggunakan kueri paralel untuk tabel tertentu atau seluruh kueri (blok).

Petunjuk ini tersedia di Aurora SQL My versi 2.11 dan lebih tinggi. Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

> **Note**
>
> Menentukan nama tabel akan memaksa pengoptimisasi untuk menerapkan petunjuk PQ/NO_PQ hanya pada tabel pilihan tersebut. Jika nama tabel tidak ditentukan, petunjuk PQ/NO_PQ akan dipaksa pada semua tabel yang terpengaruh oleh blok kueri.

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

### HASH_JOIN, TIDAK_ _ HASH JOIN

Mengaktifkan atau menonaktifkan kemampuan pengoptimisasi kueri paralel untuk memilih apakah akan menggunakan metode optimisasi hash join untuk kueri. `HASH_JOIN` memungkinkan pengoptimisasi menggunakan hash join jika mekanismenya lebih efisien. `NO_HASH_JOIN` mencegah pengoptimisasi menggunakan hash join untuk kueri. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi. Ini tidak berpengaruh di Aurora My SQL versi 3.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

### HASH_ JOIN _PROBING, TIDAK _ _ HASH _ JOIN PROBING

Di kueri hash join, tentukan apakah akan menggunakan tabel yang ditentukan untuk sisi probe dari join. Kueri ini menguji apakah nilai kolom dari tabel build ada di tabel probe, bukan membaca seluruh konten tabel probe. Anda dapat menggunakan `HASH_JOIN_PROBING` dan `HASH_JOIN_BUILDING` untuk menentukan cara kueri hash join diproses, tanpa mengatur ulang susunan tabel dalam teks kueri. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi. Ini tidak berpengaruh di Aurora My SQL versi 3.

Contoh berikut menunjukkan cara menggunakan petunjuk ini. Menentukan petunjuk `HASH_JOIN_PROBING` untuk tabel `T2` memiliki efek yang sama seperti menentukan `NO_HASH_JOIN_PROBING` untuk tabel `T1`.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

### HASH_ JOIN _BUILDING, TIDAK _ _ HASH _ JOIN BUILDING

Di kueri hash join, tentukan apakah akan menggunakan tabel yang ditentukan untuk sisi build dari join. Kueri ini memproses semua baris dari tabel ini untuk membuat daftar nilai kolom untuk melakukan referensi silang dengan tabel lain. Anda dapat menggunakan `HASH_JOIN_PROBING` dan `HASH_JOIN_BUILDING` untuk menentukan cara kueri hash join diproses, tanpa mengatur

ulang susunan tabel dalam teks kueri. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi. Ini tidak berpengaruh di Aurora My SQL versi 3.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini. Menentukan petunjuk `HASH_JOIN_BUILDING` untuk tabel `T2` memiliki efek yang sama seperti menentukan `NO_HASH_JOIN_BUILDING` untuk tabel `T1`.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

Menentukan bahwa tabel dalam kueri digabungkan berdasarkan urutan pencantumannya dalam kueri. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Ini dimaksudkan sebagai pengganti SQL `STRAIGHT_JOIN` petunjuk Saya dan setara dengan petunjuk SQL JOIN_ FIXED _ ORDER Saya. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

Menentukan urutan join untuk tabel dalam kueri. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Ini setara dengan ORDER petunjuk SQL JOIN_ Saya. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

Menentukan tabel yang dimasukkan pertama dalam urutan join. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Ini setara dengan PREFIX petunjuk SQL JOIN_ Saya. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

Menentukan tabel yang dimasukkan terakhir dalam urutan join. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Ini setara dengan SUFFIX petunjuk SQL JOIN_ Saya. Petunjuk ini tersedia di Aurora SQL My versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

Untuk informasi tentang menggunakan kueri hash join, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

## Aurora Referensi prosedur SQL tersimpan saya

Anda dapat mengelola cluster Aurora My SQL DB Anda dengan memanggil prosedur tersimpan bawaan.

Topik

- Mengumpulkan dan memelihara Sejarah Status Global
- Mengkonfigurasi, memulai, dan menghentikan replikasi log biner (binlog)
- Mengakhiri sesi atau kueri
- Mereplikasi transaksi menggunakan GTIDs
- Memutar log kueri
- Mengatur dan menampilkan konfigurasi log biner

## Mengumpulkan dan memelihara Sejarah Status Global

Amazon RDS menyediakan serangkaian prosedur yang mengambil snapshot dari nilai variabel status dari waktu ke waktu dan menuliskannya ke tabel, bersama dengan perubahan apa pun sejak snapshot terakhir. Infrastruktur ini disebut Global Status History. Untuk informasi selengkapnya, lihat Mengelola Global Status History.

Prosedur tersimpan berikut mengelola bagaimana Global Status History dikumpulkan dan dipelihara.

Topik

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

### mysql.rds_collect_global_status_history

Mengambil snapshot sesuai permintaan untuk Global Status History.

Sintaks

```
CALL mysql.rds_collect_global_status_history;
```

### mysql.rds_disable_gsh_collector

Menonaktifkan snapshot yang diambil oleh Global Status History.

Sintaks

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

Menonaktifkan rotasi tabel `mysql.global_status_history`.

Sintaks

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

Mengaktifkan Global Status History untuk mengambil snapshot default pada interval yang ditentukan oleh `rds_set_gsh_collector`.

Sintaks

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

Mengaktifkan rotasi konten tabel `mysql.global_status_history` ke `mysql.global_status_history_old` pada interval yang ditentukan oleh `rds_set_gsh_rotation`.

Sintaks

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

Merotasi konten tabel `mysql.global_status_history` ke `mysql.global_status_history_old` sesuai permintaan.

Sintaks

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

Menentukan interval, dalam menit, di antara snapshot yang diambil oleh Global Status History.

Sintaks

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

Parameter

*intervalPeriod*

Interval, dalam menit, di antara snapshot. Nilai default-nya adalah 5.

mysql.rds_set_gsh_rotation

Menentukan interval, dalam hari, antara rotasi tabel `mysql.global_status_history`.

Sintaks

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

Parameter

*intervalPeriod*

Interval, dalam hari, di antara rotasi tabel. Nilai default-nya adalah 7.

## Mengkonfigurasi, memulai, dan menghentikan replikasi log biner (binlog)

Anda dapat memanggil prosedur tersimpan berikut saat terhubung ke instance utama di cluster Aurora MySQL. Prosedur ini mengontrol bagaimana transaksi direplikasi dari database eksternal ke Aurora SQL My, atau dari Aurora SQL My ke database eksternal.

Topik


- mysql.rds_disable_session_binlog (Aurora Versi saya 2) SQL
- mysql.rds_enable_session_binlog (Aurora Versi saya 2) SQL
- mysql.rds_import_binlog_ssl_material
- mysql.rds_next_master_log SQL
- mysql.rds_next_source_log ) SQL
- mysql.rds_remove_binlog_ssl_material
- mysql.rds_reset_external_master ) RDS SQL SQL
- mysql.rds_reset_external_source ) SQL SQL
- mysql.rds_set_binlog_source_ssl (Aurora Versi saya 3) SQL
- mysql.rds_set_external_master ) SQL SQL
- mysql.rds_set_external_source ) SQL
- mysql.rds_set_external_master_with_auto_position (Aurora Versi saya 2) SQL
- mysql.rds_set_external_source_with_auto_position (Aurora Versi saya 3) SQL
- mysql.rds_set_master_auto_position ) SQL
- mysql.rds_set_read_only (Aurora Versi saya 3) SQL
- mysql.rds_set_session_binlog_format (Aurora Versi saya 2) SQL
- mysql.rds_set_source_auto_position ) SQL SQL
- mysql.rds_skip_repl_error
- mysql.rds_start_replication
- mysql.rds_start_replication_until (Aurora Versi saya 3) SQL
- mysql.rds_stop_replication


### mysql.rds_disable_session_binlog (Aurora Versi saya 2) SQL

Menonaktifkan pencatatan log biner untuk sesi saat ini dengan mengatur variabel `sql_log_bin` ke `OFF`.

Sintaksis

```
CALL mysql.rds_disable_session_binlog;
```

Parameter

Tidak ada

Catatan penggunaan

Untuk cluster Aurora My SQL DB, Anda memanggil prosedur tersimpan ini saat terhubung ke instance utama.

Untuk Aurora, prosedur ini didukung untuk Aurora My SQL versi 2.12 dan versi My 5.7 yang lebih tinggi. SQL

> **Note**
>
> Di Aurora My SQL versi 3, Anda dapat menggunakan perintah berikut untuk menonaktifkan pencatatan biner untuk sesi saat ini jika Anda memiliki hak istimewa: `SESSION_VARIABLES_ADMIN`
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog (Aurora Versi saya 2) SQL

Mengaktifkan pencatatan log biner untuk sesi saat ini dengan mengatur variabel `sql_log_bin` ke `ON`.

Sintaksis

```
CALL mysql.rds_enable_session_binlog;
```

Parameter

Tidak ada

Catatan penggunaan

Untuk cluster Aurora My SQL DB, Anda memanggil prosedur tersimpan ini saat terhubung ke instance utama.

Untuk Aurora, prosedur ini didukung untuk Aurora My SQL versi 2.12 dan versi My 5.7 yang lebih tinggi. SQL

> **Note**
>
> Di Aurora My SQL versi 3, Anda dapat menggunakan perintah berikut untuk mengaktifkan pencatatan biner untuk sesi saat ini jika Anda memiliki hak istimewa: `SESSION_VARIABLES_ADMIN`
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_import_binlog_ssl_material

Mengimpor sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien ke dalam cluster Aurora SQL My DB. Informasi diperlukan untuk SSL komunikasi dan replikasi terenkripsi.

> **Note**
>
> Saat ini, prosedur ini didukung untuk Aurora My SQL versi 2:2.09.2, 2.10.0, 2.10.1, dan 2.11.0; dan versi 3:3.01.1 dan lebih tinggi.

Sintaks

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

Parameter

*ssl_material*

JSONpayload yang berisi isi file format.pem berikut untuk klien Saya: SQL

- “ssl_ca”:”” *Certificate authority certificate*
- “ssl_cert”:”” *Client certificate*
- “ssl_key”:”” *Client key*

Catatan penggunaan

Persiapkan untuk replikasi terenkripsi sebelum Anda menjalankan prosedur ini:

- Jika Anda belum SSL mengaktifkan instance database SQL sumber saya eksternal dan tidak memiliki kunci klien dan sertifikat klien yang disiapkan, aktifkan SSL di server SQL database Saya dan buat kunci klien dan sertifikat klien yang diperlukan.
- Jika SSL diaktifkan pada instance database sumber eksternal, berikan kunci klien dan sertifikat untuk cluster Aurora My SQL DB. Jika Anda tidak memilikinya, buat kunci dan sertifikat baru untuk cluster Aurora My SQL DB. Untuk menandatangani sertifikat klien, Anda harus memiliki kunci otoritas sertifikat yang Anda gunakan untuk mengonfigurasi SSL pada instance database SQL sumber saya eksternal.

Untuk informasi selengkapnya, lihat Membuat SSL sertifikat dan kunci menggunakan openssl di dokumentasi SayaSQL.

> **Important**
>
> Setelah Anda mempersiapkan replikasi terenkripsi, gunakan SSL koneksi untuk menjalankan prosedur ini. Kunci klien tidak boleh ditransfer melalui koneksi yang tidak aman.

Prosedur ini mengimpor SSL informasi dari SQL database Saya eksternal ke cluster Aurora SQL My DB. SSLInformasinya ada dalam file format.pem yang berisi SSL informasi untuk cluster Aurora My DB. SQL Selama replikasi terenkripsi, cluster Aurora My SQL DB bertindak sebagai klien ke server database Saya. SQL Sertifikat dan kunci untuk Aurora SQL Klien saya ada dalam file dalam format.pem.

Anda dapat menyalin informasi dari file-file ini ke dalam `ssl_material` parameter dalam JSON muatan yang benar. Untuk mendukung replikasi terenkripsi, impor SSL informasi ini ke cluster Aurora My DB. SQL

JSONMuatan harus dalam format berikut.

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

Contoh

Contoh berikut mengimpor SSL informasi ke dalam Aurora My. Dalam file berformat .pem, kode konten biasanya lebih panjang dari kode konten yang ditunjukkan dalam contoh.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log SQL

Mengubah posisi log instans basis data sumber menjadi awal log biner berikutnya pada instans basis data sumber. Gunakan prosedur ini hanya jika Anda menerima kesalahan I/O 1236 replikasi pada replika baca.

### Sintaksis

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

### Parameter

*curr_master_log*

Indeks file log master saat ini. Misalnya, jika file saat ini bernama `mysql-bin-changelog.012345`, maka indeksnya adalah 12345. Untuk menentukan nama file log master saat ini, jalankan perintah `SHOW REPLICA STATUS` dan lihat kolom `Master_Log_File`.

### Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_next_master_log`.

> **Warning**
>
> Panggil `mysql.rds_next_master_log` hanya jika replikasi gagal setelah failover dari instans DB Multi-AZ yang merupakan sumber replikasi, dan kolom `Last_IO_Errno` dari laporan kesalahan I/O 1236 `SHOW REPLICA STATUS`.
> Memanggil `mysql.rds_next_master_log` dapat mengakibatkan hilangnya data di replika baca jika transaksi dalam instans sumber tidak ditulis ke log biner di disk sebelum peristiwa failover terjadi.

### Contoh

Asumsikan replikasi gagal pada replika . Menjalankan `SHOW REPLICA STATUS\G` pada replika baca akan menampilkan hasil berikut:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
```

```
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Kolom `Last_IO_Errno` menunjukkan bahwa instans menerima kesalahan I/O 1236. Kolom `Master_Log_File` menunjukkan bahwa nama file adalah `mysql-bin-changelog.012345`, yang berarti indeks file log adalah `12345`. Untuk mengatasi kesalahan, Anda dapat memanggil `mysql.rds_next_master_log` dengan parameter berikut:

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log ) SQL

Mengubah posisi log instans basis data sumber menjadi awal log biner berikutnya pada instans basis data sumber. Gunakan prosedur ini hanya jika Anda menerima kesalahan I/O 1236 replikasi pada replika baca.

Sintaksis

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

Parameter

*curr_source_log*

Indeks file log sumber saat ini. Misalnya, jika file saat ini bernama `mysql-bin-changelog.012345`, maka indeksnya adalah 12345. Untuk menentukan nama file log sumber saat ini, jalankan perintah `SHOW REPLICA STATUS` dan lihat kolom `Source_Log_File`.

Catatan penggunaan

Pengguna administratif harus menjalankan prosedur `mysql.rds_next_source_log`.

> **Warning**
>
> Panggil `mysql.rds_next_source_log` hanya jika replikasi gagal setelah failover dari instans DB Multi-AZ yang merupakan sumber replikasi, dan kolom `Last_IO_Errno` dari laporan kesalahan I/O 1236 `SHOW REPLICA STATUS`.
> Memanggil `mysql.rds_next_source_log` dapat mengakibatkan hilangnya data di replika baca jika transaksi dalam instans sumber tidak ditulis ke log biner di disk sebelum peristiwa failover terjadi. Anda dapat mengurangi kemungkinan hal ini terjadi dengan menyetel parameter instance sumber `sync_binlog` dan `innodb_support_xa` ke1, meskipun ini dapat mengurangi kinerja.

Contoh

Asumsikan replikasi gagal pada replika . Menjalankan `SHOW REPLICA STATUS\G` pada replika baca akan menampilkan hasil berikut:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
```

```
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Kolom `Last_IO_Errno` menunjukkan bahwa instans menerima kesalahan I/O 1236. Kolom `Source_Log_File` menunjukkan bahwa nama file adalah `mysql-bin-changelog.012345`, yang berarti indeks file log adalah `12345`. Untuk mengatasi kesalahan, Anda dapat memanggil `mysql.rds_next_source_log` dengan parameter berikut:

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

Menghapus sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien untuk SSL komunikasi dan replikasi terenkripsi. Informasi ini diimpor menggunakan mysql.rds_import_binlog_ssl_material.

Sintaksis

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master ) RDS SQL SQL

Mengkonfigurasi ulang instans My My SQL DB agar tidak lagi menjadi replika baca dari instance My running SQL eksternal Amazon. RDS

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB di .

Sintaksis

```
CALL mysql.rds_reset_external_master;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_reset_external_master`. Prosedur ini harus dijalankan pada instans SQL DB Saya untuk dihapus sebagai replika baca dari SQL instance Saya yang berjalan di luar AmazonRDS.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan SQL instans Saya yang berjalan di luar Amazon. RDS Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam cluster Aurora My DB bila memungkinkan. SQL Untuk informasi tentang mengelola replikasi di klaster Aurora SQL My DB, lihat. Menggunakan Replika Aurora

Untuk informasi selengkapnya tentang menggunakan replikasi untuk mengimpor data dari instance My SQL running external ke Aurora SQL My, lihat. Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner)

mysql.rds_reset_external_source ) SQL SQL

Mengkonfigurasi ulang instans My My SQL DB agar tidak lagi menjadi replika baca dari instance My running SQL eksternal Amazon. RDS

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB di .

Sintaksis

```
CALL mysql.rds_reset_external_source;
```

Catatan penggunaan

Pengguna administratif harus menjalankan prosedur `mysql.rds_reset_external_source`. Prosedur ini harus dijalankan pada instans SQL DB Saya untuk dihapus sebagai replika baca dari SQL instance Saya yang berjalan di luar AmazonRDS.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan SQL instans Saya yang berjalan di luar Amazon. RDS Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam cluster Aurora My DB bila memungkinkan. SQL Untuk informasi tentang mengelola replikasi di klaster Aurora SQL My DB, lihat. Menggunakan Replika Aurora

mysql.rds_set_binlog_source_ssl (Aurora Versi saya 3) SQL

Mengaktifkan `SOURCE_SSL` enkripsi untuk replikasi binlog. Untuk informasi selengkapnya, lihat pernyataan CHANGE REPLICATION SOURCE TO dalam SQL dokumentasi Saya.

Sintaks

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

Parameter

*mode*

Nilai yang menunjukkan apakah `SOURCE_SSL` enkripsi diaktifkan:

- `0`— `SOURCE_SSL` enkripsi dinonaktifkan. Default-nya adalah `0`.
- 1— `SOURCE_SSL` enkripsi diaktifkan. Anda dapat mengkonfigurasi enkripsi menggunakan SSL atauTLS.

Catatan penggunaan

Prosedur ini didukung untuk Aurora My SQL versi 3.06 dan yang lebih tinggi.

mysql.rds_set_external_master ) SQL SQL

Mengonfigurasi instance Aurora SQL My DB untuk menjadi replika baca dari instance SQL My running eksternal Amazon. RDS

Prosedur `mysql.rds_set_external_master` tidak digunakan lagi dan akan dihapus dalam rilis mendatang. Gunakan `mysql.rds_set_external_source` sebagai gantinya.

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB di .

Sintaksis

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameter

*host_name*

Nama host atau alamat IP dari SQL instans Saya berjalan eksternal ke Amazon RDS untuk menjadi instance database sumber.

*host_port*

Port yang digunakan oleh SQL instans Saya yang berjalan eksternal RDS ke Amazon untuk dikonfigurasi sebagai instance database sumber. Jika konfigurasi jaringan Anda menyertakan replikasi port Secure Shell (SSH) yang mengonversi nomor port, tentukan nomor port yang diekspos oleh. SSH

*replication_user_name*

ID pengguna dengan `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada SQL instans Saya yang berjalan di luar AmazonRDS. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

*replication_user_password*

Kata sandi ID pengguna yang ditentukan dalam `replication_user_name`.

*mysql_binary_log_file_name*

Nama log biner pada instans basis data sumber yang berisi informasi replikasi.

*mysql_binary_log_file_location*

Lokasi di log biner `mysql_binary_log_file_name` tempat replikasi mulai membaca informasi replikasi.

Anda dapat menentukan nama dan lokasi file binlog dengan menjalankan `SHOW MASTER STATUS` pada instans basis data sumber.

*ssl_encryption*

Nilai yang menentukan apakah enkripsi Secure Socket Layer (SSL) digunakan pada koneksi replikasi. 1 menentukan untuk menggunakan SSL enkripsi, 0 menentukan untuk tidak menggunakan enkripsi. Default-nya adalah 0.

> **Note**
>
> Opsi `MASTER_SSL_VERIFY_SERVER_CERT` tidak didukung. Opsi ini diatur ke 0, yang berarti koneksi dienkripsi, tetapi sertifikat tidak diverifikasi.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_set_external_master`. Prosedur ini harus dijalankan pada instans SQL DB Saya untuk dikonfigurasi sebagai replika baca SQL instance Saya yang berjalan di luar AmazonRDS.

Sebelum menjalankan`mysql.rds_set_external_master`, Anda harus mengonfigurasi instance My SQL running external to Amazon RDS untuk menjadi instance database sumber. Untuk menyambung ke SQL instans Saya yang berjalan eksternal ke AmazonRDS, Anda harus menentukan `replication_user_name` dan `replication_user_password` nilai yang

menunjukkan pengguna replikasi yang memiliki `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada instance eksternal My. SQL

Untuk mengkonfigurasi instance eksternal My SQL sebagai instance database sumber

1. Menggunakan SQL klien Saya pilihan Anda, sambungkan ke instance eksternal My SQL dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   SQL5.7 saya

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   Saya SQL 8.0

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
    'password';
   ```

   > **Note**
   >
   > Tetapkan kata sandi selain penggugah (prompt) yang ditampilkan di sini sebagai praktik terbaik keamanan.

2. Pada contoh eksternal MySQL, berikan `REPLICATION CLIENT` dan hak `REPLICATION SLAVE` istimewa kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna ‘repl_user’ domain Anda.

   SQL5.7 saya

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   Saya SQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

Untuk menggunakan replikasi terenkripsi, konfigurasikan instance database sumber untuk menggunakan koneksi. SSL  Selain itu, impor sertifikat otoritas sertifikat, sertifikat

klien, dan kunci klien ke dalam instans DB atau klaster DB menggunakan prosedur mysql.rds_import_binlog_ssl_material.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan SQL instans Saya yang berjalan di luar Amazon. RDS Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam cluster Aurora My DB bila memungkinkan. SQL Untuk informasi tentang mengelola replikasi di klaster Aurora SQL My DB, lihat. Menggunakan Replika Aurora

Setelah menelepon `mysql.rds_set_external_master` untuk mengonfigurasi instans Amazon RDS DB sebagai replika baca, Anda dapat memanggil mysql.rds_start_replication replika baca untuk memulai proses replikasi. Anda dapat memanggil mysql.rds_reset_external_master ) RDS SQL SQL untuk menghapus konfigurasi replika baca.

Ketika `mysql.rds_set_external_master` dipanggil, Amazon RDS mencatat waktu, pengguna, dan tindakan `set master` dalam `mysql.rds_history` dan `mysql.rds_replication_status` tabel.

Contoh

Ketika dijalankan pada instans SQL DB Saya, contoh berikut mengonfigurasi instans DB menjadi replika baca dari instance My SQL running eksternal ke Amazon. RDS

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_source ) SQL

Mengonfigurasi instance Aurora SQL My DB untuk menjadi replika baca dari instance SQL My running eksternal Amazon. RDS

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB di .

Sintaksis

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameter

*host_name*

Nama host atau alamat IP dari SQL instans Saya berjalan eksternal ke Amazon RDS untuk menjadi instance database sumber.

*host_port*

Port yang digunakan oleh SQL instans Saya yang berjalan eksternal RDS ke Amazon untuk dikonfigurasi sebagai instance database sumber. Jika konfigurasi jaringan Anda menyertakan replikasi port Secure Shell (SSH) yang mengonversi nomor port, tentukan nomor port yang diekspos oleh. SSH

*replication_user_name*

ID pengguna dengan `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada SQL instans Saya yang berjalan di luar AmazonRDS. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

*replication_user_password*

Kata sandi ID pengguna yang ditentukan dalam `replication_user_name`.

*mysql_binary_log_file_name*

Nama log biner pada instans basis data sumber yang berisi informasi replikasi.

*mysql_binary_log_file_location*

Lokasi di log biner `mysql_binary_log_file_name` tempat replikasi mulai membaca informasi replikasi.

Anda dapat menentukan nama dan lokasi file binlog dengan menjalankan `SHOW MASTER STATUS` pada instans basis data sumber.

*ssl_encryption*

Nilai yang menentukan apakah enkripsi Secure Socket Layer (SSL) digunakan pada koneksi replikasi. 1 menentukan untuk menggunakan SSL enkripsi, 0 menentukan untuk tidak menggunakan enkripsi. Default-nya adalah 0.

> **Note**
>
> Anda harus telah mengimpor SSL sertifikat khusus menggunakan mysql.rds_import_binlog_ssl_material untuk mengaktifkan opsi ini. Jika Anda belum mengimpor SSL sertifikat kustom, maka atur parameter ini ke 0 dan gunakan mysql.rds_set_binlog_source_ssl (Aurora Versi saya 3) SQL SSL untuk mengaktifkan replikasi log biner.
> Opsi `SOURCE_SSL_VERIFY_SERVER_CERT` tidak didukung. Opsi ini diatur ke 0, yang berarti koneksi dienkripsi, tetapi sertifikat tidak diverifikasi.

Catatan penggunaan

Pengguna administratif harus menjalankan prosedur `mysql.rds_set_external_source`. Prosedur ini harus dijalankan pada instans Aurora My SQL RDS DB untuk dikonfigurasi sebagai replika baca SQL instance Saya yang berjalan di luar Amazon. RDS

Sebelum menjalankan`mysql.rds_set_external_source`, Anda harus mengonfigurasi instance My SQL running external to Amazon RDS untuk menjadi instance database sumber. Untuk menyambung ke SQL instans Saya yang berjalan eksternal ke AmazonRDS, Anda harus menentukan `replication_user_name` dan `replication_user_password` nilai yang menunjukkan pengguna replikasi yang memiliki `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada instance eksternal My. SQL

Untuk mengkonfigurasi instance eksternal My SQL sebagai instance database sumber

1. Menggunakan SQL klien Saya pilihan Anda, sambungkan ke instance eksternal My SQL dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut sebuah contoh.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   > **Note**
   >
   > Tentukan kata sandi selain prompt yang ditampilkan di sini sebagai praktik terbaik keamanan.

2. Pada contoh eksternal MySQL, berikan `REPLICATION CLIENT` dan hak `REPLICATION SLAVE` istimewa kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna ‘repl_user’ domain Anda.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

Untuk menggunakan replikasi terenkripsi, konfigurasikan instance database sumber untuk menggunakan koneksi. SSL Selain itu, impor sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien ke dalam instans DB atau klaster DB menggunakan prosedur mysql.rds_import_binlog_ssl_material.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan SQL instans Saya yang berjalan di luar Amazon. RDS Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam cluster Aurora My DB bila memungkinkan. SQL Untuk informasi tentang mengelola replikasi di klaster Aurora SQL My DB, lihat. Menggunakan Replika Aurora

Setelah menelepon `mysql.rds_set_external_source` untuk mengonfigurasi instance Aurora My SQL RDS DB sebagai replika baca, Anda dapat memanggil mysql.rds_start_replication replika baca untuk memulai proses replikasi. Anda dapat memanggil mysql.rds_reset_external_source ) SQL SQL untuk menghapus konfigurasi replika baca.

Ketika `mysql.rds_set_external_source` dipanggil, Amazon RDS mencatat waktu, pengguna, dan tindakan `set master` dalam `mysql.rds_history` dan `mysql.rds_replication_status` tabel.

Contoh

Ketika dijalankan pada instance Aurora My SQL RDS DB, contoh berikut mengonfigurasi instans DB menjadi replika baca dari instance My running SQL external to Amazon. RDS

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_master_with_auto_position (Aurora Versi saya 2) SQL

Mengonfigurasi instance utama Aurora SQL My untuk menerima replikasi masuk dari instance Saya eksternal. SQL Prosedur ini juga mengonfigurasi replikasi berdasarkan pengidentifikasi transaksi global (). GTIDs

Prosedur ini tidak mengonfigurasi replikasi yang tertunda, karena Aurora SQL My tidak mendukung replikasi yang tertunda.

Sintaks

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

Parameter

*host_name*

Nama host atau alamat IP dari SQL instans Saya berjalan eksternal ke Aurora untuk menjadi sumber replikasi.

*host_port*

Port yang digunakan oleh SQL instans Saya berjalan eksternal ke Aurora untuk dikonfigurasi sebagai sumber replikasi. Jika konfigurasi jaringan Anda menyertakan replikasi port Secure Shell (SSH) yang mengonversi nomor port, tentukan nomor port yang diekspos oleh. SSH

*replication_user_name*

ID pengguna dengan `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada SQL instance Saya yang berjalan di luar Aurora. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

*replication_user_password*

Kata sandi ID pengguna yang ditentukan dalam `replication_user_name`.

*ssl_encryption*

Opsi ini belum diterapkan. Default-nya adalah 0.

Catatan penggunaan

Untuk cluster Aurora My SQL DB, Anda memanggil prosedur tersimpan ini saat terhubung ke instance utama.

Pengguna utama harus menjalankan prosedur `mysql.rds_set_external_master_with_auto_position`. Pengguna master menjalankan prosedur ini pada instance utama cluster Aurora My SQL DB yang bertindak sebagai target replikasi. Ini bisa menjadi target replikasi instans My SQL DB eksternal atau cluster Aurora SQL My DB.

Prosedur ini didukung untuk Aurora My SQL versi 2. Untuk Aurora My SQL versi 3, gunakan prosedur mysql.rds_set_external_source_with_auto_position (Aurora Versi saya 3) SQL sebagai gantinya.

Sebelum Anda menjalankan`mysql.rds_set_external_master_with_auto_position`, konfigurasikan instance My SQL DB eksternal menjadi sumber replikasi. Untuk terhubung ke SQL instance Saya eksternal, tentukan nilai untuk `replication_user_name`

dan`replication_user_password`. Nilai-nilai ini harus menunjukkan pengguna replikasi yang memiliki `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada instance Saya SQL eksternal.

Untuk mengonfigurasi SQL instance Saya eksternal sebagai sumber replikasi

1. Menggunakan SQL klien Saya pilihan Anda, sambungkan ke SQL instans Saya eksternal dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. Pada SQL instans Saya eksternal, berikan `REPLICATION CLIENT` dan `REPLICATION SLAVE` hak istimewa kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna `'repl_user'` domain Anda.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

Saat Anda meneleponmysql.rds_set_external_master_with_auto_position, Amazon RDS mencatat informasi tertentu. Informasi ini adalah waktu, pengguna, dan tindakan `"set master"` di tabel `mysql.rds_history` dan `mysql.rds_replication_status`.

Untuk melewati transaksi GTID berbasis tertentu yang diketahui menyebabkan masalah, Anda dapat menggunakan prosedur mysql.rds_skip_transaction_with_gtid (Aurora Versi saya 2 dan 3) SQL tersimpan. Untuk informasi lebih lanjut tentang bekerja dengan replikasi GTID berbasis, lihatMenggunakan replikasi GTID berbasis.

Contoh

Ketika dijalankan pada instance utama Aurora, contoh berikut mengkonfigurasi cluster Aurora untuk bertindak sebagai replika baca dari instance My running eksternal ke Aurora. SQL

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_external_source_with_auto_position (Aurora Versi saya 3) SQL

Mengonfigurasi instance utama Aurora SQL My untuk menerima replikasi masuk dari instance Saya eksternal. SQL Prosedur ini juga mengonfigurasi replikasi berdasarkan pengidentifikasi transaksi global (). GTIDs

Sintaks

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

Parameter

*host_name*

Nama host atau alamat IP dari SQL instans Saya berjalan eksternal ke Aurora untuk menjadi sumber replikasi.

*host_port*

Port yang digunakan oleh SQL instans Saya berjalan eksternal ke Aurora untuk dikonfigurasi sebagai sumber replikasi. Jika konfigurasi jaringan Anda menyertakan replikasi port Secure Shell (SSH) yang mengonversi nomor port, tentukan nomor port yang diekspos oleh. SSH

*replication_user_name*

ID pengguna dengan `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada SQL instance Saya yang berjalan di luar Aurora. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

*replication_user_password*

Kata sandi ID pengguna yang ditentukan dalam `replication_user_name`.

*ssl_encryption*

Opsi ini belum diterapkan. Default-nya adalah 0.

> **Note**
>
> Gunakan mysql.rds_set_binlog_source_ssl (Aurora Versi saya 3) SQL SSL untuk mengaktifkan replikasi log biner.

Catatan penggunaan

Untuk cluster Aurora My SQL DB, Anda memanggil prosedur tersimpan ini saat terhubung ke instance utama.

Pengguna administratif harus menjalankan prosedur `mysql.rds_set_external_source_with_auto_position`. Pengguna administratif menjalankan prosedur ini pada contoh utama cluster Aurora My SQL DB yang bertindak sebagai target replikasi. Ini bisa menjadi target replikasi instans My SQL DB eksternal atau cluster Aurora SQL My DB.

Prosedur ini didukung untuk Aurora My SQL versi 3. Prosedur ini tidak mengonfigurasi replikasi yang tertunda, karena Aurora SQL My tidak mendukung replikasi yang tertunda.

Sebelum Anda menjalankan`mysql.rds_set_external_source_with_auto_position`, konfigurasikan instance My SQL DB eksternal menjadi sumber replikasi. Untuk terhubung ke SQL instance Saya eksternal, tentukan nilai untuk `replication_user_name` dan`replication_user_password`. Nilai-nilai ini harus menunjukkan pengguna replikasi yang memiliki `REPLICATION CLIENT` dan `REPLICATION SLAVE` izin pada instance Saya SQL eksternal.

Untuk mengonfigurasi SQL instance Saya eksternal sebagai sumber replikasi

1. Menggunakan SQL klien Saya pilihan Anda, sambungkan ke SQL instans Saya eksternal dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. Pada SQL instans Saya eksternal, berikan `REPLICATION CLIENT` dan `REPLICATION SLAVE` hak istimewa kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna `'repl_user'` domain Anda.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   ```

```
IDENTIFIED BY 'SomePassW0rd'
```

Saat Anda meneleponmysql.rds_set_external_source_with_auto_position, Amazon RDS mencatat informasi tertentu. Informasi ini adalah waktu, pengguna, dan tindakan "set master" di tabel mysql.rds_history dan mysql.rds_replication_status.

Untuk melewati transaksi GTID berbasis tertentu yang diketahui menyebabkan masalah, Anda dapat menggunakan prosedur mysql.rds_skip_transaction_with_gtid (Aurora Versi saya 2 dan 3) SQL tersimpan. Untuk informasi lebih lanjut tentang bekerja dengan replikasi GTID berbasis, lihatMenggunakan replikasi GTID berbasis.

Contoh

Ketika dijalankan pada instance utama Aurora, contoh berikut mengkonfigurasi cluster Aurora untuk bertindak sebagai replika baca dari instance My running eksternal ke Aurora. SQL

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position ) SQL

Menetapkan mode replikasi untuk didasarkan pada posisi file log biner atau pada pengidentifikasi transaksi global ()GTIDs.

Sintaks

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

Parameter

*auto_position_mode*

Nilai yang menunjukkan apakah akan menggunakan replikasi posisi file log atau replikasi GTID berbasis:

- `0` – Gunakan metode replikasi berdasarkan posisi file log biner. Default-nya adalah `0`.
- 1— Gunakan metode replikasi GTID berbasis.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_set_master_auto_position`.

Prosedur ini didukung untuk Aurora My SQL versi 2.

mysql.rds_set_read_only (Aurora Versi saya 3) SQL

Menghidupkan atau menonaktifkan `read_only` mode secara global untuk instans DB.

Sintaks

```
CALL mysql.rds_set_read_only(mode);
```

Parameter

*mode*

Nilai yang menunjukkan apakah `read_only` mode aktif atau tidak aktif secara global untuk instans DB:

- `0`—OFF. Defaultnya adalah`0`.
- 1 – ON

Catatan penggunaan

Prosedur yang `mysql.rds_set_read_only` disimpan hanya memodifikasi `read_only` parameter. `innodb_read_only`Parameter tidak dapat diubah pada instans DB pembaca.

Perubahan `read_only` parameter tidak bertahan saat reboot. Untuk membuat perubahan permanen`read_only`, Anda harus menggunakan parameter cluster `read_only` DB.

Prosedur ini didukung untuk Aurora My SQL versi 3.06 dan yang lebih tinggi.

mysql.rds_set_session_binlog_format (Aurora Versi saya 2) SQL

Mengatur format log biner untuk sesi saat ini.

Sintaksis

```
CALL mysql.rds_set_session_binlog_format(format);
```

Parameter

*format*

Nilai yang menunjukkan format log biner untuk sesi saat ini:

- STATEMENT— Sumber replikasi menulis peristiwa ke log biner berdasarkan SQL pernyataan.
- ROW – Sumber replikasi menulis peristiwa ke log biner yang menunjukkan perubahan pada baris tabel individual.
- MIXEDLogging umumnya didasarkan pada SQL pernyataan, tetapi beralih ke baris dalam kondisi tertentu. Untuk informasi selengkapnya, lihat Format Pencatatan Biner Campuran di SQL dokumentasi Saya.

Catatan penggunaan

Untuk cluster Aurora My SQL DB, Anda memanggil prosedur tersimpan ini saat terhubung ke instance utama.

Untuk menggunakan prosedur tersimpan ini, Anda harus memiliki pencatatan log biner yang dikonfigurasi untuk sesi saat ini.

Untuk Aurora, prosedur ini didukung untuk Aurora My SQL versi 2.12 dan versi My 5.7 yang lebih tinggi. SQL

mysql.rds_set_source_auto_position ) SQL SQL

Menetapkan mode replikasi untuk didasarkan pada posisi file log biner atau pada pengidentifikasi transaksi global ()GTIDs.

Sintaks

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameter

*auto_position_mode*

Nilai yang menunjukkan apakah akan menggunakan replikasi posisi file log atau replikasi GTID berbasis:

- `0` – Gunakan metode replikasi berdasarkan posisi file log biner. Default-nya adalah `0`.
- 1— Gunakan metode replikasi GTID berbasis.

Catatan penggunaan

Untuk cluster Aurora My SQL DB, Anda memanggil prosedur tersimpan ini saat terhubung ke instance utama.

Pengguna administratif harus menjalankan prosedur `mysql.rds_set_source_auto_position`.

mysql.rds_skip_repl_error

Melewatkan dan menghapus kesalahan replikasi pada replika baca My SQL DB.

Sintaks

```
CALL mysql.rds_skip_repl_error;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_skip_repl_error` pada replika baca. Untuk informasi selengkapnya tentang prosedur ini, lihat Melewati kesalahan replikasi saat ini.

Untuk menentukan apakah ada kesalahan, jalankan SQL `SHOW REPLICA STATUS\G` perintah Saya. Jika kesalahan replikasi tidak parah, Anda dapat menjalankan `mysql.rds_skip_repl_error` untuk melewati kesalahan tersebut. Jika ada beberapa kesalahan, `mysql.rds_skip_repl_error` akan menghapus kesalahan pertama, lalu memberi peringatan bahwa ada kesalahan lain. Anda kemudian dapat menggunakan `SHOW REPLICA STATUS\G` untuk menentukan tindakan yang benar untuk kesalahan berikutnya. Untuk informasi tentang nilai yang dikembalikan, lihat SHOWREPLICASTATUSpernyataan dalam SQL dokumentasi Saya.

Untuk informasi lebih lanjut tentang mengatasi kesalahan replikasi dengan Aurora My, lihat. SQL Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL

Kesalahan replikasi terhenti

Ketika memanggil prosedur `mysql.rds_skip_repl_error`, Anda mungkin menerima pesan kesalahan yang menyatakan bahwa replika tidak berfungsi atau dinonaktifkan.

Pesan kesalahan ini muncul jika Anda menjalankan prosedur pada instans primer, bukan replika baca. Anda harus menjalankan prosedur ini pada replika baca agar prosedur berfungsi.

Pesan kesalahan ini mungkin juga muncul jika Anda menjalankan prosedur pada replika baca, tetapi replikasi tidak berhasil dimulai ulang.

Jika Anda perlu melewati sejumlah besar kesalahan, lag replikasi dapat meningkat hingga melampaui periode retensi default untuk file log biner (binlog). Dalam hal ini, Anda mungkin mengalami kesalahan fatal karena file binlog dibersihkan sebelum diputar ulang pada replika baca. Penghapusan ini menyebabkan replikasi berhenti, dan Anda tidak dapat lagi memanggil perintah `mysql.rds_skip_repl_error` untuk melewati kesalahan replikasi.

Anda dapat memitigasi masalah ini dengan meningkatkan jumlah jam retensi file binlog tersebut pada instans basis data sumber Anda. Setelah meningkatkan waktu retensi binlog, Anda dapat memulai ulang replikasi dan memanggil perintah `mysql.rds_skip_repl_error` sesuai kebutuhan.

Untuk mengatur waktu retensi binlog, gunakan prosedur mysql.rds_set_configuration dan tentukan parameter konfigurasi `'binlog retention hours'` bersama dengan jumlah jam untuk mempertahankan file binlog di klaster DB. Contoh berikut menetapkan periode penyimpanan file binlog menjadi 48 jam.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Memulai replikasi dari instance cluster. SQL

> **Note**
>
> Anda dapat menggunakan prosedur mysql.rds_start_replication_until (Aurora Versi saya 3) SQL atau mysql.rds_start_replication_until_gtid (Aurora Versi saya 3) SQL tersimpan untuk memulai replikasi dari instance My SQL DB dan menghentikan replikasi di lokasi file log biner yang ditentukan.

Sintaks

```
CALL mysql.rds_start_replication;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_start_replication`.

Untuk mengimpor data dari instance SQL eksternal Saya ke AmazonRDS, panggil `mysql.rds_start_replication` replika baca untuk memulai proses replikasi setelah Anda memanggil mysql.rds_set_external_master ) SQL SQL atau mysql.rds_set_external_source ) SQL membuat konfigurasi replikasi. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Untuk mengekspor data ke instance SQL eksternal Saya ke AmazonRDS, panggil `mysql.rds_start_replication` dan `mysql.rds_stop_replication` pada replika baca untuk mengontrol beberapa tindakan replikasi, seperti membersihkan log biner. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Anda juga dapat memanggil `mysql.rds_start_replication` pada replika baca untuk memulai kembali proses replikasi apa pun yang sebelumnya Anda hentikan dengan memanggil `mysql.rds_stop_replication`. Untuk informasi selengkapnya, lihat Kesalahan replikasi terhenti.

mysql.rds_start_replication_until (Aurora Versi saya 3) SQL

Memulai replikasi dari instance My SQL DB cluster dan menghentikan replikasi di lokasi file log biner yang ditentukan.

Sintaks

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

Parameter

*replication_log_file*

Nama log biner pada instans basis data sumber yang berisi informasi replikasi.

*replication_stop_point*

Lokasi di log biner `replication_log_file` tempat replikasi akan berhenti.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_start_replication_until`.

Prosedur ini didukung untuk Aurora My SQL versi 3.04 dan yang lebih tinggi.

Prosedur `mysql.rds_start_replication_until` tersimpan tidak didukung untuk replikasi terkelola, yang mencakup hal-hal berikut:

- Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS
- Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora

Nama file yang ditentukan untuk parameter `replication_log_file` harus cocok dengan nama file binlog instans basis data sumber.

Jika parameter `replication_stop_point` menentukan lokasi perhentian di masa lalu, replikasi akan segera dihentikan.

Contoh

Contoh berikut memulai replikasi dan mereplikasi perubahan hingga mencapai lokasi `120` di file log biner `mysql-bin-changelog.000777`.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_stop_replication

Menghentikan replikasi dari instance My SQL DB.

### Sintaks

```
CALL mysql.rds_stop_replication;
```

### Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_stop_replication`.

Jika Anda mengonfigurasi replikasi untuk mengimpor data dari instance My SQL running eksternal ke AmazonRDS, Anda memanggil `mysql.rds_stop_replication` replika baca untuk menghentikan proses replikasi setelah impor selesai. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Jika Anda mengonfigurasi replikasi untuk mengekspor data ke instance SQL eksternal Saya ke AmazonRDS, Anda memanggil `mysql.rds_start_replication` dan `mysql.rds_stop_replication` menggunakan replika baca untuk mengontrol beberapa tindakan replikasi, seperti membersihkan log biner. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya (replikasi log biner).

Prosedur `mysql.rds_stop_replication` tersimpan tidak didukung untuk replikasi terkelola, yang mencakup hal-hal berikut:

- Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS
- Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora

## Mengakhiri sesi atau kueri

Prosedur tersimpan berikut mengakhiri sesi atau kueri.

Topik

- mysql.rds_kill
- mysql.rds_kill_query

### mysql.rds_kill

Mengakhiri koneksi ke server MySQL.

Sintaksis

```
CALL mysql.rds_kill(processID);
```

Parameter

*processID*

Identitas utas koneksi akan diakhiri.

Catatan penggunaan

Setiap koneksi ke server MySQL berjalan di utas terpisah. Untuk mengakhiri koneksi, gunakan prosedur `mysql.rds_kill` dan teruskan ID utas koneksi tersebut. Untuk mendapatkan ID utas, gunakan perintah SHOW PROCESSLIST MySQL.

Contoh

Contoh berikut mengakhiri koneksi dengan ID utas 4243:

```
CALL mysql.rds_kill(4243);
```

### mysql.rds_kill_query

Mengakhiri kueri yang berjalan pada server MySQL.

### Sintaksis

```
CALL mysql.rds_kill_query(processID);
```

### Parameter

*processID*

Identitas proses atau utas yang menjalankan kueri yang akan diakhiri.

### Catatan penggunaan

Untuk mengakhiri kueri yang berjalan pada server MySQL, gunakan prosedur `mysql_rds_kill_query` dan teruskan ID koneksi utas yang menjalankan kueri. Lalu, prosedur akan mengakhiri koneksi.

Untuk mendapatkan ID, lakukan kueri pada tabel INFORMATION_SCHEMA PROCESSLIST MySQL atau gunakan perintah SHOW PROCESSLIST MySQL. Nilai dalam kolom ID dari `SHOW PROCESSLIST` atau `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST` adalah *processID*.

### Contoh

Contoh berikut mengakhiri kueri dengan ID utas kueri 230040:

```
CALL mysql.rds_kill_query(230040);
```

## Mereplikasi transaksi menggunakan GTIDs

Prosedur tersimpan berikut mengontrol bagaimana transaksi direplikasi menggunakan pengidentifikasi transaksi global (GTIDs) dengan Aurora My. SQL Untuk mempelajari cara menggunakan replikasi berdasarkan GTIDs Aurora SQL My, lihat. Menggunakan replikasi GTID berbasis

Topik

- mysql.rds_assign_gtids_to_anonymous_transactions (Aurora Versi saya 3) SQL
- mysql.rds_gtid_purged (Aurora Versi saya 3) SQL
- mysql.rds_skip_transaction_with_gtid (Aurora Versi saya 2 dan 3) SQL
- mysql.rds_start_replication_until_gtid (Aurora Versi saya 3) SQL

### mysql.rds_assign_gtids_to_anonymous_transactions (Aurora Versi saya 3) SQL

Mengonfigurasi opsi `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` dari pernyataan `CHANGE REPLICATION SOURCE TO`. Itu membuat saluran replikasi menetapkan transaksi yang GTID direplikasi yang tidak memilikinya. Dengan begitu, Anda dapat melakukan replikasi log biner dari sumber yang tidak menggunakan replikasi GTID berbasis ke replika yang melakukannya. Untuk informasi lebih lanjut, lihat CHANGEREPLICATIONSOURCEPernyataan dan Replikasi Dari Sumber Tanpa GTIDs Replika Dengan GTIDs di Manual SQL Referensi Saya.

Sintaks

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

Parameter

*`gtid_option`*

Nilai string. Nilai yang diizinkan adalah `OFFLOCAL`,, atau ditentukanUUID.

Catatan penggunaan

Prosedur ini memiliki efek yang sama seperti mengeluarkan pernyataan `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = gtid_option` di komunitas MySQL.

GTIDharus diputar `ON` *`gtid_option`* untuk disetel ke `LOCAL` atau spesifikUUID.

Defaultnya adalah `OFF`, yang artinya fitur tersebut tidak digunakan.

`LOCAL`menetapkan GTID termasuk replika itu sendiri UUID (`server_uuid`pengaturan).

Melewati parameter yang UUID menetapkan a GTID yang menyertakan yang ditentukanUUID, seperti `server_uuid` pengaturan untuk server sumber replikasi.

Contoh

Untuk menonaktifkan fitur ini:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

Untuk menggunakan replika itu sendiriUUID:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+------------------------------------------------------------+
| Message  |
+------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+------------------------------------------------------------+
1 row in set (0.07 sec)
```

Untuk menggunakan yang ditentukanUUID:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------------
+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------------
+
```

```
1 row in set (0.07 sec)
```

mysql.rds_gtid_purged (Aurora Versi saya 3) SQL

Menetapkan nilai global variabel sistem `gtid_purged` ke set identifier transaksi global (GTID) yang diberikan. Variabel `gtid_purged` sistem adalah GTID satu set yang terdiri GTIDs dari semua transaksi yang telah dilakukan di server, tetapi tidak ada dalam file log biner apa pun di server.

Untuk memungkinkan kompatibilitas dengan My SQL 8.0, ada dua cara untuk mengatur nilai`gtid_purged`:

- Ganti nilai `gtid_purged` dengan GTID set yang Anda tentukan.
- Tambahkan GTID set yang Anda tentukan ke GTID set yang `gtid_purged` sudah berisi.

Sintaks

Untuk mengganti nilai `gtid_purged` dengan GTID set yang Anda tentukan:

```
CALL mysql.rds_gtid_purged (gtid_set);
```

Untuk menambahkan nilai `gtid_purged` ke set yang Anda GTID tentukan:

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

Parameter

*`gtid_set`*

Nilai *`gtid_set`* harus menjadi superset dari nilai saat ini`gtid_purged`, dan tidak dapat berpotongan dengan. `gtid_subtract(gtid_executed,gtid_purged)` Artinya, GTID set baru harus menyertakan apa pun GTIDs yang sudah ada`gtid_purged`, dan tidak dapat menyertakan apa pun GTIDs `gtid_executed` yang belum dibersihkan. *`gtid_set`*Parameter juga tidak dapat menyertakan apa pun GTIDs yang ada di `gtid_owned` set global, GTIDs untuk transaksi yang saat ini sedang diproses di server.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_gtid_purged`.

Prosedur ini didukung untuk Aurora My SQL versi 3.04 dan yang lebih tinggi.

Contoh

Contoh berikut menetapkan GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23` ke variabel `gtid_purged` global.

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_transaction_with_gtid (Aurora Versi saya 2 dan 3) SQL

Melewatkan replikasi transaksi dengan identifier transaksi global yang ditentukan (GTID) pada instance utama Aurora.

Anda dapat menggunakan prosedur ini untuk pemulihan bencana ketika GTID transaksi tertentu diketahui menyebabkan masalah. Gunakan prosedur tersimpan ini untuk melewati transaksi bermasalah. Contoh transaksi bermasalah mencakup transaksi yang menonaktifkan replikasi, menghapus data penting, atau menyebabkan instans DB menjadi tidak tersedia.

Sintaksis

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

Parameter

*`gtid_to_skip`*

Transaksi replikasi untuk dilewati. GTID

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_skip_transaction_with_gtid`.

Prosedur ini didukung untuk Aurora My SQL versi 2 dan 3.

Contoh

Contoh berikut melewatkan replikasi transaksi dengan. GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid (Aurora Versi saya 3) SQL

Memulai replikasi dari instance My SQL DB cluster dan menghentikan replikasi segera setelah pengidentifikasi transaksi global yang ditentukan (). GTID

Sintaks

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

Parameter

*gtid*

GTIDSetelah itu replikasi berhenti.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_start_replication_until_gtid`.

Prosedur ini didukung untuk Aurora My SQL versi 3.04 dan yang lebih tinggi.

Prosedur `mysql.rds_start_replication_until_gtid` tersimpan tidak didukung untuk replikasi terkelola, yang mencakup hal-hal berikut:

- Mereplikasi kluster DB MySQL Amazon Aurora Wilayah AWS
- Memigrasi data dari instans RDS for My SQL DB ke cluster Amazon Aurora SQL My DB dengan menggunakan replika baca Aurora

Saat parameter `gtid` menentukan transaksi yang telah dijalankan oleh replika, replikasi akan segera dihentikan.

Contoh

Contoh berikut memulai replikasi dan mereplikasi perubahan hingga mencapai. GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

## Memutar log kueri

Prosedur tersimpan berikut memutar SQL log Saya ke tabel cadangan. Untuk informasi selengkapnya, lihat File log basis data Aurora MySQL.

Topik

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

### mysql.rds_rotate_general_log

Merotasi tabel `mysql.general_log` ke tabel cadangan.

Sintaks

```
CALL mysql.rds_rotate_general_log;
```

Catatan penggunaan

Anda bisa merotasi tabel `mysql.general_log` ke tabel cadangan dengan memanggil prosedur `mysql.rds_rotate_general_log`. Saat tabel log dirotasi, tabel log saat ini disalin ke tabel log cadangan dan entri di tabel log saat ini dihapus. Jika sudah ada, tabel log cadangan akan dihapus sebelum tabel log saat ini disalin ke cadangan. Anda dapat meminta tabel log cadangan jika diperlukan. Tabel log cadangan untuk tabel `mysql.general_log` bernama `mysql.general_log_backup`.

Anda dapat menjalankan prosedur ini hanya ketika parameter `log_output` diatur ke `TABLE`.

### mysql.rds_rotate_slow_log

Merotasi tabel `mysql.slow_log` ke tabel cadangan.

Sintaks

```
CALL mysql.rds_rotate_slow_log;
```

Catatan penggunaan

Anda bisa merotasi tabel `mysql.slow_log` ke tabel cadangan dengan memanggil prosedur `mysql.rds_rotate_slow_log`. Saat tabel log dirotasi, tabel log saat ini disalin ke tabel log

cadangan dan entri di tabel log saat ini dihapus. Jika sudah ada, tabel log cadangan akan dihapus sebelum tabel log saat ini disalin ke cadangan.

Anda dapat meminta tabel log cadangan jika diperlukan. Tabel log cadangan untuk tabel `mysql.slow_log` bernama `mysql.slow_log_backup`.

## Mengatur dan menampilkan konfigurasi log biner

Prosedur tersimpan berikut mengatur dan menampilkan parameter konfigurasi, seperti untuk retensi file log biner.

Topik

- mysql.rds_set_configuration
- mysql.rds_show_configuration

### mysql.rds_set_configuration

Menentukan jumlah jam untuk mempertahankan log biner atau jumlah detik untuk menunda replikasi.

Sintaks

```
CALL mysql.rds_set_configuration(name,value);
```

Parameter

*name*

Nama parameter konfigurasi yang akan diatur.

*value*

Nilai parameter konfigurasi.

Catatan penggunaan

Prosedur `mysql.rds_set_configuration` mendukung parameter konfigurasi berikut:

- binlog retention hours

Parameter konfigurasi disimpan secara permanen dan bertahan dari boot ulang atau failover instans DB apa pun.

binlog retention hours

Parameter `binlog retention hours` digunakan untuk menentukan jumlah jam untuk mempertahankan file log biner. Amazon Aurora biasanya membersihkan log biner sesegera mungkin, tetapi log biner mungkin masih diperlukan untuk replikasi dengan database Saya di SQL luar Aurora.

Nilai default `binlog retention hours` adalah `NULL`. Untuk Aurora MySQL, `NULL` berarti log biner dibersihkan dengan malas. Aurora Log SQL biner saya mungkin tetap berada di sistem untuk jangka waktu tertentu, yang biasanya tidak lebih dari satu hari.

Untuk menentukan jumlah jam guna mempertahankan log biner pada klaster DB, gunakan prosedur tersimpan `mysql.rds_set_configuration` dan tentukan periode dengan waktu yang cukup untuk terjadinya proses replikasi, seperti yang diperlihatkan dalam contoh berikut.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> Anda tidak dapat menggunakan nilai `0` untuk `binlog retention hours`.

Untuk Aurora SQL Versi saya 2.11.0 dan yang lebih tinggi dan cluster DB versi 3, `binlog retention hours` nilai maksimumnya adalah 2160 (90 hari).

Setelah Anda mengatur periode retensi, pantau penggunaan penyimpanan untuk instans DB guna memastikan bahwa log biner yang dipertahankan tidak memakan terlalu banyak ruang penyimpanan.

mysql.rds_show_configuration

Jumlah jam untuk mempertahankan log biner.

Sintaks

```
CALL mysql.rds_show_configuration;
```

Catatan penggunaan

Untuk memverifikasi jumlah jam Amazon RDS menyimpan log biner, gunakan prosedur yang `mysql.rds_show_configuration` disimpan.

Contoh

Contoh berikut menampilkan periode retensi:

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

## Aurora My SQL —tabel information_schema spesifik

Aurora My SQL memiliki `information_schema` tabel tertentu yang khusus untuk Aurora.

### information_schema.aurora_global_db_instance_status

Tabel `information_schema.aurora_global_db_instance_status` berisi informasi tentang status semua instans DB dalam klaster DB primer dan sekunder di basis data global. Tabel berikut menunjukkan kolom yang dapat Anda gunakan. Kolom yang tersisa hanya ditujukan untuk penggunaan internal Aurora.

> **Note**
>
> Tabel skema informasi ini hanya tersedia dengan Aurora SQL My versi 3.04.0 dan database global yang lebih tinggi.

| Kolom | Jenis data | Deskripsi |
|---|---|---|
| SERVER_ID | varchar(100) | Pengidentifikasi instans DB. |
| SESSION_ID | varchar(100) | Pengidentifikasi unik untuk sesi saat ini. Nilai `MASTER_SESSION_ID` mengidentifikasi instans DB Penulis (primer). |
| AWS_REGION | varchar(100) | Wilayah AWS Di mana instance database global ini berjalan. Untuk daftar Wilayah, lihat Ketersediaan wilayah. |

| Kolom | Jenis data | Deskripsi |
| --- | --- | --- |
| DURABLE_LSN | bigint unsigned | Nomor urutan log (LSN) dibuat tahan lama dalam penyimpan an. Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNsdipesan sedemikian rupa sehingga yang lebih besar LSN mewakili transaksi selanjutnya. |
| HIGHEST_LSN_RCVD | bigint unsigned | Yang tertinggi LSN diterima oleh instans DB dari instans DB penulis. |
| OLDEST_ READ _ _ VIEW TRX _ ID | bigint unsigned | ID transaksi terlama yang dapat dibuang oleh instans DB penulis. |
| OLDEST_READ_VIEW_LSN | bigint unsigned | Yang tertua LSN digunakan oleh instans DB untuk membaca dari penyimpanan. |
| VISIBILITY_ LAG _IN_ MSEC | float(10,0) unsigned | Untuk pembaca di klaster DB primer, seberapa jauh instans DB ini tertinggal dari instans DB penulis dalam milidetik. Untuk pembaca di DB klaster sekunder, seberapa jauh DB instans ini tertinggal dari volume sekunder dalam milidetik. |

## information_schema.aurora_global_db_status

`information_schema.aurora_global_db_status`Tabel berisi informasi tentang berbagai aspek lag database global Aurora, khususnya, lag penyimpanan Aurora yang mendasarinya (disebut lag daya tahan) dan jeda antara tujuan titik pemulihan (). RPO Tabel berikut menunjukkan kolom yang dapat Anda gunakan. Kolom yang tersisa hanya ditujukan untuk penggunaan internal Aurora.

> **Note**
>
> Tabel skema informasi ini hanya tersedia dengan Aurora SQL My versi 3.04.0 dan database global yang lebih tinggi.

| Kolom | Jenis data | Deskripsi |
|---|---|---|
| AWS_REGION | varchar(100) | Wilayah AWS Di mana instance database global ini berjalan. Untuk daftar Wilayah, lihat Ketersediaan wilayah. |
| HIGHEST_LSN_WRITTEN | bigint unsigned | Nomor urutan log tertinggi (LSN) yang saat ini ada di cluster DB ini. Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNsdipesan sedemikian rupa sehingga yang lebih besar LSN mewakili transaksi selanjutnya. |
| DURABILITY_ LAG _IN_ MILLISECONDS | float(10,0) unsigned | Perbedaan nilai stempel waktu antara `HIGHEST_LSN_WRITTEN` di klaster DB sekunder dan `HIGHEST_LSN_WRITTEN` di klaster DB primer. Nilai ini selalu 0 pada |

| Kolom | Jenis data | Deskripsi |
| --- | --- | --- |
| | | klaster DB primer di basis data global Aurora. |
| RPO_ LAG _IN_ MILLISECONDS | float(10,0) unsigned | Tujuan titik pemulihan (RPO) lag. RPOKelambatan adalah waktu yang dibutuhkan untuk transaksi pengguna terbaru COMMIT untuk disimpan di cluster DB sekunder setelah disimpan di cluster DB utama dari database global Aurora. Nilai ini selalu 0 pada klaster DB primer basis data global Aurora.<br><br>Secara sederhana, metrik ini menghitung tujuan titik pemulihan untuk setiap cluster Aurora SQL My DB di database global Aurora, yaitu, berapa banyak data yang mungkin hilang jika ada pemadaman. Seperti halnya lag, RPO diukur dalam waktu. |
| LAST_LAG_CALCULATION_TIMESTAMP | datetime | Stempel waktu yang menentukan kapan nilai terakhir dihitung untuk `DURABILITY_LAG_IN_MILLISECONDS` dan `RPO_LAG_IN_MILLISECONDS`. Nilai waktu seperti `1970-01-01 00:00:00+00` menunjukkan bahwa ini adalah klaster DB primer. |

| Kolom | Jenis data | Deskripsi |
| --- | --- | --- |
| OLDEST_ READ _ _ VIEW TRX _ ID | bigint unsigned | ID transaksi terlama yang dapat dibuang oleh instans DB penulis. |

## information_schema.replica_host_status

Tabel `information_schema.replica_host_status` berisi informasi replikasi. Kolom yang dapat Anda gunakan ditunjukkan pada tabel berikut. Kolom yang tersisa hanya ditujukan untuk penggunaan internal Aurora.

| Kolom | Jenis data | Deskripsi |
| --- | --- | --- |
| CPU | double | CPUPersentase penggunaan host replika. |
| ADALAH_ CURRENT | tinyint | Apakah replika adalah yang terbaru atau tidak. |
| LAST_UPDATE_TIMESTAMP | datetime(6) | Waktu pembaruan terakhir terjadi. Digunakan untuk menentukan apakah sebuah catatan sudah usang. |
| REPLICA_ LAG _IN_ MILLISECONDS | double | Lag replika dalam milidetik. |
| SERVER_ID | varchar(100) | ID server basis data. |
| SESSION_ID | varchar(100) | ID sesi basis data. Digunakan untuk menentukan apakah instans DB adalah instans penulis atau pembaca. |

> **Note**
>
> Ketika instans replika tertinggal, informasi yang dikueri dari tabel `information_schema.replica_host_status` milik instans replika tersebut mungkin sudah usang. Dalam situasi ini, kami menyarankan Anda mengueri dari instans penulis sebagai gantinya.
> Meskipun tabel `mysql.ro_replica_status` memiliki informasi serupa, kami tidak menyarankan Anda untuk menggunakannya.

## information_schema.aurora_forwarding_processlist

Tabel `information_schema.aurora_forwarding_processlist` berisi informasi tentang proses terkait dalam penerusan penulisan.

Konten tabel ini hanya terlihat pada instans DB penulis untuk klaster DB dengan penerusan penulisan global atau penerusan penulisan dalam klaster diaktifkan. Set hasil kosong dihasilkan pada instans DB pembaca.

| Bidang | Jenis data | Deskripsi |
|---|---|---|
| ID | bigint | Pengidentifikasi koneksi pada instans DB penulis. Pengidentifikasi ini adalah nilai yang sama yang ditampilkan di kolom `Id` untuk pernyataan `SHOW PROCESSLIST` dan dihasilkan oleh fungsi `CONNECTION_ID()` di dalam thread. |
| USER | varchar(32) | SQLPengguna saya yang mengeluarkan pernyataan. |
| HOST | varchar(255) | SQLKlien saya yang mengeluarkan pernyataan itu. Untuk pernyataan yang diteruskan, bidang ini menunjukkan alamat host klien aplikasi yang membuat koneksi pada instans DB pembaca yang melakukan penerusan. |
| DB | varchar(64) | Basis data default untuk thread. |

| Bidang | Jenis data | Deskripsi |
| --- | --- | --- |
| COMMAND | varchar(16) | Jenis perintah yang dijalankan thread atas nama klien, atau `Sleep` jika sesinya idle. Untuk deskripsi perintah thread, lihat SQL Dokumentasi saya tentang Nilai Perintah Thread dalam SQL dokumentasi Saya. |
| TIME | int | Waktu dalam hitungan detik saat thread berada dalam status saat ini. |
| STATE | varchar(64) | Tindakan, peristiwa, atau status yang menunjukkan apa yang dilakukan thread. Untuk deskripsi nilai status, lihat Status Utas Umum dalam SQL dokumentasi Saya. |
| INFO | longtext | Pernyataan bahwa thread sedang mengeksekusi pernyataan, atau `NULL` jika tidak sedang mengeksekusi pernyataan. Pernyataan ini mungkin adalah pernyataan yang dikirim ke server, atau mungkin adalah pernyataan terdalam jika mengeksekusi pernyataan lain. |
| ADALAH_FORWARDED | bigint | Menunjukkan apakah thread diteruskan dari instans DB pembaca. |
| REPLICASESSION_ID | bigint | Pengidentifikasi koneksi pada Replika Aurora. Pengidentifikasi ini adalah nilai yang sama yang ditampilkan di kolom `Id` untuk pernyataan `SHOW PROCESSLIST` pada instans DB Aurora pembaca yang melakukan penerusan. |
| REPLICA_INSTANCE_IDENTIFIER | varchar(64) | Pengidentifikasi instans DB dari thread penerusan. |
| REPLICA_CLUSTER_NAME | varchar(64) | Pengidentifikasi klaster DB dari thread penerusan. Untuk penerusan penulisan dalam klaster, pengidentifikasi ini adalah klaster DB yang sama dengan instans DB penulis. |

| Bidang | Jenis data | Deskripsi |
| --- | --- | --- |
| REPLICA_REGION | varchar(64) | Wilayah AWS Dari mana utas penerusan berasal. Untuk penerusan penulisan dalam klaster, Wilayah ini adalah Wilayah AWS yang sama dengan instans DB penulis. |

# Pembaruan mesin database untuk Amazon Aurora My SQL

Amazon Aurora merilis pembaruan secara berkala. Pembaruan diterapkan pada klaster Aurora DB selama periode pemeliharaan sistem. Pengaturan waktu saat pembaruan diterapkan tergantung pada wilayah dan pengaturan periode pemeliharaan untuk klaster DB, serta jenis pembaruannya.

Rilis Amazon Aurora tersedia untuk semua AWS Wilayah selama beberapa hari. Untuk sementara waktu, beberapa Wilayah mungkin menampilkan versi mesin yang belum tersedia di Wilayah lain.

Pembaruan diterapkan ke semua instans dalam klaster DB pada saat yang sama. Pembaruan memerlukan pengaktifan ulang basis data di semua instans dalam klaster DB, sehingga Anda akan mengalami waktu henti 20 hingga 30 detik. Setelah itu, Anda dapat melanjutkan menggunakan klaster DB Anda. Anda dapat melihat atau mengubah pengaturan jendela pemeliharaan dari AWS Management Console.

Untuk detail tentang Aurora SQL versi Saya yang didukung oleh Amazon Aurora, lihat Catatan Rilis untuk Aurora Milikku. SQL

Setelah itu, Anda dapat mempelajari cara memilih SQL versi Aurora My yang tepat untuk klaster Anda, cara menentukan versi saat Anda membuat atau memutakhirkan cluster, dan prosedur untuk meningkatkan cluster dari satu versi ke versi lainnya dengan gangguan minimal.

Topik


- Memeriksa Aurora Nomor versi saya SQL
- Dukungan jangka panjang (LTS) dan rilis beta untuk Amazon Aurora My SQL
- Mempersiapkan Amazon Aurora My SQL -Compatible Edition versi 2 akhir dukungan standar
- Mempersiapkan Amazon Aurora My SQL -Compatible Edition versi 1 akhir hayat
- Memutakhirkan Amazon Aurora SQL My DB cluster
- Pembaruan dan perbaikan mesin database untuk Amazon Aurora MySQL


## Memeriksa Aurora Nomor versi saya SQL

Meskipun Aurora My SQL -Compatible Edition kompatibel dengan mesin SQL database Saya, Aurora My SQL menyertakan fitur dan perbaikan bug yang khusus untuk versi Aurora My tertentu. SQL Pengembang aplikasi dapat memeriksa SQL versi Aurora My di aplikasi mereka dengan menggunakan. SQL Administrator database dapat memeriksa dan menentukan versi Aurora SQL My saat membuat atau memutakhirkan cluster Aurora SQL My DB dan instans DB.

Topik


- Memeriksa atau menentukan versi Aurora SQL My engine melalui AWS
- Memeriksa Aurora Versi saya menggunakan SQL SQL


## Memeriksa atau menentukan versi Aurora SQL My engine melalui AWS

Saat Anda melakukan tugas administratif menggunakan,, atau AWS Management Console AWS CLI RDSAPI, Anda menentukan SQL versi Aurora My dalam format alfanumerik deskriptif.

Dimulai dengan Aurora My SQL versi 2, versi mesin Aurora memiliki sintaks berikut.

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

Bagian *mysql-major-version-* adalah `5.7` atau `8.0`. Nilai ini mewakili versi protokol klien dan tingkat umum dukungan SQL fitur Saya untuk Aurora versi Saya SQL yang sesuai.

*aurora-mysql-version*Ini adalah nilai putus-putus dengan tiga bagian: Aurora My SQL major version, Aurora My SQL minor version, dan patch level. Versi mayor adalah `2` atau `3`. Nilai-nilai tersebut mewakili Aurora My SQL kompatibel dengan My SQL 5.7 atau 8.0, masing-masing. Versi minor merepresentasikan rilis fitur dalam seri 2.x atau 3.x. Tingkat patch dimulai dari `0` untuk setiap versi minor, dan merepresentasikan serangkaian perbaikan bug berikutnya yang berlaku untuk versi minor. Terkadang, fitur baru disertakan ke dalam versi minor, tetapi tidak segera terlihat. Dalam hal ini, fitur tersebut mendapatkan penyesuaian dan diterbitkan di tingkat patch berikutnya.

Semua versi 2.x Aurora SQL My engine kompatibel dengan kabel dengan Community SQL My 5.7.12 atau lebih tinggi. Semua versi 3.x Aurora SQL My engine kompatibel dengan kabel dengan SQL My 8.0.23 atau lebih tinggi. Anda dapat merujuk ke catatan rilis versi 3.x tertentu untuk menemukan versi SQL kompatibel Saya yang sesuai.

Misalnya, versi mesin untuk Aurora My SQL 3.04.0 dan 2.11.2 adalah sebagai berikut.

```
8.0.mysql_aurora.3.04.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> Tidak ada one-to-one korespondensi antara versi Komunitas Saya dan SQL versi Aurora SQL My 2.x. Untuk Aurora My SQL versi 3, ada pemetaan yang lebih langsung. Untuk memeriksa perbaikan bug dan fitur baru yang ada di Aurora Rilis SQL saya tertentu, lihat Pembaruan

mesin database untuk Amazon Aurora Versi saya 3 dan pembaruan mesin Database untuk Amazon Aurora Versi SQL saya 2 di Catatan Rilis untuk Aurora Milik SQL Saya. SQL Untuk daftar kronologis fitur dan rilis baru, lihat Riwayat dokumen. Untuk memeriksa versi minimum yang diperlukan untuk perbaikan terkait keamanan, lihat Kerentanan keamanan diperbaiki di Aurora Milik di Catatan Rilis untuk Aurora SQL Milik Saya. SQL

Anda menentukan versi Aurora My SQL engine di beberapa AWS CLI perintah dan RDS API operasi. Misalnya, Anda menentukan `--engine-version` opsi ketika Anda menjalankan AWS CLI perintah create-db-clusterdan modify-db-cluster. Anda menentukan `EngineVersion` parameter ketika Anda menjalankan RDS API operasi C reateDBCluster dan M odifyDBCluster.

Di Aurora My SQL versi 2 dan lebih tinggi, versi mesin di AWS Management Console juga menyertakan versi Aurora. Peningkatan klaster akan mengubah nilai yang ditampilkan. Perubahan ini membantu Anda menentukan dan memeriksa SQL versi Aurora My yang tepat, tanpa perlu terhubung ke cluster atau menjalankan perintah apa punSQL.

> **Tip**
>
> Untuk cluster Aurora yang dikelola melalui AWS CloudFormation, perubahan dalam `EngineVersion` pengaturan ini dapat memicu tindakan oleh. AWS CloudFormation Untuk informasi tentang cara AWS CloudFormation memperlakukan perubahan pada `EngineVersion` setelan, lihat AWS CloudFormation dokumentasi.

## Memeriksa Aurora Versi saya menggunakan SQL SQL

Nomor versi Aurora yang dapat Anda ambil dalam aplikasi Anda menggunakan SQL kueri menggunakan format. *`<major version>`*`.`*`<minor version>`*`.`*`<patch version>` Anda bisa mendapatkan nomor versi ini untuk instans DB apa pun di SQL cluster Aurora My Anda dengan menanyakan variabel sistem`AURORA_VERSION`. Untuk mendapatkan nomor versi ini, gunakan salah satu kueri berikut.

```
select aurora_version();
select @@aurora_version;
```

Kueri tersebut menghasilkan output seperti yang berikut ini.

```
mysql> select aurora_version(), @@aurora_version;
```

```
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Nomor versi yang konsol,CLI, dan RDS API kembali dengan menggunakan teknik yang dijelaskan dalam Memeriksa atau menentukan versi Aurora SQL My engine melalui AWS biasanya lebih deskriptif.

# Dukungan jangka panjang (LTS) dan rilis beta untuk Amazon Aurora My SQL

Aurora My SQL menyediakan dukungan jangka panjang (LTS) dan rilis beta untuk beberapa versi mesin Aurora My. SQL

Topik

- Aurora Dukungan SQL jangka panjang saya () rilis LTS
- Aurora My beta rilis SQL

## Aurora Dukungan SQL jangka panjang saya () rilis LTS

Setiap SQL versi Aurora My yang baru tetap tersedia untuk jangka waktu tertentu untuk Anda gunakan saat Anda membuat atau meningkatkan cluster DB. Setelah periode ini, Anda harus meningkatkan klaster yang menggunakan versi tersebut. Anda dapat memutakhirkan klaster Anda secara manual sebelum periode dukungan berakhir, atau Aurora dapat secara otomatis memutakhirkannya untuk Anda ketika SQL versi Aurora My tidak lagi didukung.

Aurora menunjuk SQL versi Aurora My tertentu sebagai rilis support () jangka panjang. LTS Cluster DB yang menggunakan LTS rilis dapat tetap pada versi yang sama lebih lama dan mengalami siklus peningkatan yang lebih sedikit daripada cluster yang menggunakan LTS non-rilis. Aurora mendukung setiap LTS rilis setidaknya selama tiga tahun setelah rilis itu tersedia. Ketika cluster DB yang ada dalam LTS rilis diperlukan untuk meningkatkan, Aurora meningkatkannya ke rilis berikutnya. LTS Dengan demikian, klaster ini tidak perlu ditingkatkan lagi dalam jangka waktu lama.

Selama masa SQL LTS rilis Aurora My, level patch baru memperkenalkan perbaikan pada masalah penting. Tingkat patch tidak mencakup fitur baru apa pun. Anda dapat memilih apakah akan menerapkan tambalan tersebut ke cluster DB yang menjalankan rilis. LTS Untuk perbaikan penting

tertentu, Amazon mungkin melakukan upgrade terkelola ke tingkat patch dalam LTS rilis yang sama. Peningkatan terkelola tersebut dilakukan secara otomatis dalam periode pemeliharaan klaster.

Kami menyarankan Anda meningkatkan ke rilis terbaru, alih-alih menggunakan LTS rilis, untuk sebagian besar cluster Aurora My SQL Anda. Melakukan hal itu memanfaatkan Aurora sebagai layanan terkelola dan memberi Anda akses ke fitur terbaru dan perbaikan bug. LTSRilis ditujukan untuk cluster dengan karakteristik sebagai berikut:

- Anda tidak dapat membayar downtime pada aplikasi Aurora SQL My Anda untuk peningkatan di luar kejadian langka untuk tambalan kritis.
- Siklus pengujian untuk cluster dan aplikasi terkait membutuhkan waktu lama untuk setiap pembaruan ke mesin SQL database Aurora My.
- Versi database untuk Aurora My SQL cluster Anda memiliki semua fitur mesin DB dan perbaikan bug yang dibutuhkan aplikasi Anda.

LTSRilis saat ini untuk Aurora My SQL adalah sebagai berikut:

- Aurora SQL Versi saya 3.04.*. Untuk detail selengkapnya tentang LTS versi ini, lihat Pembaruan mesin database untuk Amazon Aurora SQL Versi saya 3 di Catatan Rilis untuk Aurora Milikku. SQL

> **Note**
>
> Kami menyarankan Anda menonaktifkan upgrade versi minor otomatis untuk LTS versi. Setel `AutoMinorVersionUpgrade` parameter `kefalse`, atau hapus kotak centang Aktifkan pemutakhiran versi minor otomatis pada AWS Management Console.
> Jika Anda tidak menonaktifkannya, cluster DB Anda dapat ditingkatkan ke LTS non-versi seperti 3.05.2.

## Aurora My beta rilis SQL

Rilis Aurora My SQL beta adalah rilis awal, perbaikan keamanan—hanya rilis dalam jumlah terbatas Wilayah AWS. Perbaikan ini kemudian diterapkan secara lebih luas di semua Wilayah dengan rilis patch berikutnya.

Penomoran untuk rilis beta mirip dengan versi minor Aurora SQL My, tetapi dengan digit keempat tambahan, misalnya 2.12.0.1 atau 3.05.0.1.

Untuk informasi selengkapnya, lihat Pembaruan mesin database untuk Amazon Aurora My SQL version 2 dan pembaruan mesin Database untuk Amazon Aurora SQL My version 3 di Catatan Rilis untuk Aurora My. SQL

## Mempersiapkan Amazon Aurora My SQL -Compatible Edition versi 2 akhir dukungan standar

Amazon Aurora My SQL -Compatible Edition versi 2 (dengan kompatibilitas My SQL 5.7) direncanakan akan mencapai akhir dukungan standar pada 31 Oktober 2024. Kami menyarankan Anda memutakhirkan semua cluster yang menjalankan Aurora SQL My versi 2 ke Aurora SQL My versi 3 default (dengan kompatibilitas SQL My 8.0) atau lebih tinggi sebelum Aurora SQL My versi 2 mencapai akhir periode dukungan standarnya. Pada 31 Oktober 2024, Amazon RDS akan secara otomatis mendaftarkan database Anda ke Amazon Extended RDS Support. Jika Anda menjalankan Amazon Aurora My SQL versi 2 (dengan kompatibilitas SQL 5.7 Saya) di Aurora Serverless cluster versi 1, ini tidak berlaku untuk Anda. Jika Anda ingin meng-upgrade Aurora Serverless klaster versi 1 ke Aurora Versi SQL saya 3, lihat. Jalur upgrade untuk Aurora Serverless v1 Klaster DB

Anda dapat menemukan end-of-support tanggal mendatang untuk Aurora versi SQL utama saya di Kalender rilis untuk Aurora versi utama saya. SQL

Jika Anda memiliki cluster yang menjalankan Aurora SQL My versi 2, Anda akan menerima pemberitahuan berkala dengan informasi terbaru tentang cara melakukan peningkatan saat kami mendekati akhir tanggal dukungan standar. Kami akan memperbarui halaman ini secara berkala dengan informasi terbaru.

### Akhir dari garis waktu dukungan standar

1. Sekarang hingga 31 Oktober 2024 - Anda dapat memutakhirkan cluster dari Aurora My SQL versi 2 (dengan SQL kompatibilitas 5.7 Saya) ke Aurora My SQL versi 3 (dengan kompatibilitas My 8.0). SQL
2. 31 Oktober 2024 - Pada tanggal ini, Aurora SQL My versi 2 akan mencapai akhir dukungan standar dan Amazon secara otomatis mendaftarkan cluster Anda ke RDS Amazon Extended Support. RDS

Kami akan secara otomatis mendaftarkan Anda di RDS Extended Support. Untuk informasi selengkapnya, lihat Amazon RDS Extended Support dengan .

## Menemukan cluster yang terpengaruh oleh proses ini end-of-life

Untuk menemukan cluster yang terpengaruh oleh end-of-life proses ini, gunakan prosedur berikut.

> **Important**
>
> Pastikan untuk melakukan instruksi ini di setiap Wilayah AWS dan untuk setiap Akun AWS tempat sumber daya Anda berada.

### Konsol

Untuk menemukan cluster Aurora My SQL version 2

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Di kotak Filter berdasarkan basis data, masukkan 5.7.
4. Periksa Aurora My SQL di kolom mesin.

### AWS CLI

Untuk menemukan cluster yang terpengaruh oleh end-of-life proses ini menggunakan AWS CLI, panggil describe-db-clustersperintah. Anda dapat menggunakan contoh skrip berikut.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
 DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
 --region us-east-1

        +-----------------------------------------------------------------+
        |                       DescribeDBClusters                        |
        +---------------------+---------------+-------------------------+
        |         DBCI        |      EM       |          EV             |
        +---------------------+---------------+-------------------------+
        |    aurora-mysql2    |  provisioned  | 5.7.mysql_aurora.2.11.3 |
        | aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
        +---------------------+---------------+-------------------------+
```

RDS API

Untuk menemukan cluster Aurora My SQL DB yang menjalankan Aurora My SQL versi 2, gunakan escribeDBClusters API operasi RDS D dengan parameter yang diperlukan berikut:

- `DescribeDBClusters`
  - Filters.Filter.N
    - Name
      - engine
    - Values.Value.N
      - ['aurora']

## Amazon RDS Extended Support

Anda dapat menggunakan Amazon RDS Extended Support melalui komunitas My SQL 5.7 tanpa biaya hingga akhir tanggal dukungan, 31 Oktober 2024. Pada tanggal 31 Oktober 2024, Amazon RDS secara otomatis mendaftarkan database Anda ke Extended RDS Support for Aurora My version 2. SQL RDS Extended Support for Aurora adalah layanan berbayar yang menyediakan hingga 28 bulan dukungan tambahan untuk Aurora My SQL versi 2 hingga akhir Extended RDS Support pada Februari 2027. RDS Extended Support hanya akan ditawarkan untuk Aurora My SQL minor versi 2.11 dan 2.12. Untuk menggunakan Amazon Aurora My SQL versi 2 melewati akhir dukungan standar, rencanakan untuk menjalankan database Anda di salah satu versi minor ini sebelum 31 Oktober 2024.

Untuk informasi selengkapnya tentang RDS Extended Support, seperti biaya dan pertimbangan lainnya, lihatAmazon RDS Extended Support dengan .

## Melakukan upgrade

Peningkatan antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama. Kami ingin menjelaskan peningkatan sebagai proses tiga langkah, dengan aktivitas sebelum peningkatan, selama peningkatan, dan setelah peningkatan.

Sebelum peningkatan:

Sebelum peningkatan, sebaiknya Anda memeriksa kompatibilitas aplikasi, performa, prosedur pemeliharaan, dan pertimbangan serupa untuk klaster yang ditingkatkan, sehingga akan mengonfirmasi bahwa setelah peningkatan, aplikasi Anda akan berfungsi seperti yang diharapkan. Berikut adalah lima rekomendasi yang akan membantu memberi Anda pengalaman peningkatan yang lebih baik.

- Pertama, sangat penting untuk dipahamiCara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL.
- Selanjutnya, jelajahi teknik peningkatan yang tersedia saatUpgrade dari Aurora MySQL versi 2 ke versi 3.
- Untuk membantu Anda memutuskan waktu dan pendekatan yang tepat untuk meningkatkan, Anda dapat mempelajari perbedaan antara Aurora My SQL versi 3 dan lingkungan Anda saat ini. Membandingkan Aurora SQL Versi saya 2 dan Aurora Versi saya 3 SQL
- Setelah Anda memutuskan opsi yang nyaman dan berfungsi paling baik, coba upgrade tiruan di tempat pada kloning kloning, gunakan. Merencanakan peningkatan versi mayor untuk klaster Aurora MySQL
- Tinjau Pra-cek pemutakhiran versi utama untuk Aurora My SQL. Pra-pemeriksa pemutakhiran dapat menjalankan dan menentukan apakah database Anda dapat ditingkatkan dengan sukses, dan jika ada masalah ketidakcocokan aplikasi pasca-peningkatan serta kinerja, prosedur pemeliharaan, dan pertimbangan serupa.
- Tidak semua jenis atau versi Aurora My SQL cluster dapat menggunakan mekanisme peningkatan di tempat. Untuk informasi selengkapnya, lihat Jalur peningkatan versi mayor Aurora MySQL.

Jika Anda memiliki pertanyaan atau masalah, AWS Support Team tersedia di forum komunitas dan Premium Support.

Melakukan peningkatan:

Anda dapat menggunakan salah satu teknik peningkatan berikut. Jumlah downtime yang akan dialami sistem Anda tergantung pada teknik yang dipilih.

- Penerapan Biru/Hijau — Untuk situasi di mana prioritas utamanya adalah mengurangi waktu henti aplikasi, Anda dapat menggunakan Amazon RDSBlue/Green Deployment untuk melakukan peningkatan versi utama di klaster Amazon Aurora DB yang disediakan. Deployment blue/green membuat lingkungan staging yang menyerupai lingkungan produksi. Anda dapat membuat perubahan tertentu pada klaster DB Aurora di lingkungan green (staging) tanpa memengaruhi beban kerja produksi. Switchover biasanya memakan waktu kurang dari satu menit tanpa

kehilangan data. Untuk informasi selengkapnya, lihat Ikhtisar Amazon Aurora Penerapan Biru/Hijau. Hal ini meminimalkan waktu henti, tetapi mengharuskan Anda untuk menjalankan sumber daya tambahan saat melakukan peningkatan.

- Upgrade di tempat — Anda dapat melakukan upgrade di tempat di mana Aurora secara otomatis melakukan proses precheck untuk Anda, membuat cluster offline, mencadangkan cluster Anda, melakukan upgrade, dan menempatkan cluster Anda kembali online. Upgrade versi utama di tempat dapat dilakukan dalam beberapa klik, dan tidak melibatkan koordinasi atau kegagalan lain dengan cluster lain, tetapi melibatkan downtime. Untuk informasi selengkapnya, silakan lihat Cara melakukan peningkatan di tempat
- Pemulihan snapshot - Anda dapat memutakhirkan cluster Aurora SQL My version 2 Anda dengan memulihkan dari snapshot Aurora My version 2 ke cluster Aurora SQL My version 3. SQL Untuk melakukannya, Anda harus mengikuti proses untuk mengambil snapshot dan memulihkannya. Proses ini melibatkan gangguan database karena Anda memulihkan dari snapshot.

Setelah peningkatan:

Setelah upgrade, Anda perlu memonitor sistem Anda (aplikasi dan database) dan membuat perubahan fine-tuning jika perlu. Mengikuti langkah-langkah pra-peningkatan dengan cermat akan meminimalkan perubahan yang diperlukan. Untuk informasi selengkapnya, lihat Memecahkan masalah Amazon Aurora Kinerja database saya SQL.

Untuk mempelajari lebih lanjut tentang metode, perencanaan, pengujian, dan pemecahan masalah peningkatan versi SQL utama Aurora My, pastikan untuk membaca secara menyeluruh, termasuk. Meningkatkan versi mayor klaster DB Amazon Aurora MySQL Pemecahan masalah untuk Aurora Peningkatan di tempat saya SQL Juga, perhatikan bahwa beberapa jenis instance tidak didukung untuk Aurora My SQL versi 3. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB.

## Jalur upgrade untuk Aurora Serverless v1 Klaster DB

Peningkatan antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama. Kami ingin menjelaskan peningkatan sebagai proses tiga langkah, dengan aktivitas sebelum peningkatan, selama peningkatan, dan setelah peningkatan.

Aurora SQL Versi saya 2 (dengan kompatibilitas SQL 5.7 Saya) akan terus menerima dukungan standar untuk Aurora Serverless v1 klaster.

Jika Anda ingin meningkatkan ke Amazon Aurora My SQL 3 (dengan kompatibilitas My SQL 8.0) dan terus berjalan Aurora Serverless, Anda dapat menggunakan Amazon Aurora Serverless v2. Untuk memahami perbedaan antara Aurora Serverless v1 and Aurora Serverless v2, lihat Perbandingan Aurora Serverless v2 and Aurora Serverless v1.

Tingkatkan ke Aurora Serverless v2:  Anda dapat meng-upgrade Aurora Serverless v1 cluster ke Aurora Serverless v2Untuk informasi selengkapnya, lihat Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2.

## Mempersiapkan Amazon Aurora My SQL -Compatible Edition versi 1 akhir hayat

Amazon Aurora My SQL -Compatible Edition versi 1 (dengan kompatibilitas My SQL 5.6) direncanakan akan mencapai akhir masa pakai pada 28 Februari 2023. Amazon menyarankan agar Anda memutakhirkan semua cluster (disediakan dan) Aurora Serverless menjalankan Aurora My versi SQL 1 ke Aurora My versi 2 (dengan kompatibilitas My 5.7) atau Aurora My SQL versi 3 (dengan SQL kompatibilitas My 8.0). SQL SQL Lakukan ini sebelum Aurora SQL Versi saya 1 mencapai akhir periode dukungannya.

Untuk cluster DB yang disediakan Aurora, Anda dapat menyelesaikan peningkatan dari Aurora My versi SQL 1 ke Aurora My versi 2 dengan beberapa metode. SQL Anda dapat menemukan petunjuk untuk mekanisme peningkatan di tempat dalam Cara melakukan peningkatan di tempat. Cara lain untuk menyelesaikan peningkatan adalah dengan mengambil snapshot dari cluster Aurora SQL My version 1 dan mengembalikan snapshot ke cluster Aurora SQL My version 2. Atau, Anda dapat mengikuti proses beberapa langkah yang menjalankan klaster lama dan baru secara berdampingan. Untuk detail selengkapnya tentang setiap metode, lihat Meningkatkan versi mayor klaster DB Amazon Aurora MySQL.

Untuk cluster Aurora Serverless v1 DB, Anda dapat melakukan peningkatan di tempat dari Aurora My versi SQL 1 ke Aurora My versi 2. SQL Untuk detail selengkapnya tentang metode ini, lihat Memodifikasi Aurora Serverless v1 Klaster DB.

Untuk cluster DB yang disediakan Aurora, Anda dapat menyelesaikan peningkatan dari Aurora My versi SQL 1 ke Aurora My versi 3 dengan menggunakan proses peningkatan dua tahap: SQL

1. Tingkatkan dari Aurora My SQL version 1 ke Aurora My SQL version 2 menggunakan metode yang dijelaskan sebelumnya.
2. Tingkatkan dari Aurora SQL Versi saya 2 ke Aurora SQL Versi saya 3 menggunakan metode yang sama seperti untuk meningkatkan dari versi 1 ke versi 2. Untuk detail selengkapnya, lihat Upgrade

dari Aurora MySQL versi 2 ke versi 3. Perhatikan Perbedaan fitur antara Aurora My SQL versi 2 dan 3.

Anda dapat menemukan end-of-life tanggal mendatang untuk versi utama Aurora di. Versi Amazon Aurora Amazon secara otomatis memutakhirkan cluster apa pun yang tidak Anda tingkatkan sendiri sebelum tanggal. end-of-life Setelah end-of-life tanggal, upgrade otomatis ini ke versi utama berikutnya terjadi selama jendela pemeliharaan terjadwal untuk cluster.

Berikut ini adalah tonggak tambahan untuk meningkatkan klaster Aurora My SQL versi 1 (disediakan danAurora Serverless) yang mencapai akhir hayat. Untuk masing-masing, waktu mulai adalah 00:00 Waktu Terkoordinasi Universal ()UTC.

1. Sekarang hingga 28 Februari 2023 - Anda dapat setiap saat memulai peningkatan cluster Aurora My SQL versi 1 (dengan kompatibilitas SQL 5.6 Saya) ke Aurora My SQL versi 2 (dengan kompatibilitas 5.7 Saya). SQL Dari Aurora My SQL versi 2, Anda dapat melakukan peningkatan lebih lanjut ke Aurora My SQL versi 3 (dengan SQL kompatibilitas My 8.0) untuk cluster DB yang disediakan Aurora.
2. 16 Januari 2023 — Setelah waktu ini, Anda tidak dapat membuat cluster atau instance Aurora SQL My version 1 baru dari atau AWS Management Console (). AWS Command Line Interface AWS CLI Anda juga tidak dapat menambahkan Wilayah sekunder baru ke sebuah basis data global Aurora. Ini mungkin memengaruhi kemampuan Anda untuk pulih dari pemadaman yang tidak direncanakan seperti yang diuraikan dalam Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan karena Anda tidak dapat menyelesaikan langkah 5 dan 6 setelah waktu ini. Anda juga tidak akan dapat membuat replika baca Lintas wilayah baru yang menjalankan Aurora My versi 1. SQL Anda masih dapat melakukan hal berikut untuk cluster Aurora My SQL versi 1 yang ada hingga 28 Februari 2023:
   - Kembalikan snapshot yang diambil dari cluster Aurora SQL My version 1 ke versi yang sama dengan cluster snapshot asli.
   - Tambahkan replika baca (tidak berlaku untuk klaster DB Aurora Serverless).
   - Ubah konfigurasi instans.
   - Lakukan point-in-time pemulihan.
   - Buat klon dari klaster versi 1 yang ada.
   - Buat replika baca Lintas wilayah baru yang menjalankan Aurora My SQL versi 2 atau lebih tinggi.

3. 28 Februari 2023 — Setelah waktu ini, kami berencana untuk secara otomatis meningkatkan klaster Aurora SQL My versi 1 ke versi default Aurora SQL My versi 2 dalam jendela pemeliharaan terjadwal berikut. Memulihkan snapshot Aurora SQL My version 1 DB menghasilkan upgrade otomatis cluster yang dipulihkan ke versi default Aurora SQL My versi 2 pada waktu itu.

Peningkatan antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama.

Untuk situasi yang prioritas utamanya adalah mengurangi waktu henti, Anda juga dapat menggunakan deployment blue/green untuk melakukan peningkatan versi mayor di klaster DB Amazon Aurora terprovisi. Deployment blue/green membuat lingkungan staging yang menyerupai lingkungan produksi. Anda dapat membuat perubahan pada klaster DB Aurora di lingkungan green (staging) tanpa memengaruhi beban kerja produksi. Switchover biasanya memakan waktu kurang dari satu menit tanpa kehilangan data dan tidak memerlukan perubahan aplikasi. Untuk informasi selengkapnya, lihat Ikhtisar Amazon Aurora Penerapan Biru/Hijau.

Setelah peningkatan selesai, Anda mungkin juga memiliki pekerjaan tindak lanjut yang harus dilakukan. Misalnya, Anda mungkin perlu menindaklanjuti karena perbedaan SQL kompatibilitas, cara kerja fitur SQL terkait Saya tertentu, atau pengaturan parameter antara versi lama dan baru.

Untuk mempelajari lebih lanjut tentang metode, perencanaan, pengujian, dan pemecahan masalah peningkatan versi SQL utama Aurora My, pastikan untuk membaca secara menyeluruh. Meningkatkan versi mayor klaster DB Amazon Aurora MySQL

## Menemukan cluster yang terpengaruh oleh proses ini end-of-life

Untuk menemukan cluster yang terpengaruh oleh end-of-life proses ini, gunakan prosedur berikut.

> **Important**
>
> Pastikan untuk melakukan instruksi ini di setiap Wilayah AWS dan untuk setiap Akun AWS tempat sumber daya Anda berada.

Konsol

Untuk menemukan cluster Aurora My SQL version 1

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Basis Data.
3. Di kotak Filter berdasarkan basis data, masukkan 5.6.
4. Periksa Aurora My SQL di kolom mesin.

AWS CLI

Untuk menemukan cluster yang terpengaruh oleh end-of-life proses ini menggunakan AWS CLI, panggil describe-db-clustersperintah. Anda dapat menggunakan contoh skrip berikut.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1

        +-----------------------------------------+
        |            DescribeDBClusters           |
        +---------------+--------------+----------+
        |     DBCI      |     EM       |    EV    |
        +---------------+--------------+----------+
        |  my-database-1|  serverless  |  5.6.10a |
        +---------------+--------------+----------+
```

RDS API

Untuk menemukan klaster Aurora My SQL DB yang menjalankan Aurora My SQL versi 1, gunakan escribeDBClusters API operasi RDS D dengan parameter yang diperlukan berikut:

- `DescribeDBClusters`
  - Filters.Filter.N
    - Name
      - engine
    - Values.Value.N
      - ['aurora']

# Memutakhirkan Amazon Aurora SQL My DB cluster

Anda dapat memutakhirkan cluster Aurora My SQL DB untuk mendapatkan perbaikan bug, SQL fitur Aurora My baru, atau untuk mengubah ke versi yang sama sekali baru dari mesin database yang mendasarinya. Bagian berikut menunjukkan caranya.

> **Note**
>
> Jenis peningkatan yang Anda lakukan bergantung pada durasi waktu henti klaster yang dapat Anda toleransi, jumlah pengujian verifikasi yang berencana Anda lakukan, seberapa penting perbaikan bug tertentu atau fitur baru untuk kasus penggunaan Anda, dan apakah Anda berencana akan sering melakukan peningkatan kecil atau peningkatan sesekali yang melewati beberapa versi perantara. Untuk setiap peningkatan, Anda dapat mengubah versi mayor, versi minor, dan tingkat patch untuk klaster Anda. Jika Anda tidak terbiasa dengan perbedaan antara Aurora versi utama SQL saya, versi minor, dan level patch, Anda dapat membaca informasi latar belakang di. Memeriksa Aurora Nomor versi saya SQL

> **Tip**
>
> Anda dapat meminimalkan waktu henti yang diperlukan untuk peningkatan klaster DB dengan menggunakan deployment blue/green. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

Topik

- Memutakhirkan versi minor atau level patch cluster Aurora SQL My DB
- Meningkatkan versi mayor klaster DB Amazon Aurora MySQL

## Memutakhirkan versi minor atau level patch cluster Aurora SQL My DB

Anda dapat menggunakan metode berikut untuk meningkatkan versi minor klaster DB atau untuk mem-patch klaster DB:

- Meningkatkan Aurora MySQL dengan mengubah versi mesin(untuk Aurora My SQL versi 2 dan 3)
- Mengaktifkan upgrade otomatis antara Aurora minor versi Saya SQL

Untuk informasi tentang cara patching zero-downtime dapat mengurangi gangguan selama proses peningkatan, lihat Menggunakan zero-downtime patching.

Untuk informasi tentang melakukan upgrade versi minor untuk klaster Aurora My SQL DB Anda, lihat topik berikut.

Topik


- Sebelum melakukan upgrade versi minor
- Pra-cek pemutakhiran versi minor untuk Aurora My SQL
- Meningkatkan Aurora MySQL dengan mengubah versi mesin
- Mengaktifkan upgrade otomatis antara Aurora minor versi Saya SQL
- Menggunakan zero-downtime patching
- Teknik peningkatan blue/green alternatif


## Sebelum melakukan upgrade versi minor

Kami menyarankan Anda melakukan tindakan berikut untuk mengurangi waktu henti selama upgrade versi minor:

- Pemeliharaan cluster Aurora DB harus dilakukan selama periode lalu lintas rendah. Gunakan Performance Insights untuk mengidentifikasi periode waktu ini agar dapat mengonfigurasi jendela pemeliharaan dengan benar. Untuk informasi selengkapnya tentang Performance Insights, lihat Memantau pemuatan DB dengan Performance Insights di Amazon. RDS Untuk informasi lebih lanjut tentang jendela pemeliharaan cluster DB,Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.
- Gunakan AWS SDKsyang mendukung backoff eksponensial dan jitter sebagai praktik terbaik. Untuk informasi lebih lanjut, lihat Exponential Backoff And Jitter.

## Pra-cek pemutakhiran versi minor untuk Aurora My SQL

Saat Anda memulai pemutakhiran versi minor, Amazon Aurora menjalankan prechecks secara otomatis.

Pra-pemeriksaan ini wajib dilakukan. Anda tidak dapat memilih untuk melewatinya. Pra-pemeriksaan menyediakan manfaat berikut:

- Hal ini memungkinkan Anda menghindari waktu henti yang tidak direncanakan selama peningkatan.
- Jika ada inkompatibilitas, Amazon Aurora akan mencegah peningkatan dan menyediakan log bagi Anda untuk mempelajarinya. Anda kemudian dapat menggunakan log untuk mempersiapkan

database Anda untuk upgrade dengan mengurangi ketidakcocokan. Untuk informasi rinci tentang menghapus ketidakcocokan, lihat Mempersiapkan instalasi Anda untuk upgrade dalam dokumentasi Saya. SQL

Pra-pemeriksaan berjalan sebelum instans DB dihentikan untuk peningkatan, sehingga instans tersebut tidak akan menyebabkan waktu henti ketika berjalan. Jika pra-pemeriksaan menemukan inkompatibilitas, Aurora secara otomatis membatalkan peningkatan sebelum instans DB dihentikan. Aurora juga menghasilkan peristiwa untuk inkompatibilitas. Untuk informasi selengkapnya tentang peristiwa Amazon Aurora, lihat Bekerja dengan pemberitahuan RDS acara Amazon.

Aurora mencatat informasi terperinci tentang setiap inkompatibilitas dalam file log `PrePatchCompatibility.log`. Dalam kebanyakan kasus, entri log menyertakan tautan ke SQL dokumentasi Saya untuk memperbaiki ketidakcocokan. Untuk informasi selengkapnya tentang format file log, lihat Melihat dan mencantumkan file log basis data.

Karena sifatnya, pra-pemeriksaan akan menganalisis objek di basis data Anda. Analisis ini mengakibatkan konsumsi sumber daya dan menambah waktu penyelesaian peningkatan.

Meningkatkan Aurora MySQL dengan mengubah versi mesin

Memutakhirkan versi minor dari cluster DB MySQL Aurora menerapkan perbaikan tambahan dan fitur baru ke cluster yang ada.

Peningkatan semacam ini berlaku untuk cluster Aurora MySQL di mana versi asli dan versi yang ditingkatkan keduanya memiliki versi utama Aurora MySQL yang sama, baik 2 atau 3. Prosesnya cepat dan mudah karena tidak memerlukan konversi apa pun untuk metadata Aurora MySQL atau penyusunan ulang data tabel Anda.

Anda melakukan upgrade semacam ini dengan memodifikasi versi mesin dari cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API. Misalnya, jika cluster Anda menjalankan Aurora MySQL 3.x, pilih versi 3.x yang lebih tinggi.

Jika Anda melakukan peningkatan kecil pada Database Global Aurora, tingkatkan semua cluster sekunder sebelum Anda memutakhirkan klaster utama.

> **Note**
>
> Untuk melakukan upgrade versi minor ke Aurora MySQL versi 3.04.* atau lebih tinggi, atau versi 2.12.*, gunakan proses berikut:

1. Hapus semua Wilayah sekunder dari klaster global. Ikuti langkah-langkahnya dalam Menghapus klaster dari basis data global Amazon Aurora.
2. Tingkatkan versi mesin Wilayah utama ke versi 3.04.* atau lebih tinggi, atau versi 2.12.*, sebagaimana berlaku. Ikuti langkah-langkahnya dalam To modify the engine version of a DB cluster.
3. Tambahkan Wilayah sekunder ke klaster global. Ikuti langkah-langkahnya dalam Menambahkan Wilayah AWS ke database global Amazon Aurora.

Untuk mengubah versi mesin klaster DB

- Dengan menggunakan konsol – Ubah properti klaster Anda. Di jendela Modifikasi klaster DB, ubah versi mesin Aurora MySQL dalam kotak Versi mesin DB. Jika Anda tidak memahami prosedur umum untuk mengubah klaster, ikuti petunjuk dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.
- Dengan menggunakan AWS CLI — Panggil modify-db-cluster AWS CLI perintah, dan tentukan nama cluster DB Anda untuk `--db-cluster-identifier` opsi dan versi mesin untuk `--engine-version` opsi tersebut.

  Misalnya, untuk meningkatkan ke Aurora MySQL versi 3.04.1, atur opsi ke. `--engine-version 8.0.mysql_aurora.3.04.1` Tentukan opsi `--apply-immediately` untuk segera memperbarui versi mesin untuk klaster DB Anda.
- Dengan menggunakan RDS API — Panggil operasi Modify DBCluster API, dan tentukan nama cluster DB Anda untuk `DBClusterIdentifier` parameter dan versi engine untuk `EngineVersion` parameter tersebut. Tetapkan parameter `ApplyImmediately` ke `true` untuk segera memperbarui versi mesin untuk klaster DB Anda.

Mengaktifkan upgrade otomatis antara Aurora minor versi Saya SQL

Untuk klaster Amazon Aurora My SQL DB, Anda dapat menentukan bahwa Aurora memutakhirkan cluster DB secara otomatis ke versi minor baru. Anda melakukannya dengan menyetel `AutoMinorVersionUpgrade` properti (Peningkatan versi minor otomatis di AWS Management Console) dari cluster DB.

Peningkatan otomatis terjadi selama periode pemeliharaan. Jika instans DB individu di klaster DB memiliki periode pemeliharaan yang berbeda dengan periode pemeliharaan klaster, periode pemeliharaan klaster akan diutamakan.

Upgrade versi minor otomatis tidak berlaku untuk jenis klaster Aurora My SQL berikut:

- Klaster yang merupakan bagian dari basis data global Aurora.
- Klaster yang memiliki replika lintas Wilayah.

Durasi pemadaman bervariasi tergantung pada beban kerja, ukuran cluster, jumlah data log biner, dan jika Aurora dapat menggunakan fitur zero-downtime patching (). ZDP Aurora mengaktifkan ulang klaster basis data, sehingga Anda mungkin mengalami periode ketidaktersediaan yang singkat sebelum melanjutkan penggunaan klaster Anda. Secara khusus, jumlah data log biner akan memengaruhi waktu pemulihan. Instans DB memproses data log biner selama pemulihan. Dengan demikian, volume tinggi data log biner akan menambah waktu pemulihan.

> **Note**
>
> Aurora hanya melakukan peningkatan otomatis jika semua instans DB di cluster DB Anda mengaktifkan pengaturan. `AutoMinorVersionUpgrade` Untuk informasi tentang cara mengaturnya, dan cara kerjanya saat diterapkan di tingkat cluster dan instance, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.
> Kemudian jika jalur pemutakhiran ada untuk instance cluster DB ke versi mesin DB minor yang telah `AutoUpgrade` disetel ke true, pemutakhiran akan dilakukan. `AutoUpgrade`Pengaturannya dinamis, dan diatur olehRDS.
> Peningkatan versi minor otomatis dilakukan ke versi minor default.

Anda dapat menggunakan CLI perintah seperti berikut ini untuk memeriksa status `AutoMinorVersionUpgrade` pengaturan untuk semua instans DB di cluster Aurora SQL My Anda.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Perintah ini menghasilkan output yang serupa dengan berikut:

```
[
```

```
  {
      "DBInstanceIdentifier": "db-t2-medium-instance",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-t2-small-original-size",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "instance-2020-05-01-2332",
      "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

Dalam contoh ini, opsi Aktifkan peningkatan versi minor otomatis dinonaktifkan untuk klaster DB `cluster-57-2020-06-03-6411`, karena dinonaktifkan untuk salah satu instans DB dalam klaster.

Menggunakan zero-downtime patching

Melakukan upgrade untuk Aurora SQL My DB cluster melibatkan kemungkinan pemadaman ketika database dimatikan dan saat sedang ditingkatkan. Secara default, jika Anda memulai peningkatan saat basis data sibuk, Anda akan kehilangan semua koneksi dan transaksi yang sedang diproses oleh klaster DB. Jika Anda menunggu hingga basis data idle untuk melakukan peningkatan, Anda mungkin harus menunggu lama.

Fitur zero-downtime patching (ZDP) mencoba, dengan upaya terbaik, untuk mempertahankan koneksi klien melalui peningkatan Aurora My. SQL Jika ZDP berhasil diselesaikan, sesi aplikasi dipertahankan dan mesin database restart saat upgrade sedang berlangsung. Pengaktifan ulang mesin basis data dapat menyebabkan penurunan throughput yang berlangsung selama beberapa detik hingga kira-kira satu menit.

ZDPtidak berlaku untuk yang berikut:

- Patch dan peningkatan sistem operasi (OS)
- Peningkatan versi mayor

ZDPtersedia untuk semua SQL versi Aurora My yang didukung dan kelas instans DB.

ZDPtidak didukung untuk Aurora Serverless v1 atau database global Aurora.

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Anda dapat melihat metrik atribut penting selama ZDP di log SQL kesalahan saya. Anda juga dapat melihat informasi tentang kapan Aurora My SQL menggunakan ZDP atau memilih untuk tidak menggunakan ZDP pada halaman Acara di halaman. AWS Management Console

Di Aurora My, SQL Aurora dapat melakukan patch zero-downtime apakah replikasi log biner diaktifkan atau tidak. Jika replikasi log biner diaktifkan, Aurora SQL My secara otomatis menjatuhkan koneksi ke target binlog selama operasi. ZDP Aurora My SQL secara otomatis terhubung kembali ke target binlog dan melanjutkan replikasi setelah restart selesai.

ZDPjuga berfungsi dalam kombinasi dengan peningkatan reboot di Aurora My. SQL Patching terhadap instans DB penulis akan secara otomatis menjalankan patching terhadap pembaca secara bersamaan. Setelah melakukan patching, Aurora memulihkan koneksi pada instans DB penulis dan pembaca.

ZDPmungkin tidak berhasil diselesaikan dalam kondisi berikut:

- Kueri atau transaksi berjalan lama sedang berlangsung. Jika Aurora dapat melakukan ZDP dalam kasus ini, setiap transaksi terbuka dibatalkan tetapi koneksi mereka dipertahankan.
- Tabel sementara, kunci pengguna, atau kunci tabel sedang digunakan, misalnya saat pernyataan bahasa definisi data (DDL) berjalan. Aurora menjatuhkan koneksi ini.
- Ada perubahan parameter tertunda.

Jika tidak ada jendela waktu yang sesuai untuk melakukan ZDP menjadi tersedia karena satu atau lebih dari kondisi ini, patch kembali ke perilaku standar.

Meskipun koneksi tetap utuh setelah ZDP operasi yang berhasil, beberapa variabel dan fitur diinisialisasi ulang. Jenis informasi berikut ini tidak dipertahankan selama pengaktifan ulang yang disebabkan oleh zero-downtime patching:

- Variabel global. Aurora memulihkan variabel sesi, tetapi tidak memulihkan variabel global setelah pengaktifan ulang.
- Variabel status. Secara khusus, nilai uptime yang dilaporkan oleh status engine diatur ulang setelah restart yang menggunakan ZDP mekanisme ZDR atau.
- `LAST_INSERT_ID`.
- Status `auto_increment` dalam memori untuk tabel. Status inkremen otomatis dalam memori diinisialisasi ulang. Untuk informasi selengkapnya tentang nilai kenaikan otomatis, lihat Panduan SQLReferensi Saya.
- Informasi diagnostik dari tabel `INFORMATION_SCHEMA` dan `PERFORMANCE_SCHEMA`. Informasi diagnostik ini juga muncul dalam output perintah seperti `SHOW PROFILE` dan `SHOW PROFILES`.

Aktivitas berikut yang berkaitan dengan pengaktifan ulang dengan nol waktu henti akan dilaporkan di halaman Peristiwa:

- Mencoba meningkatkan basis data dengan nol waktu henti.
- Mencoba meningkatkan basis data dengan nol waktu henti selesai. Peristiwa ini melaporkan berapa lama prosesnya berjalan. Peristiwa ini juga melaporkan berapa banyak koneksi yang dipertahankan selama pengaktifan ulang dan berapa banyak koneksi yang terputus. Anda dapat melihat log kesalahan basis data untuk melihat detail selengkapnya tentang apa yang terjadi selama pengaktifan ulang.

Teknik peningkatan blue/green alternatif

Dalam situasi tertentu, prioritas utama Anda adalah melakukan switchover langsung dari klaster lama ke klaster yang ditingkatkan. Dalam situasi seperti itu, Anda dapat menggunakan proses multistep yang menjalankan cluster side-by-side lama dan baru. Di sini, Anda mereplikasi data dari klaster lama ke klaster baru hingga Anda siap untuk mengambil alih klaster baru. Untuk detailnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

## Meningkatkan versi mayor klaster DB Amazon Aurora MySQL

Dalam nomor versi MySQL Aurora seperti 3.04.1, 3 mewakili versi utama. Aurora MySQL versi 2 kompatibel dengan MySQL 5.7. Aurora MySQL versi 3 kompatibel dengan MySQL 8.0.

Peningkatan antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama. Setelah peningkatan selesai, Anda mungkin juga memiliki pekerjaan lanjutan yang harus dilakukan. Misalnya, ini mungkin terjadi karena

perbedaan kompatibilitas SQL atau cara kerja fitur terkait MySQL tertentu. Atau, ini mungkin terjadi karena pengaturan parameter yang berbeda antara versi lama dan baru.

Daftar Isi


- Upgrade dari Aurora MySQL versi 2 ke versi 3
- Jalur peningkatan versi mayor Aurora MySQL
- Cara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL
- Merencanakan peningkatan versi mayor untuk klaster Aurora MySQL
  - Mensimulasikan peningkatan dengan mengkloning cluster DB Anda
  - Deployment Blue/Green
- Pra-cek pemutakhiran versi utama untuk Aurora My SQL
  - Proses precheck untuk Aurora My SQL
  - Prececk format log untuk Aurora My SQL
  - Contoh keluaran log precheck untuk Aurora My SQL
  - Prececk kinerja untuk Aurora My SQL
  - Ringkasan dari Community My SQL upgrade prechecks
  - Ringkasan dari Aurora My upgrade prechecks SQL
  - Referensi deskripsi precheck untuk Aurora MySQL
    - Kesalahan
      - MySQL memeriksa sebelumnya yang melaporkan kesalahan
      - Aurora MySQL memeriksa sebelumnya yang melaporkan kesalahan
    - Peringatan
      - MySQL memeriksa sebelumnya yang melaporkan peringatan
      - Aurora MySQL mengecek peringatan laporan itu
    - Pemberitahuan
    - Kesalahan, peringatan, atau pemberitahuan
- Cara melakukan peningkatan di tempat
  - Bagaimana peningkatan di tempat memengaruhi grup parameter untuk klaster
  - Perubahan pada properti klaster di antara versi Aurora MySQL
  - Peningkatan besar di tempat untuk basis data global
  - Upgrade di tempat untuk cluster DB dengan replika baca lintas wilayah

- Aurora Tutorial peningkatan di tempat SQL saya
- Menemukan alasan Aurora Kegagalan peningkatan versi SQL utama saya
- Pemecahan masalah untuk Aurora Peningkatan di tempat saya SQL
- Pembersihan pasca-upgrade untuk Aurora Versi saya 3 SQL
  - Indeks spasial

Upgrade dari Aurora MySQL versi 2 ke versi 3

Jika Anda memiliki klaster yang kompatibel dengan MySQL 5.7 dan ingin meningkatkan-nya ke klaster yang kompatibel dengan MySQL 8.0, Anda dapat melakukannya dengan menjalankan proses peningkatan pada klaster itu sendiri. Peningkatan semacam ini adalah peningkatan di tempat, berbeda dengan peningkatan yang Anda lakukan dengan membuat klaster baru. Teknik ini mempertahankan titik akhir dan karakteristik lain yang sama dari klaster. Peningkatan ini relatif cepat karena tidak memerlukan penyalinan semua data Anda ke volume klaster baru. Stabilitas ini membantu meminimalkan perubahan konfigurasi dalam aplikasi Anda. Hal ini juga membantu mengurangi jumlah pengujian untuk klaster yang ditingkatkan. Hal ini karena jumlah instans DB dan kelas instansnya semua tetap sama.

Mekanisme peningkatan di tempat mengharuskan klaster DB Anda dinonaktifkan saat operasi berlangsung. Aurora melakukan penonaktifan bersih dan menyelesaikan operasi yang tertunda seperti rollback transaksi dan pembersihan undo. Untuk informasi selengkapnya, lihat Cara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL.

Metode peningkatan di tempat ini praktis karena mudah dilakukan dan meminimalkan perubahan konfigurasi pada aplikasi terkait. Misalnya, tingkatkan di tempat mempertahankan titik akhir dan kumpulan instans DB untuk klaster Anda. Namun, waktu yang diperlukan untuk peningkatan di tempat dapat bervariasi tergantung pada properti skema Anda dan seberapa sibuk klasternya. Jadi, tergantung pada kebutuhan untuk cluster Anda, Anda dapat memilih di antara teknik upgrade:

- Peningkatan di tempat
- Penerapan Biru/Hijau
- Pemulihan snapshot

> **Note**
>
> Jika Anda menggunakan AWS CLI atau RDS API untuk metode upgrade pemulihan snapshot, Anda harus menjalankan operasi berikutnya untuk membuat instance DB penulis di cluster DB yang dipulihkan.

Untuk informasi umum tentang Aurora MySQL versi 3 dan fitur-fiturnya yang baru, lihat Aurora SQL Versi saya 3 kompatibel dengan My 8.0 SQL.

Untuk detail tentang merencanakan peningkatan, lihat Merencanakan peningkatan versi mayor untuk klaster Aurora MySQL dan Cara melakukan peningkatan di tempat.

Jalur peningkatan versi mayor Aurora MySQL

Tidak semua jenis atau versi klaster Aurora MySQL dapat menggunakan mekanisme peningkatan di tempat. Anda dapat mempelajari jalur peningkatan yang sesuai untuk setiap klaster Aurora MySQL dengan melihat tabel berikut.

| Jenis klaster DB Aurora MySQL | Apakah klaster dapat menggunakan peningkatan di tempat? | Tindakan |
|---|---|---|
| Kluster yang disediakan Aurora MySQL, versi 2 | Ya | Upgrade di tempat didukung untuk klaster MySQL MySQL 5.7 yang kompatibel dengan MySQL.<br><br>Untuk informasi tentang peningkatan ke Aurora MySQL versi 3, lihat Merencanakan peningkatan versi mayor untuk klaster Aurora MySQL dan Cara melakukan peningkatan di tempat. |
| Kluster yang disediakan Aurora MySQL, versi 3 | Tidak berlaku | Gunakan prosedur peningkatan versi minor untuk peningkatan di antara Aurora MySQL versi 3. |

| Jenis klaster DB Aurora MySQL | Apakah klaster dapat menggunakan peningkatan di tempat? | Tindakan |
|---|---|---|
| Aurora Serverless v1 cluster | Tidak berlaku | Aurora Serverless v1 didukung untuk Aurora MySQL hanya pada versi 2. |
| Aurora Serverless v2 cluster | Tidak berlaku | Aurora Serverless v2 didukung untuk Aurora MySQL hanya pada versi 3. |
| Klaster dalam basis data global Aurora | Ya | Untuk meningkatkan Aurora MySQL dari versi 2 ke versi 3, ikuti prosedur untuk melakukan peningkatan di tempat untuk klaster di basis data global Aurora. Lakukan peningkatan pada klaster global. Aurora meningkatkan klaster primer dan semua klaster sekunder dalam basis data global pada saat yang sama.<br><br>Jika Anda menggunakan API AWS CLI atau RDS, panggil `modify-global-cluster` perintah atau `ModifyGlobalCluster` operasi alih-alih `modify-db-cluster` atau`ModifyDBCluster` .<br><br>Anda dapat melakukan upgrade di tempat dari Aurora MySQL versi 2 ke versi 3 hanya `lower_case_table_names` jika parameter diatur ke default dan Anda reboot database global Anda. Untuk informasi selengkapnya, lihat Peningkatan versi utama. |
| Klaster kueri paralel | Ya | Anda dapat melakukan peningkatan di tempat. |

| Jenis klaster DB Aurora MySQL | Apakah klaster dapat menggunakan peningkatan di tempat? | Tindakan |
|---|---|---|
| Klaster yang merupakan target replikasi log biner | Mungkin | Jika replikasi log biner berasal dari klaster Aurora MySQL, Anda dapat melakukan peningkatan di tempat. Anda tidak dapat melakukan peningkatan jika replikasi log biner berasal dari instans DB RDS for MySQL atau instans DB MySQL on-premise. Dalam hal ini, Anda dapat melakukan peningkatan menggunakan mekanisme pemulihan snapshot. |
| Klaster dengan nol instans DB | Tidak | Menggunakan AWS CLI atau RDS API, Anda dapat membuat cluster Aurora MySQL tanpa instans DB yang terpasang. Dengan cara yang sama, Anda juga dapat menghapus semua instans DB dari klaster Aurora MySQL, tetapi membiarkan data dalam volume klaster tetap utuh. Meskipun sebuah klaster tidak memiliki instans DB, Anda tidak dapat melakukan peningkatan di tempat.<br><br>Mekanisme peningkatan memerlukan instans penulis di klaster untuk melakukan konversi pada tabel sistem, file data, dan sebagainya. Dalam hal ini, gunakan AWS CLI atau RDS API untuk membuat instance penulis untuk cluster. Kemudian, Anda dapat melakukan peningkatan di tempat. |
| Klaster dengan pelacakan mundur diaktifkan | Ya | Anda dapat melakukan upgrade di tempat untuk cluster Aurora MySQL yang menggunakan fitur Backtrack. Namun, setelah peningkatan, Anda tidak dapat melacak mundur klaster ke waktu sebelum peningkatan. |

Cara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL

Aurora MySQL melakukan peningkatan versi mayor sebagai proses multitahap. Anda dapat memeriksa status peningkatan saat ini. Beberapa langkah peningkatan juga memberikan informasi progres. Seiring setiap tahap dimulai, Aurora MySQL akan mencatat sebuah peristiwa. Anda dapat memeriksa peristiwa yang terjadi di halaman Peristiwa di konsol RDS. Untuk informasi selengkapnya tentang menggunakan peristiwa, lihat Bekerja dengan pemberitahuan RDS acara Amazon.

> **Important**
>
> Setelah proses dimulai, proses ini berjalan sampai peningkatan berhasil atau gagal. Anda tidak dapat membatalkan peningkatan saat sedang berlangsung. Jika peningkatan gagal, Aurora akan mengembalikan semua perubahan dan klaster Anda akan memiliki versi mesin, metadata, dan karakteristik lainnya yang sama seperti sebelumnya.

Proses peningkatan terdiri dari tiga tahap:

1. Aurora melakukan serangkaian pra-pemeriksaan sebelum memulai proses peningkatan. Klaster Anda terus berjalan saat Aurora melakukan pemeriksaan ini. Misalnya, klaster tidak dapat memiliki transaksi XA apa pun dalam status siap atau memproses pernyataan bahasa definisi data (DDL). Misalnya, Anda mungkin perlu menutup aplikasi yang mengirimkan jenis pernyataan SQL tertentu. Atau, Anda mungkin hanya perlu menunggu sampai pernyataan berjalan lama tertentu diselesaikan. Kemudian, coba mulai peningkatan lagi. Beberapa pemeriksaan akan menguji kondisi yang tidak mencegah peningkatan, tetapi mungkin akan membuat peningkatan memakan waktu lama.

   Jika Aurora mendeteksi bahwa kondisi yang diperlukan tidak terpenuhi, ubah kondisi yang diidentifikasi dalam detail peristiwa. Ikuti panduan dalam Pemecahan masalah untuk Aurora Peningkatan di tempat saya SQL. Jika Aurora mendeteksi kondisi yang mungkin menyebabkan peningkatan yang lambat, rencanakan untuk memantau peningkatan dalam waktu lama.
2. Aurora akan membuat klaster Anda menjadi offline. Kemudian, Aurora melakukan serangkaian pengujian serupa seperti pada tahap sebelumnya untuk memastikan bahwa tidak ada masalah baru yang muncul selama proses penonaktifan. Jika Aurora mendeteksi kondisi apa pun pada saat ini yang akan mencegah peningkatan, Aurora akan membatalkan peningkatan dan membuat klaster kembali online. Dalam hal ini, konfirmasikan jika kondisi tidak lagi berlaku dan mulai peningkatan lagi.

3. Aurora membuat snapshot dari volume klaster Anda. Misalkan Anda menemukan kompatibilitas atau jenis masalah lainnya setelah peningkatan selesai. Atau misalkan Anda ingin melakukan pengujian menggunakan klaster asli dan klaster yang ditingkatkan. Dalam kasus tersebut, Anda dapat memulihkan dari snapshot ini untuk membuat klaster baru dengan versi mesin asli dan data asli.

   > **Tip**
   >
   > Snapshot ini adalah snapshot manual. Namun, Aurora dapat membuatnya dan melanjutkan proses peningkatan bahkan jika Anda telah mencapai kuota untuk snapshot manual. Snapshot ini tetap ada secara permanen (jika diperlukan) hingga Anda menghapusnya. Setelah Anda menyelesaikan semua pengujian pasca-peningkatan, Anda dapat menghapus snapshot ini untuk meminimalkan biaya penyimpanan.

4. Aurora akan mengkloning volume klaster Anda. Kloning adalah operasi cepat yang tidak memerlukan penyalinan data tabel yang sebenarnya. Jika Aurora mengalami masalah selama peningkatan, ia kembali ke data asli dari volume klaster kloning dan membawa klaster kembali online. Volume yang dikloning sementara selama peningkatan tidak memiliki batas yang biasanya berlaku pada jumlah klon untuk satu volume klaster.
5. Aurora melakukan penonaktifan bersih untuk instans DB penulis. Selama penonaktifan bersih, progres peristiwa dicatat setiap 15 menit untuk operasi berikut. Anda dapat memeriksa peristiwa yang terjadi di halaman Peristiwa di konsol RDS.
   - Aurora membersihkan catatan undo untuk baris versi lama.
   - Aurora mengembalikan setiap transaksi yang tidak dikomit.
6. Aurora meningkatkan versi mesin pada instans DB penulis:
   - Aurora menginstal biner untuk versi mesin baru pada instans DB penulis.
   - Aurora menggunakan instans DB penulis untuk meningkatkan data Anda ke format yang kompatibel dengan MySQL 5.7. Selama tahap ini, Aurora memodifikasi tabel sistem dan melakukan konversi lain yang memengaruhi data dalam volume klaster Anda. Secara khusus, Aurora meningkatkan metadata partisi di tabel sistem agar kompatibel dengan format partisi MySQL 5.7. Tahap ini dapat memakan waktu lama jika tabel di klaster Anda memiliki sejumlah besar partisi.

     Jika terjadi kesalahan selama tahap ini, Anda dapat menemukan detailnya dalam log kesalahan MySQL. Setelah tahap ini dimulai, jika proses peningkatan gagal karena alasan apa pun, Aurora akan mengembalikan data asli dari volume klaster yang dikloning.

7. Aurora meningkatkan versi mesin pada instans DB pembaca.
8. Proses peningkatan selesai. Aurora mencatat peristiwa akhir untuk menunjukkan bahwa proses peningkatan berhasil selesai. Sekarang klaster DB Anda menjalankan versi mayor baru.

### Merencanakan peningkatan versi mayor untuk klaster Aurora MySQL

Untuk membantu Anda memutuskan waktu dan pendekatan yang tepat untuk meningkatkan, Anda dapat mempelajari perbedaan antara Aurora MySQL versi 3 dan lingkungan Anda saat ini:

- Jika Anda mengonversi dari RDS untuk MySQL 8.0 atau MySQL 8.0 Community Edition, lihat. Membandingkan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition
- Jika Anda meningkatkan dari Aurora MySQL versi 2, RDS untuk MySQL 5.7, atau komunitas MySQL 5.7, lihat. Membandingkan Aurora SQL Versi saya 2 dan Aurora Versi saya 3 SQL
- Buat versi baru yang kompatibel dengan MySQL 8.0 untuk setiap grup parameter kustom. Terapkan nilai parameter kustom yang diperlukan ke grup parameter baru. Baca Perubahan parameter untuk Aurora Versi saya 3 SQL untuk mempelajari tentang perubahan parameter.
- Tinjau definisi skema dan objek basis data Aurora MySQL versi 2 untuk penggunaan kata kunci baru yang dicadangkan, yang diperkenalkan di MySQL 8.0 Community Edition. Lakukan hal ini sebelum Anda meningkatkan. Untuk informasi selengkapnya, lihat MySQL 8.0 New Keywords and Reserved Words dalam dokumentasi MySQL.

Anda juga dapat menemukan lebih banyak pertimbangan dan tips peningkatan khusus MySQL dalam Changes in MySQL 8.0 dalam Panduan Referensi MySQL. Misalnya, Anda dapat menggunakan perintah `mysqlcheck --check-upgrade` untuk menganalisis basis data Aurora MySQL yang ada dan mengidentifikasi potensi masalah peningkatan.

> **Note**
>
> Sebaiknya gunakan kelas instans DB yang lebih besar saat meningkatkan ke Aurora MySQL versi 3 menggunakan teknik peningkatan atau pemulihan snapshot di tempat. Contohnya adalah db.r5.24xlarge dan db.r6g.16xlarge. Hal ini membantu menyelesaikan proses peningkatan lebih cepat dengan menggunakan sebagian besar kapasitas CPU yang tersedia pada instans DB. Anda dapat mengubah ke kelas instans DB yang Anda inginkan setelah peningkatan versi mayor selesai.

Setelah Anda menyelesaikan peningkatan itu sendiri, Anda dapat mengikuti prosedur pasca-peningkatan dalam Pembersihan pasca-upgrade untuk Aurora Versi saya 3 SQL. Terakhir, uji fungsionalitas dan performa aplikasi Anda.

Jika Anda mengonversi dari RDS dari MySQL atau MySQL komunitas, ikuti prosedur migrasi yang dijelaskan di. Migrasi data ke cluster Amazon Aurora My DB SQL Dalam beberapa kasus, Anda mungkin menggunakan replikasi log biner untuk menyinkronkan data Anda dengan klaster Aurora MySQL versi 3 sebagai bagian dari migrasi. Jika demikian, sistem sumber harus menjalankan versi yang kompatibel dengan cluster DB target Anda.

Untuk memastikan aplikasi dan prosedur administrasi Anda berjalan dengan lancar setelah meningkatkan klaster di antara versi mayor, lakukan beberapa perencanaan dan persiapan sebelumnya. Untuk melihat jenis kode manajemen apa yang akan diperbarui untuk AWS CLI skrip atau aplikasi berbasis API RDS, lihat. Bagaimana peningkatan di tempat memengaruhi grup parameter untuk klaster Lihat juga Perubahan pada properti klaster di antara versi Aurora MySQL.

Untuk mempelajari masalah apa yang mungkin Anda temui selama peningkatan, lihatPemecahan masalah untuk Aurora Peningkatan di tempat saya SQL. Untuk masalah yang mungkin menyebabkan peningkatan memakan waktu lama, Anda dapat menguji kondisi tersebut terlebih dahulu dan memperbaikinya.

> **Note**
>
> Upgrade di tempat melibatkan mematikan cluster DB Anda saat operasi berlangsung. Aurora MySQL melakukan shutdown bersih dan menyelesaikan operasi luar biasa seperti undo purge. Upgrade mungkin memakan waktu lama jika ada banyak catatan yang akan diurungkan untuk dibersihkan. Kami merekomendasikan melakukan peningkatan hanya setelah panjang daftar riwayat (HLL) rendah. Nilai yang dapat diterima secara umum untuk HLL adalah 100.000 atau kurang. Untuk informasi lebih lanjut, lihat posting blog ini.

### Mensimulasikan peningkatan dengan mengkloning cluster DB Anda

Anda dapat memeriksa kompatibilitas aplikasi, performa, prosedur pemeliharaan, dan pertimbangan serupa untuk klaster yang ditingkatkan. Untuk melakukannya, Anda dapat melakukan simulasi peningkatan sebelum melakukan peningkatan yang sebenarnya. Teknik ini dapat sangat berguna untuk klaster produksi. Di sini, penting untuk meminimalkan waktu henti dan menyiapkan klaster yang telah ditingkatkan agar siap digunakan segera setelah peningkatan selesai.

Gunakan langkah-langkah berikut:

1. Membuat klon dari klaster asli. Ikuti prosedur dalam Mengkloning volume untuk klaster DB Amazon Aurora.
2. Siapkan satu set instans DB penulis dan pembaca yang serupa seperti di klaster asli.
3. Melakukan peningkatan di tempat terhadap klaster yang dikloning. Ikuti prosedur dalam Cara melakukan peningkatan di tempat.

   Mulai peningkatan segera setelah membuat klon. Dengan demikian, volume klaster masih identik dengan status klaster aslinya. Jika klon dalam keadaan idle sebelum Anda melakukan peningkatan, Aurora akan melakukan proses pembersihan basis data di latar belakang. Dalam hal ini, tingkatkan klon bukanlah simulasi yang akurat untuk peningkatan klaster asli.
4. Uji kompatibilitas aplikasi, performa, prosedur administrasi, dan sebagainya, menggunakan klaster yang dikloning.
5. Jika Anda mengalami masalah apa pun, sesuaikan rencana peningkatan Anda untuk mengantisipasinya. Misalnya, sesuaikan kode aplikasi apa pun agar kompatibel dengan kumpulan fitur dari versi yang lebih tinggi. Perkirakan berapa lama waktu yang dibutuhkan untuk peningkatan berdasarkan jumlah data di klaster Anda. Anda juga dapat memilih untuk menjadwalkan peningkatan pada saat klaster tidak sibuk.
6. Setelah Anda yakin bahwa aplikasi dan beban kerja Anda berfungsi dengan baik dengan klaster uji, Anda dapat melakukan peningkatan di tempat untuk klaster produksi Anda.
7. Berusahalah untuk meminimalkan total waktu henti klaster Anda selama peningkatan versi mayor. Untuk melakukannya, pastikan bahwa beban kerja di klaster rendah atau nol pada saat peningkatan. Secara khusus, pastikan bahwa tidak ada transaksi berjalan lama yang sedang berlangsung saat Anda memulai peningkatan.

### Deployment Blue/Green

Dalam situasi tertentu, prioritas utama Anda adalah melakukan switchover langsung dari klaster lama ke klaster yang ditingkatkan. Dalam situasi seperti itu, Anda dapat menggunakan proses multistep yang menjalankan cluster side-by-side lama dan baru. Di sini, Anda mereplikasi data dari klaster lama ke klaster baru hingga Anda siap untuk mengambil alih klaster baru. Untuk detailnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

Pra-cek pemutakhiran versi utama untuk Aurora My SQL

Memutakhirkan My SQL dari satu versi utama ke versi utama lainnya, seperti beralih dari My SQL 5.7 ke My SQL 8.0, melibatkan beberapa perubahan arsitektur signifikan yang memerlukan perencanaan dan persiapan yang matang. Tidak seperti upgrade versi minor di mana fokusnya terutama pada memperbarui perangkat lunak mesin database dan dalam beberapa kasus tabel sistem, SQL upgrade utama My sering memperkenalkan perubahan mendasar pada bagaimana database menyimpan dan mengelola metadatanya.

Untuk membantu Anda mengidentifikasi ketidakcocokan tersebut, saat memutakhirkan dari Aurora SQL My versi 2 ke versi 3, Aurora menjalankan pemeriksaan kompatibilitas pemutakhiran (prechecks) secara otomatis untuk memeriksa objek di cluster database Anda dan mengidentifikasi ketidakcocokan yang diketahui yang dapat memblokir peningkatan dari proses. Untuk detail tentang Aurora My SQL Prececks, lihat. Referensi deskripsi precheck untuk Aurora MySQL Prececks Aurora berjalan selain yang dijalankan oleh utilitas pemeriksa SQL pemutakhiran Komunitas Saya.

Pra-pemeriksaan ini wajib dilakukan. Anda tidak dapat memilih untuk melewatinya. Pra-pemeriksaan menyediakan manfaat berikut:

- Mereka dapat mengurangi kemungkinan mengalami kegagalan peningkatan yang dapat menyebabkan waktu henti yang diperpanjang.
- Jika ada ketidakcocokan, Amazon Aurora mencegah peningkatan berjalan dan menyediakan log bagi Anda untuk mempelajarinya. Anda kemudian dapat menggunakan log untuk mempersiapkan database Anda untuk upgrade ke versi 3 dengan menyelesaikan ketidakcocokan. Untuk informasi rinci tentang menyelesaikan ketidakcocokan, lihat Mempersiapkan instalasi Anda untuk upgrade di SQL dokumentasi Saya dan Upgrade ke My 8.0? SQL Inilah yang perlu Anda ketahui... di Blog SQL Server Saya.

  Untuk informasi selengkapnya tentang memutakhirkan ke My SQL 8.0, lihat Memutakhirkan Milik Saya SQL di dokumentasi Saya. SQL

Prececks berjalan sebelum cluster DB Anda diambil offline untuk upgrade versi utama. Jika pra-pemeriksaan menemukan inkompatibilitas, Aurora secara otomatis membatalkan peningkatan sebelum instans DB dihentikan. Aurora juga menghasilkan peristiwa untuk inkompatibilitas. Untuk informasi selengkapnya tentang peristiwa Amazon Aurora, lihat Bekerja dengan pemberitahuan RDS acara Amazon.

Setelah prechecks selesai, Aurora mencatat informasi rinci tentang setiap ketidakcocokan dalam file. `upgrade-prechecks.log` Dalam kebanyakan kasus, entri log menyertakan tautan ke SQL dokumentasi Saya untuk memperbaiki ketidakcocokan. Untuk informasi selengkapnya tentang format file log, lihat Melihat dan mencantumkan file log basis data.

> **Note**
>
> Karena sifatnya, pra-pemeriksaan akan menganalisis objek di basis data Anda. Analisis ini mengakibatkan konsumsi sumber daya dan menambah waktu penyelesaian peningkatan. Untuk informasi selengkapnya tentang pertimbangan kinerja precheck, lihat. Proses precheck untuk Aurora My SQL

Daftar Isi


- Proses precheck untuk Aurora My SQL
- Prececk format log untuk Aurora My SQL
- Contoh keluaran log precheck untuk Aurora My SQL
- Prececk kinerja untuk Aurora My SQL
- Ringkasan dari Community My SQL upgrade prechecks
- Ringkasan dari Aurora My upgrade prechecks SQL
- Referensi deskripsi precheck untuk Aurora MySQL
  - Kesalahan
    - MySQL memeriksa sebelumnya yang melaporkan kesalahan
    - Aurora MySQL memeriksa sebelumnya yang melaporkan kesalahan
  - Peringatan
    - MySQL memeriksa sebelumnya yang melaporkan peringatan
    - Aurora MySQL mengecek peringatan laporan itu
  - Pemberitahuan
  - Kesalahan, peringatan, atau pemberitahuan

Proses precheck untuk Aurora My SQL

Seperti yang dijelaskan sebelumnya, proses SQL upgrade Aurora My melibatkan menjalankan pemeriksaan kompatibilitas (prechecks) pada database Anda sebelum upgrade versi utama dapat dilanjutkan.

Untuk peningkatan di tempat, precheck berjalan pada instance DB penulis Anda saat sedang online. Jika precheck berhasil, upgrade berlangsung. Jika kesalahan ditemukan, mereka masuk ke `upgrade-prechecks.log` file dan pemutakhiran dibatalkan. Sebelum mencoba upgrade lagi, selesaikan kesalahan yang dikembalikan dalam `upgrade-prechecks.log` file.

Untuk peningkatan snapshot-restore, precheck berjalan selama proses pemulihan. Jika berhasil, database Anda akan meng-upgrade ke versi Aurora SQL My yang baru. Jika kesalahan ditemukan, mereka masuk ke `upgrade-prechecks.log` file dan pemutakhiran dibatalkan. Sebelum mencoba upgrade lagi, selesaikan kesalahan yang dikembalikan dalam `upgrade-prechecks.log` file.

Untuk informasi selengkapnya, silakan lihat Menemukan alasan Aurora Kegagalan peningkatan versi SQL utama saya dan Referensi deskripsi precheck untuk Aurora MySQL.

Untuk memantau status precheck, Anda dapat melihat peristiwa berikut di cluster DB Anda.

| Status precheck | Pesan peristiwa | Tindakan |
| --- | --- | --- |
| Dimulai | Persiapan peningkatan sedang berlangsung: Memulai prapecek peningkatan online. | Tidak ada |
| Failed | Kluster basis data berada dalam keadaan yang tidak dapat ditingkatkan: Prakecek pemutakhiran gagal. Untuk detail selengkapnya, lihat file upgrade-prechecks.log.<br><br>Untuk informasi selengkap nya tentang pemecahan masalah penyebab kegagalan pemutakhiran, lihat | `upgrade-prechecks. log` Tinjau kesalahan.<br><br>Memperbaiki kesalahan.<br><br>Coba lagi upgrade. |

| Status precheck | Pesan peristiwa | Tindakan |
| --- | --- | --- |
| | https://docs.aws.amazon.com/AmazonRDS/terbaru//AuroraUserGuide.upgrade. TroubleShooting.html AuroraMy SQL | |
| Berhasil | Persiapan peningkatan sedang berlangsung: Prakecek peningkatan online yang telah selesai. | Precheck berhasil tanpa kesalahan yang dikembalikan.<br><br>Tinjau `upgrade-prechecks.log` peringatan dan pemberitahuan. |

Untuk informasi selengkapnya tentang melihat acara, lihatMelihat RDS acara Amazon.

Prececk format log untuk Aurora My SQL

Setelah pemeriksaan kompatibilitas pemutakhiran (prechecks) selesai, Anda dapat meninjau `upgrade-prechecks.log` file tersebut. File log berisi hasil, objek yang terpengaruh, dan informasi remediasi untuk setiap precheck.

Kesalahan memblokir peningkatan. Anda harus menyelesaikannya sebelum mencoba kembali peningkatan.

Peringatan dan pemberitahuan kurang penting, tetapi kami tetap menyarankan Anda untuk memeriksanya dengan cermat untuk memastikan bahwa tidak ada masalah kompatibilitas dengan beban kerja aplikasi. Atasi masalah yang teridentifikasi segera.

File log memiliki format berikut:

- `targetVersion`— Versi My SQL -kompatibel dari upgrade Aurora SQL My.
- `auroraServerVersion`— SQL Versi Aurora My di mana precheck dijalankan.
- `auroraTargetVersion`— SQL Versi Aurora My yang Anda tingkatkan.
- `checksPerformed`— Berisi daftar prechecks yang dilakukan.
- `id`— Nama precheck yang sedang dijalankan.
- `title`— Deskripsi precheck yang sedang dijalankan.

- `status`— Ini tidak menunjukkan apakah precheck berhasil atau gagal, tetapi menunjukkan status kueri precheck:
  - `OK`— Kueri precheck berjalan dan diselesaikan dengan sukses.
  - `ERROR`— Kueri precheck gagal dijalankan. Hal ini dapat terjadi karena masalah seperti kendala sumber daya, restart instance yang tidak terduga, atau kueri precheck kompatibilitas terganggu.

    Untuk informasi lebih lanjut, lihat contoh ini.
- `description`— Gambaran umum tentang ketidakcocokan, dan bagaimana memperbaiki masalah.
- `documentationLink`— Jika berlaku, tautan ke SQL dokumentasi Aurora Saya SQL atau Saya yang relevan dicatat di sini. Untuk informasi selengkapnya, lihat Referensi deskripsi precheck untuk Aurora MySQL.
- `detectedProblems`— Jika precheck mengembalikan kesalahan, peringatan, atau pemberitahuan, ini menunjukkan rincian ketidakcocokan, dan objek yang tidak kompatibel jika berlaku:
  - `level`— Tingkat ketidakcocokan yang terdeteksi oleh precheck. Level yang valid adalah sebagai berikut:
    - `Error`— Upgrade tidak dapat dilanjutkan sampai Anda menyelesaikan ketidakcocokan.
    - `Warning`— Upgrade dapat dilanjutkan, tetapi objek, sintaks, atau konfigurasi yang tidak digunakan lagi terdeteksi. Tinjau peringatan dengan hati-hati, dan selesaikan segera untuk menghindari masalah di rilis mendatang.
    - `Notice`— Upgrade dapat dilanjutkan, tetapi objek, sintaks, atau konfigurasi yang tidak digunakan lagi terdeteksi. Tinjau pemberitahuan dengan cermat, dan selesaikan segera untuk menghindari masalah di rilis mendatang.
  - `dbObject`— Nama objek database di mana ketidakcocokan terdeteksi.
  - `description`— Penjelasan rinci tentang ketidakcocokan, dan bagaimana memperbaiki masalah.
- `errorCount`— Jumlah kesalahan ketidakcocokan yang terdeteksi. Ini memblokir peningkatan.
- `warningCount`— Jumlah peringatan ketidakcocokan yang terdeteksi. Ini tidak memblokir peningkatan, tetapi segera mengatasinya untuk menghindari masalah di rilis mendatang.
- `noticeCount`— Jumlah pemberitahuan ketidakcocokan yang terdeteksi. Ini tidak memblokir peningkatan, segera mengatasinya untuk menghindari masalah di rilis mendatang.
- `Summary`— Ringkasan kesalahan kompatibilitas precheck, peringatan, dan jumlah pemberitahuan.

Contoh keluaran log precheck untuk Aurora My SQL

Contoh berikut menunjukkan output log precheck yang mungkin Anda lihat. Untuk detail prechecks yang dijalankan, lihatReferensi deskripsi precheck untuk Aurora MySQL.

Prececk status OK, tidak ada ketidakcocokan yang terdeteksi

Query precheck berhasil diselesaikan. Tidak ada ketidakcocokan yang terdeteksi.

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinytext",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny text columns",
  "status": "OK",
  "detectedProblems": []
},
```

Prececk status OK, kesalahan terdeteksi

Query precheck berhasil diselesaikan. Satu kesalahan terdeteksi.

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexes",
  "status": "OK",
  "description": "Consider dropping the prefix indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test25.sbtest1",
        "description": "Table `test25`.`sbtest1` has an index `idx_t1` on geometry
 column/s. Mysql 8.0 does not support this type of index on a geometry column
 https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run 'DROP
 INDEX `idx_t1` ON `test25`.`sbtest1`;"
      },
 }
```

Prececk status OK, peringatan terdeteksi

Peringatan dapat dikembalikan ketika precheck berhasil atau tidak berhasil.

Di sini kueri precheck berhasil diselesaikan. Dua peringatan terdeteksi.

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
    ]
}
```


Status precheckERROR, tidak ada ketidakcocokan yang dilaporkan

Kueri precheck gagal dengan kesalahan, sehingga ketidakcocokan tidak dapat diverifikasi.


```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "ERROR",
  "description": "Can't connect to MySQL server on 'localhost:3306' (111) at
 13/08/2024 12:22:20 UTC. This failure can occur due to low memory available on the
 instance for executing upgrade prechecks. Please check 'FreeableMemory' Cloudwatch
 metric to verify the available memory on the instance while executing prechecks. If
 instance ran out of memory, we recommend to retry the upgrade on a higher instance
 class."
}
```

Kegagalan ini dapat terjadi karena restart instance yang tidak terduga atau kueri precheck kompatibilitas terganggu pada database saat berjalan. Misalnya, pada kelas instans DB yang lebih kecil, Anda mungkin mengalami ini ketika memori yang tersedia pada instance berjalan rendah.

Anda dapat menggunakan CloudWatch metrik `FreeableMemory` Amazon untuk memverifikasi memori yang tersedia pada instance saat menjalankan prechecks. Jika instance kehabisan memori, kami sarankan untuk mencoba kembali peningkatan pada kelas instans DB yang lebih besar. Dalam beberapa kasus, Anda dapat menggunakan penerapan Biru/Hijau Ini memungkinkan prechecks dan upgrade untuk berjalan pada cluster DB "hijau" independen dari beban kerja produksi, yang juga mengkonsumsi sumber daya sistem.

Untuk informasi selengkapnya, lihat Memecahkan masalah penggunaan memori untuk database Aurora MySQL.

Ringkasan precheck, satu kesalahan dan tiga peringatan terdeteksi

Precheck kompatibilitas juga berisi informasi tentang sumber dan target Aurora versi SQL Saya, dan ringkasan kesalahan, peringatan, dan jumlah pemberitahuan di akhir output precheck.

Misalnya, output berikut menunjukkan bahwa upaya telah dilakukan untuk meningkatkan dari Aurora My SQL 2.11.6 ke Aurora My 3.07.1. SQL Upgrade mengembalikan satu kesalahan, tiga peringatan, dan tidak ada pemberitahuan. Karena pemutakhiran tidak dapat dilanjutkan ketika kesalahan dikembalikan, Anda harus menyelesaikan masalah routineSyntaxCheckkompatibilitas dan mencoba lagi pemutakhiran.

```
{
  "serverAddress": "/tmp%2Fmysql.sock",
  "serverVersion": "5.7.12 - MySQL Community Server (GPL)",
  "targetVersion": "8.0.36",
  "auroraServerVersion": "2.11.6",
  "auroraTargetVersion": "3.07.1",
  "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
  "checksPerformed": [{
      ... output for each individual precheck ...
      .
      .
      {
        "id": "oldTemporalCheck",
        "title": "Usage of old temporal type",
        "status": "OK",
          "detectedProblems": []
```

```
      },
      {
        "id": "routinesSyntaxCheck",
        "title": "MySQL 8.0 syntax check for routine-like objects",
        "status": "OK",
        "description": "The following objects did not pass a syntax check with
 the latest MySQL 8.0 grammar. A common reason is that they reference names that
 conflict with new reserved keywords. You must update these routine definitions and
 `quote` any such references before upgrading.",
        "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
        "detectedProblems": [{
            "level": "Error",
            "dbObject": "test.select_res_word",
            "description": "at line 2,18: unexpected token 'except'"
        }]
      },
      .
      .
      .
      {
        "id": "zeroDatesCheck",
        "title": "Zero Date, Datetime, and Timestamp values",
        "status": "OK",
        "description": "Warning: By default zero date/datetime/timestamp values
 are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are
 included in SQL_MODE by default. These modes should be used with strict mode as
 they will be merged with strict mode in a future release. If you do not include
 these modes in your SQL_MODE setting, you are able to insert date/datetime/
timestamp values that contain zeros. It is strongly advised to replace zero values
 with valid ones, as they may not work correctly in the future.",
        "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
        "detectedProblems": [{
            "level": "Warning",
            "dbObject": "global.sql_mode",
            "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
            },
            {
            "level": "Warning",
            "dbObject": "session.sql_mode",
            "description": " of 8 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
            }
          ]
```

```
        },
        .
        .
        .
  }],
  "errorCount": 1,
  "warningCount": 3,
  "noticeCount": 0,
  "Summary": "1 errors were found. Please correct these issues before upgrading to
 avoid compatibility issues."
}
```

Prececk kinerja untuk Aurora My SQL

Prakecek kompatibilitas dijalankan sebelum instans DB diambil offline untuk peningkatan, jadi dalam keadaan biasa mereka tidak menyebabkan downtime instans DB saat berjalan. Namun, mereka dapat memengaruhi beban kerja aplikasi yang berjalan pada instance DB penulis. Prechecks mengakses kamus data melalui tabel information_schema, yang bisa lambat jika ada banyak objek database. Pertimbangkan faktor-faktor berikut:

- Durasi precheck bervariasi dengan jumlah objek database seperti tabel, kolom, rutinitas, dan kendala. Cluster DB dengan sejumlah besar objek dapat memakan waktu lebih lama untuk dijalankan.

  Misalnya, removedFunctionsCheckdapat memakan waktu lebih lama dan menggunakan lebih banyak sumber daya berdasarkan jumlah objek yang disimpan.
- Untuk peningkatan di tempat, menggunakan kelas instans DB yang lebih besar (misalnya, db.r5.24xlarge atau db.r6g.16xlarge) dapat membantu peningkatan selesai lebih cepat dengan menggunakan lebih banyak. CPU Anda dapat berhemat setelah upgrade.
- Kueri `information_schema` di beberapa database bisa lambat, terutama dengan banyak objek dan pada instance DB yang lebih kecil. Dalam kasus seperti itu, pertimbangkan untuk menggunakan kloning, pemulihan snapshot, atau penerapan Biru/Hijau untuk peningkatan.
- Penggunaan sumber daya precheck (CPU, memori) dapat meningkat dengan lebih banyak objek, yang menyebabkan waktu berjalan lebih lama pada instans DB yang lebih kecil. Dalam kasus seperti itu, pertimbangkan pengujian menggunakan kloning, pemulihan snapshot, atau penerapan Biru/Hijau untuk peningkatan.

Jika precheck gagal karena kurangnya sumber daya, Anda dapat mendeteksi ini di log precheck menggunakan output status:

```
"status": "ERROR",
```

Untuk informasi selengkapnya, silakan lihat Cara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL dan Merencanakan peningkatan versi mayor untuk klaster Aurora MySQL.

Ringkasan dari Community My SQL upgrade prechecks

Berikut ini adalah daftar umum ketidakcocokan antara My SQL 5.7 dan 8.0:

- Cluster DB yang SQL kompatibel dengan My 5.7 tidak boleh menggunakan fitur yang tidak didukung di My 8.0. SQL

  Untuk informasi selengkapnya, lihat Fitur yang dihapus di SQL 8.0 Saya di SQL dokumentasi Saya.
- Tidak boleh ada pelanggaran terkait kata kunci atau kata yang digunakan sistem. Beberapa kata kunci mungkin dicadangkan di My SQL 8.0 yang sebelumnya tidak dipesan.

  Untuk informasi selengkapnya, lihat Kata kunci dan kata cadangan dalam SQL dokumentasi Saya.
- Untuk dukungan Unicode yang lebih baik, pertimbangkan untuk mengonversi objek yang menggunakan set karakter `utf8mb3` untuk menggunakan set karakter `utf8mb4`. Set karakter `utf8mb3` sudah tidak digunakan lagi. Selain itu, pertimbangkan untuk menggunakan `utf8mb4` untuk referensi set karakter, bukan `utf8`, karena saat ini `utf8` adalah alias untuk set karakter `utf8mb3`.

  Untuk informasi selengkapnya, lihat Kumpulan karakter utf8mb3 (pengkodean unicode 3-byte UTF-8) di dokumentasi Saya. SQL
- Tidak boleh ada tabel InnoDB dengan format baris non-default.
- Tidak boleh ada atribut jenis panjang `ZEROFILL` atau `display`.
- Tidak boleh ada tabel partisi yang menggunakan mesin penyimpanan yang tidak memiliki dukungan partisi native.
- Tidak boleh ada tabel di database `mysql` sistem My SQL 5.7 yang memiliki nama yang sama dengan tabel yang digunakan oleh kamus data My SQL 8.0.
- Tidak boleh ada tabel yang menggunakan jenis atau fungsi data yang usang.
- Tidak boleh ada nama pembatasan kunci asing yang lebih panjang dari 64 karakter.

- Tidak boleh ada SQL mode usang yang ditentukan dalam pengaturan variabel `sql_mode` sistem Anda.
- Tidak boleh ada tabel atau prosedur tersimpan dengan elemen kolom `ENUM` atau `SET` individual dengan panjang melebihi 255 karakter.
- Tidak boleh ada partisi tabel yang berada dalam ruang tabel InnoDB bersama.
- Tidak boleh ada referensi sirkular di jalur file data ruang tabel.
- Tidak boleh ada kueri dan definisi program tersimpan yang menggunakan pengualifikasi `ASC` atau `DESC` untuk klausa `GROUP BY`.
- Tidak boleh ada variabel sistem yang dihapus, dan variabel sistem harus menggunakan nilai default baru untuk My SQL 8.0.
- Tidak boleh ada nilai nol (`0`) untuk date, datetime, atau timestamp.
- Tidak boleh ada inkonsistensi skema yang dihasilkan dari penghapusan atau kerusakan file.
- Tidak boleh ada nama tabel yang berisi string karakter `FTS`.
- Tidak boleh ada tabel InnoDB yang dimiliki oleh mesin yang berbeda.
- Tidak boleh ada nama tabel atau skema yang tidak valid untuk My 5.7. SQL

Untuk detail prechecks yang dijalankan, lihatReferensi deskripsi precheck untuk Aurora MySQL.

Untuk informasi selengkapnya tentang memutakhirkan ke My SQL 8.0, lihat Memutakhirkan Milik Saya SQL di dokumentasi Saya. SQL Untuk gambaran umum tentang perubahan di SQL 8.0 Saya, lihat Apa yang baru di SQL 8.0 Saya di dokumentasi SayaSQL.

Ringkasan dari Aurora My upgrade prechecks SQL

Aurora My SQL memiliki persyaratan spesifiknya sendiri saat meningkatkan dari versi 2 ke versi 3, termasuk yang berikut ini:

- Tidak boleh ada SQL sintaks usang, seperti,, dan, dalam tampilan `SQL_CACHESQL_NO_CACHE`, rutinitas`QUERY_CACHE`, pemicu, dan peristiwa.
- Tidak boleh ada kolom `FTS_DOC_ID` yang ada di tabel apa pun tanpa indeks `FTS`.
- Tidak boleh ada ketidakcocokan definisi kolom antara kamus data InnoDB dan definisi tabel yang sebenarnya.
- Semua nama basis data dan tabel harus huruf kecil ketika parameter `lower_case_table_names` diatur ke 1.

- Peristiwa dan pemicu tidak boleh memiliki pendefinisi yang hilang atau kosong atau konteks pembuatan yang tidak valid.
- Semua nama pemicu dalam basis data harus unik.
- DDLpemulihan dan Cepat DDL tidak didukung di Aurora My SQL versi 3. Tidak boleh ada artefak dalam basis data yang terkait dengan fitur-fitur ini.
- Tabel dengan format baris `COMPACT` atau `REDUNDANT` tidak dapat memiliki indeks yang lebih besar dari 767 byte.
- Panjang awalan indeks yang ditentukan pada kolom teks `tiny` tidak boleh melebihi 255 byte. Dengan set karakter `utf8mb4`, panjang awalan yang didukung dibatasi hingga 63 karakter.

  Panjang awalan yang lebih besar diizinkan di My SQL 5.7 menggunakan parameter. `innodb_large_prefix` Parameter ini tidak digunakan lagi di My 8.0. SQL
- Tidak boleh ada inkonsistensi metadata InnoDB dalam tabel `mysql.host`.
- Tidak boleh ada inkompatibilitas jenis data kolom dalam tabel sistem.
- Tidak boleh ada transaksi XA dalam status `prepared`.
- Nama kolom dalam tampilan tidak boleh lebih panjang dari 64 karakter.
- Karakter khusus dalam prosedur tersimpan tidak boleh tidak konsisten.
- Tabel tidak boleh memiliki inkonsistensi jalur file data.

Untuk detail prechecks yang dijalankan, lihatReferensi deskripsi precheck untuk Aurora MySQL.

Referensi deskripsi precheck untuk Aurora MySQL

Pra-cek pemutakhiran untuk Aurora MySQL dijelaskan di sini secara rinci.

Daftar Isi


- Kesalahan
  - MySQL memeriksa sebelumnya yang melaporkan kesalahan
  - Aurora MySQL memeriksa sebelumnya yang melaporkan kesalahan
- Peringatan
  - MySQL memeriksa sebelumnya yang melaporkan peringatan
  - Aurora MySQL mengecek peringatan laporan itu
- Pemberitahuan

- Kesalahan, peringatan, atau pemberitahuan

Kesalahan

Prececks berikut menghasilkan kesalahan saat precheck gagal, dan pemutakhiran tidak dapat dilanjutkan.

Topik

- MySQL memeriksa sebelumnya yang melaporkan kesalahan
- Aurora MySQL memeriksa sebelumnya yang melaporkan kesalahan

MySQL memeriksa sebelumnya yang melaporkan kesalahan

Precheck berikut berasal dari MySQL Komunitas:

- checkTableMysqlSkema
- circularDirectoryCheck
- columnsWhichCannotHaveDefaultsCheck
- depreciatedSyntaxCheck
- engineMixupCheck
- enumSetElementLengthCheck
- foreignKeyLengthMemeriksa
- getDuplicateTriggers
- getEventsWithNullDefiner
- getMismatchedMetadata
- getTriggersWithNullDefiner
- getValueOfVariableLower_case_table_names
- groupByAscSyntaxCheck
- mysqlEmptyDotTableSyntaxCheck
- mysqlIndexTooLargeCheck
- MySQLinvalid57 NamesCheck
- mysqlOrphanedRoutinesMemeriksa

- mysqlSchemaCheck
- nonNativePartitioningMemeriksa
- oldTemporalCheck
- partitionedTablesInBerbagi TablespaceCheck
- removedFunctionsCheck
- routineSyntaxCheck
- schemaInconsistencyCheck

checkTableMysqlSkema

Tingkat precheck: Kesalahan

Masalah yang dilaporkan oleh **`check table x for upgrade`** perintah untuk **`mysql`** skema

Sebelum memulai upgrade ke Aurora MySQL versi 3, `check table for upgrade` dijalankan pada setiap tabel dalam skema pada instance DB. `mysql check table for upgrade`Perintah memeriksa tabel untuk setiap potensi masalah yang mungkin timbul selama upgrade ke versi MySQL yang lebih baru. Menjalankan perintah ini sebelum mencoba upgrade dapat membantu mengidentifikasi dan menyelesaikan ketidakcocokan sebelumnya, membuat proses upgrade yang sebenarnya lebih lancar.

Perintah ini melakukan berbagai pemeriksaan pada setiap tabel, seperti berikut ini:

- Memverifikasi bahwa struktur tabel dan metadata kompatibel dengan versi MySQL target
- Memeriksa fitur yang tidak digunakan lagi atau dihapus yang digunakan oleh tabel
- Memastikan bahwa tabel dapat ditingkatkan dengan benar tanpa kehilangan data

Untuk informasi selengkapnya, lihat pernyataan CHECK TABLE dalam dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "checkTableMysqlSchema",
  "title": "Issues reported by 'check table x for upgrade' command for mysql
 schema.",
  "status": "OK",
  "detectedProblems": []
}
```

Output untuk precheck ini tergantung pada kesalahan yang ditemui, dan ketika ditemui, karena `check table for upgrade` melakukan beberapa pemeriksaan.

Jika Anda menemukan kesalahan dengan precheck ini, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.

circularDirectoryCheck

Tingkat precheck: Kesalahan

Referensi direktori melingkar di jalur file data tablespace

Pada MySQL 8.0.17, klausa tidak lagi mengizinkan referensi direktori `CREATE TABLESPACE ... ADD DATAFILE` melingkar. Untuk menghindari masalah peningkatan, hapus referensi direktori melingkar dari jalur file data tablespace sebelum memutakhirkan ke Aurora MySQL versi 3.

Contoh keluaran:

```
{
  "id": "circularDirectory",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "description": "Error: Following tablespaces contain circular directory references
 (e.g. the reference '/../') in data file paths which as of MySQL 8.0.17 are not
 permitted by the CREATE TABLESPACE ... ADD DATAFILE clause. An exception to the
 restriction exists on Linux, where a circular directory reference is permitted if
 the preceding directory is a symbolic link. To avoid upgrade issues, remove any
 circular directory references from tablespace data file paths before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-innodb-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "ts2",
        "description": "circular reference in datafile path: '/home/ec2-user/dbdata/
mysql_5_7_44/../ts2.ibd'",
        "dbObjectType": "Tablespace"
      }
  ]
}
```

Jika Anda menerima kesalahan ini, buat kembali tabel Anda menggunakan file-per-table tablespace. Gunakan jalur file default untuk semua definisi tablespace dan tabel.

> **Note**
>
> Aurora MySQL tidak mendukung ruang meja atau perintah umum. `CREATE TABLESPACE` Sebelum membangun kembali tablespace, lihat Operasi DDL online di dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

Setelah membangun kembali, precheck berlalu, memungkinkan peningkatan untuk melanjutkan.

```
{
  "id": "circularDirectoryCheck",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "detectedProblems": []
},
```

columnsWhichCannotHaveDefaultsCheck

Tingkat precheck: Kesalahan

Kolom yang tidak dapat memiliki nilai default

Sebelum MySQL 8.0.13`BLOB`,,, `TEXT`GEOMETRY, `JSON` dan kolom tidak dapat memiliki nilai default. Hapus klausa default pada kolom ini sebelum memutakhirkan ke Aurora MySQL versi 3. Untuk informasi selengkapnya tentang perubahan penanganan default untuk tipe data ini, lihat nilai default tipe Data dalam dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "description": "Error: The following columns are defined as either BLOB, TEXT,
 GEOMETRY or JSON and have a default value set. These data types cannot have default
 values in MySQL versions prior to 8.0.13, while starting with 8.0.13, the default
```

```
 value must be specified as an expression. In order to fix this issue, please use
 the ALTER TABLE ... ALTER COLUMN ... DROP DEFAULT statement.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/data-type-
defaults.html#data-type-defaults-explicit",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test_blob_default.geo_col",
        "description": "geometry"
      }
  ]
},
```

Precheck mengembalikan kesalahan karena `geo_col` kolom dalam `test.test_blob_default` tabel menggunakan`BLOB`,, `TEXT``GEOMETRY`, atau tipe `JSON` data dengan nilai default yang ditentukan.

Melihat definisi tabel, kita dapat melihat bahwa `geo_col` kolom didefinisikan sebagai`geo_col geometry NOT NULL default ''`.

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL DEFAULT ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Menghapus klausa default ini untuk memungkinkan precheck lulus.

> **Note**
>
> Sebelum menjalankan `ALTER TABLE` pernyataan atau membangun kembali ruang tabel, lihat Operasi DDL online di dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

```
mysql> ALTER TABLE test_blob_default modify COLUMN geo_col geometry NOT NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Precheck berlalu, dan Anda dapat mencoba kembali upgrade.

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "detectedProblems": []
},
```

depreciatedSyntaxCheck

Tingkat precheck: Kesalahan

Penggunaan kata kunci yang disusutkan dalam definisi

MySQL 8.0 telah menghapus cache kueri. Akibatnya, beberapa sintaks SQL khusus cache kueri telah dihapus. Jika salah satu objek database Anda berisi`QUERY CACHE`,`SQL_CACHE`, atau `SQL_NO_CACHE` kata kunci, kesalahan precheck dikembalikan. Untuk mengatasi masalah ini, buat ulang objek ini, hapus kata kunci yang disebutkan.

Contoh keluaran:

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "description": "Error: The following DB objects contain keywords like 'QUERY
 CACHE', 'SQL_CACHE', 'SQL_NO_CACHE' which are not supported in major version 8.0.
 It is recommended to drop these DB objects or rebuild without any of the above
 keywords before upgrade.",
  "detectedProblems": [
      {
"level": "Error",
"dbObject": "test.no_query_cache_check",
```

```
"description": "PROCEDURE uses depreciated words in definition"
      }
  ]
}
```

Precheck melaporkan bahwa prosedur yang `test.no_query_cache_check` disimpan menggunakan salah satu kata kunci yang dihapus. Melihat definisi prosedur, kita dapat melihat bahwa ia menggunakan`SQL_NO_CACHE`.

```
mysql> show create procedure test.no_query_cache_check\G
*************************** 1. row ***************************
           Procedure: no_query_cache_check
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `no_query_cache_check`()
BEGIN
    SELECT SQL_NO_CACHE k from sbtest1 where id > 10 and id < 20 group by k asc;
END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Hapus kata kunci.

```
mysql> drop procedure test.no_query_cache_check;
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter //

mysql> CREATE DEFINER=`reinvent`@`%` PROCEDURE `no_query_cache_check`() BEGIN
 SELECT k from sbtest1 where id > 10 and id < 20 group by k asc; END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Setelah menghapus kata kunci, precheck selesai dengan sukses.

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
```

```
  "status": "OK",
  "detectedProblems": []
}
```

engineMixupCheck

Tingkat precheck: Kesalahan

Tabel yang diakui oleh InnoDB milik mesin yang berbeda

Mirip dengan schemaInconsistencyCheck, precheck ini memverifikasi bahwa metadata tabel di MySQL konsisten sebelum melanjutkan dengan peningkatan.

Jika Anda menemukan kesalahan dengan precheck ini, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.

Contoh keluaran:

```
{
  "id": "engineMixupCheck",
  "title": "Tables recognized by InnoDB that belong to a different engine",
  "status": "OK",
  "description": "Error: Following tables are recognized by InnoDB engine while the
 SQL layer believes they belong to a different engine. Such situation may happen
 when one removes InnoDB table files manually from the disk and creates e.g. a
 MyISAM table with the same name.\n\nA possible way to solve this situation is to
 e.g. in case of MyISAM table:\n\n1. Rename the MyISAM table to a temporary name
 (RENAME TABLE).\n2. Create some dummy InnoDB table (its definition does not need to
 match), then copy (copy, not move) and rename the dummy .frm and .ibd files to the
 orphan name using OS file commands.\n3. The orphan table can be then dropped (DROP
 TABLE), as well as the dummy table.\n4. Finally the MyISAM table can be renamed
 back to its original name.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.general_log_backup",
        "description": "recognized by the InnoDB engine but belongs to CSV"
      }
  ]
}
```

enumSetElementLengthCheck

Tingkat precheck: Kesalahan

**ENUM**dan definisi **SET** kolom yang mengandung elemen yang lebih panjang dari 255 karakter

Tabel dan prosedur yang disimpan tidak boleh memiliki ENUM atau elemen SET kolom melebihi 255 karakter atau 1020 byte. Sebelum MySQL 8.0, panjang gabungan maksimum adalah 64K, tetapi 8.0 membatasi elemen individu untuk 255 karakter atau 1020 byte (mendukung multibyte). Jika Anda mendapatkan kegagalan precheck`enumSetElementLengthCheck`, ubah elemen apa pun yang melebihi batas baru ini sebelum mencoba kembali pemutakhiran.

Contoh keluaran:

```
{
  "id": "enumSetElementLengthCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "description": "Error: The following columns are defined as either ENUM or SET and
 contain at least one element longer that 255 characters. They need to be altered so
 that all elements fit into the 255 characters limit.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/string-type-
overview.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.large_set.s",
        "description": "SET contains element longer than 255 characters"
      }
  ]
},
```

Precheck melaporkan kesalahan karena kolom `s` dalam `test.large_set` tabel berisi SET elemen yang lebih besar dari 255 karakter.

Setelah mengurangi SET ukuran untuk kolom ini, precheck berlalu, memungkinkan peningkatan untuk melanjutkan.

```
{
  "id": "enumSetElementLenghtCheck",
```

```
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "detectedProblems": []
},
```

foreignKeyLengthMemeriksa

Tingkat precheck: Kesalahan

Nama kendala kunci asing lebih panjang dari 64 karakter

Di MySQL, panjang pengidentifikasi dibatasi hingga 64 karakter, sebagaimana diuraikan dalam dokumentasi MySQL. Karena masalah yang diidentifikasi di mana panjang kunci asing dapat sama atau melebihi nilai ini, yang menyebabkan kegagalan peningkatan, precheck ini diterapkan. Jika Anda mengalami kesalahan dengan precheck ini, Anda harus mengubah atau mengganti nama kendala Anda sehingga kurang dari 64 karakter sebelum mencoba kembali pemutakhiran.

Contoh keluaran:

```
{
  "id": "foreignKeyLength",
  "title": "Foreign key constraint names longer than 64 characters",
  "status": "OK",
  "detectedProblems": []
}
```

getDuplicateTriggers

Tingkat precheck: Kesalahan

Semua nama pemicu dalam database harus unik.

Karena perubahan dalam implementasi kamus data, MySQL 8.0 tidak mendukung pemicu case-sensitive dalam database. Precheck ini memvalidasi bahwa cluster DB Anda tidak memiliki satu atau lebih database yang berisi pemicu duplikat. Untuk informasi selengkapnya, lihat Sensitivitas huruf pengenal dalam dokumentasi MySQL.

Contoh keluaran:

```
{
```

```
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "description": "Error: You have one or more database containing duplicate
 triggers. Mysql 8.0 does not support case sensitive triggers within a database
 https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html. To
 upgrade to MySQL 8.0, drop the triggers with case-insensitive duplicate names and
 recreate with distinct names.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_product"
      },
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_PRODUCT"
      }
  ]
}
```

Precheck melaporkan kesalahan bahwa cluster database memiliki dua pemicu dengan nama yang sama, tetapi menggunakan kasus yang berbeda: `test.before_insert_product` dan. `test.before_insert_PRODUCT`

Sebelum memutakhirkan, ganti nama pemicu atau jatuhkan dan buat ulang dengan nama baru.

Setelah mengganti nama `test.before_insert_PRODUCT` menjadi`test.before_insert_product_2`, precheck berhasil.

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "detectedProblems": []
}
```

getEventsWithNullDefiner

Tingkat precheck: Kesalahan

Kolom definer untuk tidak **`mysql.event`** bisa null atau kosong.

`DEFINER`Atribut menentukan akun MySQL yang memiliki definisi objek tersimpan, seperti pemicu, prosedur tersimpan, atau peristiwa. Atribut ini sangat berguna dalam situasi di mana Anda ingin mengontrol konteks keamanan di mana objek yang disimpan berjalan. Saat membuat objek tersimpan, jika `DEFINER` tidak ditentukan, defaultnya adalah pengguna yang membuat objek tersebut.

Saat memutakhirkan ke MySQL 8.0, Anda tidak dapat memiliki objek tersimpan yang memiliki definer atau kosong dalam kamus `null` data MySQL. Jika Anda memiliki objek yang disimpan seperti itu, kesalahan precheck akan muncul. Anda harus memperbaikinya sebelum upgrade dapat dilanjutkan.

Contoh kesalahan:

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "description": "Error: Set definer column in mysql.event to a valid non-null
 definer.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.get_version",
        "description": "Set definer for event get_version in Schema test"
      }
  ]
}
```

Precheck mengembalikan kesalahan untuk `test.get_version` acara karena memiliki `null` definer.

Untuk mengatasi ini, Anda dapat memeriksa definisi acara. Seperti yang Anda lihat, definernya kosong `null` atau kosong.

```
mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+---------+
| db   | name        | definer |
+------+-------------+---------+
| test | get_version |         |
+------+-------------+---------+
```

```
1 row in set (0.00 sec)
```

Jatuhkan atau buat ulang acara dengan definer yang valid.

> **Note**
>
> Sebelum menjatuhkan atau mendefinisikan ulang`DEFINER`, tinjau dan periksa aplikasi dan persyaratan hak istimewa Anda dengan cermat. Untuk informasi selengkapnya, lihat Kontrol akses objek tersimpan dalam dokumentasi MySQL.

```
mysql> drop event test.get_version;
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> delimiter $$
mysql> CREATE EVENT get_version
    ->     ON SCHEDULE
    ->       EVERY 1 DAY
    ->     DO
    ->      ///DO SOMETHING //
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+------------+
| db   | name        | definer    |
+------+-------------+------------+
| test | get_version | reinvent@% |
+------+-------------+------------+
1 row in set (0.00 sec)
```

Sekarang precheck berlalu.

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "detectedProblems": []},
```

getMismatchedMetadata

Tingkat precheck: Kesalahan

Ketidakcocokan definisi kolom antara kamus data InnoDB dan definisi tabel aktual

Mirip dengan schemaInconsistencyCheck, precheck ini memverifikasi bahwa metadata tabel di MySQL konsisten sebelum melanjutkan dengan peningkatan. Dalam hal ini, precheck memverifikasi bahwa definisi kolom cocok antara kamus data InnoDB dan definisi tabel MySQL. Jika ketidakcocokan jika terdeteksi, pemutakhiran tidak dilanjutkan.

Jika Anda menemukan kesalahan dengan precheck ini, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.

Contoh keluaran:

```
{
  "id": "getMismatchedMetadata",
  "title": "Column definition mismatch between InnoDB Data Dictionary and actual
 table definition.",
  "status": "OK",
  "description": "Error: Your database has mismatched metadata. The upgrade to mysql
 8.0 will not succeed until this is fixed.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.mismatchTable",
        "description": "Table `test/mismatchTable` column names mismatch with InnoDb
 dictionary column names: iD <> id"
      }
  ]
}
```

Precheck melaporkan ketidakcocokan dalam metadata untuk `id` kolom dalam tabel. `test.mismatchTable` Secara khusus, metadata MySQL memiliki nama kolom sebagai`iD`, sedangkan InnoDB memilikinya sebagai. `id`

getTriggersWithNullDefiner

Tingkat precheck: Kesalahan

Kolom definer untuk tidak **`information_schema.triggers`** bisa **`null`** atau kosong.

Precheck memvalidasi bahwa database Anda tidak memiliki pemicu yang didefinisikan dengan `null` atau definer kosong. Untuk informasi selengkapnya tentang persyaratan definer untuk objek yang disimpan, lihat getEventsWithNullDefiner.

Contoh keluaran:

```
{
  "id": "getTriggersWithNullDefiner",
  "title": "The definer column for information_schema.triggers cannot be null or
 blank.",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.example_trigger",
        "description": "Set definer for trigger example_trigger in Schema test"
      }
  ]
}
```

Precheck mengembalikan kesalahan karena `example_trigger` pemicu dalam `test` skema memiliki definer. `null` Untuk memperbaiki masalah ini, perbaiki definer dengan membuat ulang pemicu dengan pengguna yang valid, atau lepaskan pelatuknya. Untuk informasi selengkapnya, lihat contoh di getEventsWithNullDefiner.

> **Note**
>
> Sebelum menjatuhkan atau mendefinisikan ulang`DEFINER`, tinjau dan periksa aplikasi dan persyaratan hak istimewa Anda dengan cermat. Untuk informasi selengkapnya, lihat Kontrol akses objek tersimpan dalam dokumentasi MySQL.

getValueOfVariableLower_case_table_names

Tingkat precheck: Kesalahan

Semua nama database atau tabel harus huruf kecil ketika **`lower_case_table_names`** parameter diatur ke. **`1`**

Sebelum MySQL 8.0, nama database, nama tabel, dan objek lainnya berhubungan dengan file dalam direktori data, seperti metadata berbasis file (.frm). Variabel sistem lower_case_table_names memungkinkan pengguna untuk mengontrol bagaimana server menangani sensitivitas kasus identifier untuk objek database, dan penyimpanan objek metadata tersebut. Parameter ini dapat diubah pada server yang sudah diinisialisasi setelah reboot.

Namun, di MySQL 8.0, sementara parameter ini masih mengontrol bagaimana server menangani sensitivitas kasus pengenal, itu tidak dapat diubah setelah kamus data diinisialisasi. Saat memutakhirkan atau membuat database MySQL 8.0, nilai yang ditetapkan `lower_case_table_names` untuk pertama kalinya kamus data dimulai di MySQL, digunakan untuk seumur hidup database itu. Pembatasan ini diberlakukan sebagai bagian dari implementasi implementasi Atomic Data Dictionary, di mana objek database dimigrasikan dari metadata berbasis file ke tabel InnoDB internal dalam skema. `mysql`

Untuk informasi selengkapnya, lihat Perubahan kamus data dalam dokumentasi MySQL.

Untuk menghindari masalah selama pemutakhiran saat memperbarui metadata berbasis file ke Kamus Data Atom baru, pemeriksaan awal ini memvalidasi bahwa ketika`lower_case_table_names = 1`, semua tabel disimpan di disk dalam huruf kecil. Jika tidak, kesalahan precheck dikembalikan, dan Anda harus memperbaiki metadata sebelum melanjutkan dengan peningkatan.

Contoh keluaran:

```
{
  "id": "getValueOfVariablelower_case_table_names",
  "title": "MySQL pre-checks that all database or table names are lowercase when the
 lower_case_table_names parameter is set to 1.",
  "status": "OK",
  "description": "Error: You have one or more databases or tables with uppercase
 letters in the names, but the lower_case_table_names parameter is set to 1. To
 upgrade to MySQL 8.0, either change all database or table names to lowercase, or
 set the parameter to 0.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.TEST",
        "description": "Table test.TEST contains one or more capital letters in name
 while lower_case_table_names = 1"
      }
  ]
```

```
}
```

Kesalahan dikembalikan karena tabel `test.TEST` berisi huruf besar, tetapi `lower_case_table_names` diatur ke. `1`

Untuk mengatasi masalah ini, Anda dapat mengganti nama tabel untuk menggunakan huruf kecil, atau memodifikasi `lower_case_table_names` parameter pada cluster DB sebelum memulai pemutakhiran.

> **Note**
>
> Uji dan tinjau dokumentasi sensitivitas huruf besar/kecil di MySQL dengan cermat, dan bagaimana perubahan tersebut dapat memengaruhi aplikasi Anda.
> Juga tinjau dokumentasi MySQL 8.0 tentang bagaimana lower_case_table_names ditangani secara berbeda di MySQL 8.0.

groupByAscSyntaxCheck

Tingkat precheck: Kesalahan

Penggunaan **`GROUP BY ASC/DESC`** sintaks yang dihapus

Pada MySQL 8.0.13, usang atau sintaks untuk `ASC` klausa telah dihapus`DESC`. `GROUP BY` Kueri yang mengandalkan `GROUP BY` penyortiran sekarang dapat menghasilkan hasil yang berbeda. Untuk mendapatkan urutan pengurutan tertentu, gunakan `ORDER BY` klausa sebagai gantinya. Jika ada objek yang ada dalam database Anda menggunakan sintaks ini, Anda harus membuat ulang mereka menggunakan `ORDER BY` klausa sebelum mencoba kembali upgrade. Untuk informasi selengkapnya, lihat perubahan SQL dalam dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "groupbyAscSyntaxCheck",
  "title": "Usage of removed GROUP BY ASC/DESC syntax",
  "status": "OK",
  "description": "Error: The following DB objects use removed GROUP BY ASC/DESC
 syntax. They need to be altered so that ASC/DESC keyword is removed from GROUP BY
 clause and placed in appropriate ORDER BY clause.",
  "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/
news-8-0-13.html#mysqld-8-0-13-sql-syntax",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.groupbyasc",
        "description": "PROCEDURE uses removed GROUP BY ASC syntax",
        "dbObjectType": "Routine"
      }
  ]
}
```

mysqlEmptyDotTableSyntaxCheck

Tingkat precheck: Kesalahan

Periksa **`.<table>`** sintaks usang yang digunakan dalam rutinitas.

Di MySQL 8.0, rutinitas tidak dapat lagi berisi sintaks pengenal usang (). `\".<table>\"` Jika ada rutinitas atau pemicu yang disimpan berisi pengidentifikasi tersebut, pemutakhiran gagal. Misalnya, `.dot_table` referensi berikut tidak lagi diizinkan:

```
mysql> show create procedure incorrect_procedure\G
*************************** 1. row ***************************
           Procedure: incorrect_procedure
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `incorrect_procedure`()
BEGIN delete FROM .dot_table; select * from .dot_table where 1=1; END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Setelah Anda membuat ulang rutinitas dan pemicu untuk menggunakan sintaks pengenal yang benar dan melarikan diri, precheck lolos, dan pemutakhiran dapat dilanjutkan. Untuk informasi selengkapnya tentang pengidentifikasi, lihat nama objek skema dalam dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "mysqlEmptyDotTableSyntaxCheck",
  "title": "Check for deprecated '.<table>' syntax used in routines.",
  "status": "OK",
```

```
  "description": "Error: The following routines contain identifiers in deprecated
 identifier syntax (\".<table>\"), and should be corrected before upgrade:\n",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.incorrect_procedure",
        "description": " routine body contains deprecated identifiers."
      }
  ]
}
```

Precheck mengembalikan kesalahan untuk `incorrect_procedure` rutin dalam `test` database karena berisi sintaks usang.

Setelah Anda memperbaiki rutinitas, precheck berhasil, dan Anda dapat mencoba lagi upgrade.

mysqlIndexTooLargeCheck

Tingkat precheck: Kesalahan

Periksa indeks yang terlalu besar untuk bekerja pada versi MySQL yang lebih tinggi dari 5.7

Untuk format baris yang ringkas atau berlebihan, seharusnya tidak mungkin membuat indeks dengan awalan yang lebih besar dari 767 byte. Namun, sebelum MySQL versi 5.7.35 ini dimungkinkan. Untuk informasi selengkapnya, lihat catatan rilis MySQL 5.7.35.

Setiap indeks yang terpengaruh oleh bug ini akan menjadi tidak dapat diakses setelah memutakhirkan ke MySQL 8.0. Precheck ini mengidentifikasi indeks yang menyinggung yang harus dibangun kembali sebelum pemutakhiran diizinkan untuk dilanjutkan.

```
{
 "id": "mysqlIndexTooLargeCheck",
 "title": "Check for indexes that are too large to work on higher versions of MySQL
Server than 5.7",
 "status": "OK",
 "description": "Error: The following indexes ware made too large for their format
in an older version of MySQL (older than 5.7.34). Normally those indexes within
tables with compact or redundant row formats shouldn't be larger than 767 bytes. To
fix this problem those indexes should be dropped before upgrading or those tables
will be inaccessible.",
 "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_large_idx",
```

```
        "description": "IDX_2"
      }
  ]
}
```

Precheck mengembalikan kesalahan karena `test.table_with_large_idx` tabel berisi indeks pada tabel menggunakan format baris kompak atau redundan yang lebih besar dari 767 byte. Tabel ini akan menjadi tidak dapat diakses setelah memutakhirkan ke MySQL 8.0. Sebelum melanjutkan dengan upgrade, lakukan salah satu hal berikut:

- Jatuhkan indeks yang disebutkan dalam precheck.
- Tambahkan indeks yang disebutkan di precheck.
- Ubah format baris yang digunakan oleh tabel.

Di sini kita membangun kembali tabel untuk menyelesaikan kegagalan precheck. Sebelum membangun kembali tabel, pastikan bahwa innodb_file_format diatur ke, dan innodb_default_row_format diatur ke`Barracuda.` `dynamic` Ini adalah default di MySQL 5.7. Untuk informasi selengkapnya, lihat format baris InnoDB dan manajemen format file InnoDB dalam dokumentasi MySQL.

> **Note**
>
> Sebelum membangun kembali tablespace, lihat Operasi DDL online di dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

```
mysql > select @@innodb_file_format,@@innodb_default_row_format;
+----------------------+-----------------------------+
| @@innodb_file_format | @@innodb_default_row_format |
+----------------------+-----------------------------+
| Barracuda            | dynamic                     |
+----------------------+-----------------------------+
1 row in set (0.00 sec)

mysql> optimize table table_with_large_idx;
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| Table                     | Op       | Msg_type | Msg_text
                                |
```

```
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| test.table_with_large_idx | optimize | note     | Table does not support optimize,
 doing recreate + analyze instead |
| test.table_with_large_idx | optimize | status   | OK
                                  |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.02 sec)

# Verify FILE_FORMAT and ROW_FORMAT
mysql>  select * from information_schema.innodb_sys_tables where name like 'test/
table_with_large_idx';
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
| TABLE_ID | NAME                      | FLAG | N_COLS | SPACE | FILE_FORMAT |
 ROW_FORMAT | ZIP_PAGE_SIZE | SPACE_TYPE |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
|       43 | test/table_with_large_idx |   33 |      4 |    26 | Barracuda   |
 Dynamic    |             0 | Single     |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
1 row in set (0.00 sec)
```

Setelah membangun kembali tabel, precheck berlalu, dan peningkatan dapat dilanjutkan.

```
{
  "id": "mysqlIndexTooLargeCheck",
  "title": "Check for indexes that are too large to work on higher versions of MySQL
 Server than 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

MySQLinkValid57 NamesCheck

Tingkat precheck: Kesalahan

Periksa nama tabel dan skema yang tidak valid yang digunakan di MySQL 5.7

Saat bermigrasi ke kamus data baru di MySQL 8.0, instance database Anda tidak dapat berisi skema atau tabel yang diawali dengan. `#mysql50#` Jika ada objek seperti itu, pemutakhiran

gagal. Untuk mengatasi masalah ini, jalankan mysqlcheck terhadap skema dan tabel yang dikembalikan.

> **Note**
>
> Pastikan Anda menggunakan versi MySQL 5.7, karena - fix-db-names - dan - fix-table-names - telah dihapus `mysqlcheck` dari MySQL 8.0.

Contoh keluaran:

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "description": "The following tables and/or schemas have invalid names. In order
 to fix them use the mysqlcheck utility as follows:\n\n  $ mysqlcheck --check-
upgrade --all-databases\n  $ mysqlcheck --fix-db-names --fix-table-names --all-
databases\n\nOR via mysql client, for eg:\n\n  ALTER DATABASE `#mysql50#lost+found`
 UPGRADE DATA DIRECTORY NAME;",
  "documentationLink": "https://dev.mysql.com/doc/refman/5.7/en/identifier-
mapping.html https://dev.mysql.com/doc/refman/5.7/en/alter-database.html https://
dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "#mysql50#fix_db_names",
        "description": "Schema name"
      }
  ]
}
```

Precheck melaporkan bahwa skema `#mysql50#fix_db_names` memiliki nama yang tidak valid.

Setelah memperbaiki nama skema, precheck lolos, memungkinkan peningkatan untuk melanjutkan.

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "detectedProblems": []
```

```
},
```

mysqlOrphanedRoutinesMemeriksa

Tingkat precheck: Kesalahan

Periksa rutinitas yatim piatu di 5.7

Saat bermigrasi ke kamus data baru di MySQL 8.0, jika ada prosedur tersimpan dalam database di mana skema tidak ada lagi, pemutakhiran gagal. Precheck ini memverifikasi bahwa semua skema yang direferensikan dalam prosedur tersimpan pada instans DB Anda masih ada. Untuk memungkinkan pemutakhiran dilanjutkan, jatuhkan prosedur yang disimpan ini.

Contoh keluaran:

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "description": "Error: The following routines have been orphaned. Schemas that
 they are referencing no longer exists.\nThey have to be cleaned up or the upgrade
 will fail.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "dropped_db.get_version",
        "description": "is orphaned"
      }
  ]
},
```

Precheck melaporkan bahwa prosedur yang `get_version` disimpan dalam `dropped_db` database adalah yatim piatu.

Untuk membersihkan prosedur ini, Anda dapat membuat ulang skema yang hilang.

```
mysql> create database dropped_db;
Query OK, 1 row affected (0.01 sec)
```

Setelah skema dibuat ulang, Anda dapat menghentikan prosedur untuk memungkinkan peningkatan dilanjutkan.

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlSchemaCheck

Tingkat precheck: Kesalahan

Nama tabel dalam **`mysql`** skema yang bertentangan dengan tabel baru di MySQL 8.0

Kamus Data Atom baru yang diperkenalkan di MySQL 8.0 menyimpan semua metadata dalam satu set tabel InnoDB internal dalam skema. `mysql` Selama upgrade, tabel kamus data internal baru dibuat dalam `mysql` skema. Untuk menghindari tabrakan penamaan, yang akan mengakibatkan kegagalan pemutakhiran, precheck memeriksa semua nama tabel dalam `mysql` skema untuk memastikan bahwa tidak ada nama tabel baru yang sudah digunakan. Jika ya, kesalahan dikembalikan, dan pemutakhiran tidak diizinkan untuk dilanjutkan.

Contoh keluaran:

```
{
  "id": "mysqlSchema",
  "title": "Table names in the mysql schema conflicting with new tables in the
 latest MySQL.",
  "status": "OK",
  "description": "Error: The following tables in mysql schema have names that will
 conflict with the ones introduced in the latest version. They must be renamed or
 removed before upgrading (use RENAME TABLE command). This may also entail changes
 to applications that use the affected tables.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrade-before-you-
begin.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.tablespaces",
        "description": "Table name used in mysql schema.",
        "dbObjectType": "Table"
      }
  ]
}
```

Kesalahan dikembalikan karena ada tabel bernama `tablespaces` dalam `mysql` skema. Ini adalah salah satu nama tabel kamus data internal baru di MySQL 8.0. Anda harus mengganti nama atau menghapus tabel tersebut sebelum memutakhirkan, dengan menggunakan perintah. `RENAME TABLE`

nonNativePartitioningMemeriksa

Tingkat precheck: Kesalahan

Tabel yang dipartisi menggunakan mesin dengan partisi non-asli

Menurut dokumentasi MySQL 8.0, dua mesin penyimpanan saat ini menyediakan dukungan partisi asli: InnoDB dan NDB. Dari jumlah tersebut, hanya InnoDB yang didukung di Aurora MySQL versi 3, kompatibel dengan MySQL 8.0. Setiap upaya untuk membuat tabel yang dipartisi di MySQL 8.0 menggunakan mesin penyimpanan lainnya gagal. Precheck ini mencari tabel di cluster DB Anda yang menggunakan partisi non-native. Jika ada yang dikembalikan, Anda harus menghapus partisi atau mengubah mesin penyimpanan ke InnoDB.

Contoh keluaran:

```
{
  "id": "nonNativePartitioning",
  "title": "Partitioned tables using engines with non native partitioning",
  "status": "OK",
  "description": "Error: In the latest MySQL storage engine is responsible for
 providing its own partitioning handler, and the MySQL server no longer provides
 generic partitioning support. InnoDB and NDB are the only storage engines that
 provide a native partitioning handler that is supported in the latest MySQL. A
 partitioned table using any other storage engine must be altered—either to convert
 it to InnoDB or NDB, or to remove its partitioning—before upgrading the server,
 else it cannot be used afterwards.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-configuration-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partMyisamTable",
         "description": "MyISAM engine does not support native partitioning",
         "dbObjectType": "Table"
      }
  ]
}
```

Di sini tabel myISAM menggunakan partisi, yang memerlukan tindakan sebelum pemutakhiran dapat dilanjutkan.

oldTemporalCheck

Tingkat precheck: Kesalahan

Penggunaan tipe temporal lama

“Temporal lama” mengacu pada kolom tipe temporal (seperti `TIMESTAMP` dan`DATETIME`) yang dibuat di MySQL versi 5.5 dan lebih rendah. Di MySQL 8.0, dukungan untuk tipe data temporal lama ini dihapus, yang berarti bahwa peningkatan di tempat dari MySQL 5.7 ke 8.0 tidak dimungkinkan jika database berisi tipe temporal lama ini. Untuk memperbaikinya, Anda harus membangun kembali tabel apa pun yang berisi jenis tanggal temporal lama seperti itu, sebelum melanjutkan dengan peningkatan.

Untuk informasi lebih lanjut tentang penghentian tipe data temporal lama di MySQL 5.7, lihat blog ini. Untuk informasi lebih lanjut tentang penghapusan tipe data temporal lama di MySQL 8.0, lihat blog ini.

> **Note**
>
> Sebelum membangun kembali tablespace, lihat Operasi DDL online di dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

Contoh keluaran:

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "description": "Error: Following table columns use a deprecated and no longer
 supported temporal disk storage format. They must be converted to the new format
 before upgrading. It can by done by rebuilding the table using 'ALTER TABLE
 <table_name> FORCE' command",
  "documentationLink": "https://dev.mysql.com/blog-archive/mysql-8-0-removing-
support-for-old-temporal-datatypes/",
  "detectedProblems": [
      {
```

```
        "level": "Error",
        "dbObject": "test.55_temporal_table.timestamp_column",
        "description": "timestamp /* 5.5 binary format */",
        "dbObjectType": "Column"
      }
  ]
},
```

Kesalahan dilaporkan untuk kolom `timestamp_column` dalam tabel`test.55_temporal_table`, karena menggunakan format penyimpanan disk temporal lama yang tidak lagi didukung.

Untuk mengatasi masalah ini dan memungkinkan peningkatan untuk melanjutkan, buat kembali tabel untuk mengonversi format penyimpanan disk temporal lama ke yang baru diperkenalkan di MySQL 5.6. Untuk informasi selengkapnya dan prasyarat sebelum melakukannya, lihat Mengonversi antara set karakter Unicode 3-byte dan 4-byte dalam dokumentasi MySQL.

Menjalankan perintah berikut untuk membangun kembali tabel yang disebutkan dalam precheck ini mengubah tipe temporal lama ke format yang lebih baru dengan presisi fraksional-detik.

```
ALTER TABLE ... ENGINE=InnoDB;
```

Untuk informasi selengkapnya tentang membangun kembali tabel, lihat pernyataan ALTER TABLE dalam dokumentasi MySQL.

Setelah membangun kembali tabel yang dimaksud dan memulai ulang pemutakhiran, pemeriksaan kompatibilitas berlalu, memungkinkan peningkatan untuk melanjutkan.

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "detectedProblems": []
}
```

partitionedTablesInBerbagi TablespaceCheck

Tingkat precheck: Kesalahan

Penggunaan tabel yang dipartisi di ruang tabel bersama

Pada MySQL 8.0.13, dukungan untuk menempatkan partisi tabel di ruang tabel bersama dihapus. Sebelum memutakhirkan, pindahkan tabel tersebut dari ruang tabel bersama ke file-per-table ruang tabel.

> **Note**
>
> Sebelum membangun kembali ruang tabel, lihat Mempartisi operasi dalam dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

Contoh keluaran:

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "description": "Error: The following tables have partitions in shared tablespaces.
 They need to be moved to file-per-table tablespace before upgrading. You can do
 this by running query like 'ALTER TABLE table_name REORGANIZE PARTITION X INTO
 (PARTITION X VALUES LESS THAN (30) TABLESPACE=innodb_file_per_table);'",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partInnoDBTable",
        "description": "Partition p1 is in shared tablespace innodb",
        "dbObjectType": "Table"
      }
  ]
}
```

Precheck gagal karena partisi `p1` dari tabel `test.partInnoDBTable` ada di tablespace sistem.

Untuk mengatasi masalah ini, buat kembali `test.partInnodbTable` tabel, tempatkan partisi yang menyinggung `p1` di tablespace. file-per-table

```
mysql > ALTER TABLE partInnodbTable REORGANIZE PARTITION p1
    ->   INTO (PARTITION p1 VALUES LESS THAN ('2014-01-01')
 TABLESPACE=innodb_file_per_table);
Query OK, 0 rows affected, 1 warning (0.02 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0
```

Setelah melakukannya, precheck berlalu.

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "detectedProblems": []
}
```

removedFunctionsCheck

Tingkat precheck: Kesalahan

Penggunaan fungsi yang telah dihapus dari versi MySQL terbaru

Di MySQL 8.0, sejumlah fungsi bawaan telah dihapus. Precheck ini memeriksa database Anda untuk objek yang mungkin menggunakan fungsi-fungsi ini. Jika ditemukan, kesalahan akan dikembalikan. Anda harus menyelesaikan masalah sebelum mencoba kembali pemutakhiran.

Mayoritas fungsi yang dihapus adalah fungsi spasial, yang telah diganti dengan ST_* fungsi yang setara. Dalam kasus ini, Anda memodifikasi objek database untuk menggunakan penamaan prosedur baru. Untuk informasi selengkapnya, lihat Features removed in MySQL 8.0 dalam dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "description": "Error: The following DB objects use functions that were removed
 in the latest MySQL version. Please make sure to update them to use supported
 alternatives before upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.GetLocationsInPolygon",
        "description": "PROCEDURE uses removed function POLYGONFROMTEXT (consider
 using ST_POLYGONFROMTEXT instead)",
```

```
        "dbObjectType": "Routine"
      },
      {
        "level": "Error",
        "dbObject": "test.InsertLocation",
        "description": "PROCEDURE uses removed function POINTFROMTEXT (consider
 using ST_POINTFROMTEXT instead)",
        "dbObjectType": "Routine"
      }
  ]
},
```

Precheck melaporkan bahwa prosedur yang `test.GetLocationsInPolygon` disimpan menggunakan dua fungsi yang dihapus: POLYGONFROMTEXT dan POINTFROMTEXT. Ini juga menyarankan agar Anda menggunakan ST_POLYGONFROMTEXT dan ST_POINTFROMTEXT baru sebagai pengganti. Setelah membuat ulang prosedur menggunakan saran, precheck berhasil diselesaikan.

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "detectedProblems": []
},
```

> **Note**
>
> Meskipun dalam kebanyakan kasus, fungsi yang tidak digunakan lagi memiliki penggantian langsung, pastikan Anda menguji aplikasi dan meninjau dokumentasi untuk setiap perubahan perilaku sebagai akibat dari perubahan tersebut.

routineSyntaxCheck

Tingkat precheck: Kesalahan

Pemeriksaan sintaks MySQL untuk objek seperti rutin

MySQL 8.0 memperkenalkan kata kunci cadangan yang tidak dicadangkan sebelumnya. Upgrade prechecker mengevaluasi penggunaan kata kunci yang dicadangkan dalam nama objek database dan dalam definisi dan badannya. Jika mendeteksi kata kunci cadangan yang digunakan dalam

objek database, seperti prosedur, fungsi, peristiwa, dan pemicu yang disimpan, pemutakhiran gagal dan kesalahan dipublikasikan ke file. `upgrade-prechecks.log` Untuk mengatasi masalah ini, Anda harus memperbarui definisi objek ini dan melampirkan referensi tersebut dalam tanda kutip tunggal (') sebelum memutakhirkan. Untuk informasi selengkapnya tentang melarikan diri dari kata-kata yang dicadangkan di MySQL, lihat String literal dalam dokumentasi MySQL.

Atau, Anda dapat mengubah nama ke nama lain, yang mungkin memerlukan perubahan aplikasi.

Contoh keluaran:

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "description": "The following objects did not pass a syntax check with the latest
 MySQL grammar. A common reason is that they reference names that conflict with new
 reserved keywords. You must update these routine definitions and `quote` any such
 references before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
         "level": "Error",
         "dbObject": "test.select_res_word",
         "description": "at line 2,18: unexpected token 'except'",
         "dbObjectType": "Routine"
      }
  ]
}
```

Untuk memperbaiki masalah ini, periksa definisi rutin.

```
SHOW CREATE PROCEDURE test.select_res_word\G

*************************** 1. row ***************************
           Procedure: select_res_word
            sql_mode:
 ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_Z
    Create Procedure: CREATE PROCEDURE 'select_res_word'()
BEGIN
    SELECT * FROM except;
END
character_set_client: utf8
collation_connection: utf8_general_ci
```

```
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Prosedur ini menggunakan tabel bernama`except`, yang merupakan kata kunci baru di MySQL 8.0. Buat ulang prosedur dengan melarikan diri dari string literal.

```
> drop procedure if exists select_res_word;
Query OK, 0 rows affected (0.00 sec)

> DELIMITER $$
 > CREATE PROCEDURE select_res_word()
    -> BEGIN
    ->     SELECT * FROM 'except';
    -> END$$
Query OK, 0 rows affected (0.00 sec)

 > DELIMITER ;
```

Precheck sekarang berlalu.

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "detectedProblems": []
}
```

schemaInconsistencyCheck

Tingkat precheck: Kesalahan

Ketidakkonsistenan skema yang dihasilkan dari penghapusan file atau korupsi

Seperti dijelaskan sebelumnya, MySQL 8.0 memperkenalkan Atomic Data Dictionary, yang menyimpan semua metadata dalam satu set tabel InnoDB internal dalam skema. `mysql` Arsitektur baru ini menyediakan cara transaksional, sesuai ACID untuk mengelola metadata database, memecahkan masalah “DDL atom” dari pendekatan berbasis file lama. Sebelum MySQL 8.0, objek skema mungkin menjadi yatim piatu jika operasi DDL tiba-tiba terganggu. Migrasi metadata berbasis file ke tabel Atomic Data Dictionary baru selama upgrade memastikan bahwa tidak ada objek skema yatim piatu seperti itu dalam instance DB. Jika ada objek yatim piatu yang ditemui, pemutakhiran gagal.

Untuk membantu mendeteksi objek yatim piatu ini sebelum memulai pemutakhiran, `schemaInconsistencyCheck` precheck dijalankan untuk memastikan bahwa semua objek metadata kamus data disinkronkan. Jika ada objek metadata yatim piatu yang terdeteksi, pemutakhiran tidak dilanjutkan. Untuk melanjutkan dengan upgrade, bersihkan objek metadata yatim piatu ini.

Jika Anda menemukan kesalahan dengan precheck ini, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.

Contoh keluaran:

```
{
  "id": "schemaInconsistencyCheck",
  "title": "Schema inconsistencies resulting from file removal or corruption",
  "status": "OK",
  "description": "Error: Following tables show signs that either table datadir
 directory or frm file was removed/corrupted. Please check server logs, examine
 datadir to detect the issue and fix it before upgrade",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.schemaInconsistencyCheck_failure",
        "description": "present in INFORMATION_SCHEMA's INNODB_SYS_TABLES table but
 missing from TABLES table"
      }
  ]
}
```

Precheck melaporkan bahwa `test.schemaInconsistencyCheck_failure` tabel memiliki metadata yang tidak konsisten. Dalam hal ini, tabel ada di metadata mesin penyimpanan InnoDB (`information_schema.INNODB_SYS_TABLES`), tetapi tidak di metadata MySQL (). `information_schema.TABLES`

Aurora MySQL memeriksa sebelumnya yang melaporkan kesalahan

Precheck berikut khusus untuk Aurora MySQL:

- AuroraCheck DDLRecovery

- auroraCheckRdsUpgradePrechecksTable
- Aurora Cek FODUpgrade
- auroraGetDanglingFulltextIndex
- auroraUpgradeCheckForDatafilePathInconsistency
- auroraUpgradeCheckForFtsSpaceIdZero
- auroraUpgradeCheckForIncompleteXATransactions
- auroraUpgradeCheckForInstanceLimit
- auroraUpgradeCheckForInternalUsers
- auroraUpgradeCheckForMasterUser
- auroraUpgradeCheckForPrefixIndexOnGeometryColumns
- auroraUpgradeCheckForSpecialCharactersInProcedures
- auroraUpgradeCheckForSysSchemaObjectTypeMismatch
- auroraUpgradeCheckForViewColumnNameLength
- auroraUpgradeCheckIndexLengthLimitOnTinyColumns
- auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable
- auroraUpgradeCheckForInvalidUtf8mb3 ColumnComments

AuroraCheck DDLRecovery

Tingkat precheck: Kesalahan

Periksa artefak yang terkait dengan fitur pemulihan Aurora DDL

Sebagai bagian dari implementasi pemulihan Data Definition Language (DDL) di Aurora MySQL, metadata pada pernyataan DDL dalam pesawat dipertahankan dalam dan tabel dalam skema. `ddl_log_md_table` `ddl_log_table` `mysql` Implementasi pemulihan DDL Aurora tidak didukung untuk versi 3 dan seterusnya, karena fungsinya merupakan bagian dari implementasi Atomic Data Dictionary baru di MySQL 8.0. Jika Anda memiliki pernyataan DDL yang berjalan selama pemeriksaan kompatibilitas, precheck ini mungkin gagal. Kami menyarankan Anda mencoba upgrade sementara tidak ada pernyataan DDL yang berjalan.

Jika precheck ini gagal tanpa menjalankan pernyataan DDL, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.

Jika ada pernyataan DDL yang berjalan, output precheck mencetak pesan berikut:

```
“There are DDL statements in process. Please allow DDL statements to finish before
 upgrading.”
```

Contoh keluaran:

```
{
  "id": "auroraCheckDDLRecovery",
  "title": "Check for artifacts related to Aurora DDL recovery feature",
  "status": "OK",
  "description": "Aurora implementation of DDL recovery is not supported from 3.x
 onwards. This check verifies that the database do not have artifacts realted to the
 feature",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_md_table",
        "description": "Table mysql.ddl_log_md_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_table",
        "description": "Table mysql.ddl_log_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "information_schema.processlist",
        "description": "There are DDL statements in process. Please allow DDL
 statements to finish before upgrading."
      }
  ]
}
```

Precheck mengembalikan kesalahan karena DDL dalam pesawat berjalan bersamaan dengan pemeriksaan kompatibilitas. Kami menyarankan Anda mencoba lagi upgrade tanpa DDLs berjalan.

auroraCheckRdsUpgradePrechecksTable

Tingkat precheck: Kesalahan

Periksa keberadaan **`mysql.rds_upgrade_prechecks`** tabel

Ini adalah precheck internal saja yang dilakukan oleh layanan RDS. Kesalahan apa pun akan ditangani secara otomatis saat upgrade dan dapat diabaikan dengan aman.

Jika Anda menemukan kesalahan dengan precheck ini, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.


```
{
  "id": "auroraCheckRdsUpgradePrechecksTable",
  "title": "Check existence of mysql.rds_upgrade_prechecks table",
  "status": "OK",
  "detectedProblems": []
}
```


Aurora Cek FODUpgrade

Tingkat precheck: Kesalahan

Periksa artefak yang terkait dengan fitur DDL cepat Aurora

Optimasi Fast DDL diperkenalkan dalam mode lab pada Aurora MySQL versi 2 untuk meningkatkan efisiensi beberapa operasi DDL. Di Aurora MySQL versi 3, mode lab telah dihapus, dan implementasi Fast DDL telah digantikan oleh fitur MySQL 8.0 yang disebut Instant DDL.

Sebelum memutakhirkan ke Aurora MySQL versi 3, tabel apa pun yang menggunakan Fast DDL dalam mode lab harus dibangun kembali dengan `ALTER TABLE ... ENGINE=InnoDB` menjalankan perintah or untuk memastikan kompatibilitas dengan `OPTIMIZE TABLE` Aurora MySQL versi 3.

Precheck ini mengembalikan daftar tabel tersebut. Setelah tabel yang dikembalikan telah dibangun kembali, Anda dapat mencoba kembali upgrade.

Contoh keluaran:


```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
```

```
  "description": "Aurora fast DDL is not supported from 3.x onwards. This check
 verifies that the database does not have artifacts related to the feature",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/AuroraMySQL.Managing.FastDDL.html#AuroraMySQL.Managing.FastDDL-v2",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test",
        "description": "Your table has pending Aurora fast DDL operations. Run
 'OPTIMIZE TABLE <table name>' for the table to apply all the pending DDL updates.
 Then try the upgrade again."
      }
  ]
}
```

Precheck melaporkan bahwa tabel `test.test` memiliki operasi Fast DDL yang tertunda.

Untuk memungkinkan pemutakhiran dilanjutkan, Anda dapat membangun kembali tabel, lalu coba lagi pemutakhiran.

> **Note**
>
> Sebelum membangun kembali tablespace, lihat Operasi DDL online di dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

```
mysql> optimize table test.test;
+-----------+----------+----------
+-------------------------------------------------------------------+
| Table     | Op       | Msg_type | Msg_text
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
| test.test | optimize | note     | Table does not support optimize, doing recreate
 + analyze instead |
| test.test | optimize | status   | OK
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.04 sec)
```

Setelah membangun kembali tabel, precheck berhasil.

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "detectedProblems": []
}
```

auroraGetDanglingFulltextIndex

Tingkat precheck: Kesalahan

Tabel dengan referensi **FULLTEXT** indeks menggantung

Sebelum MySQL 5.6.26, ada kemungkinan bahwa setelah menjatuhkan indeks pencarian teks lengkap, kolom tersembunyi dan akan menjadi yatim piatu. `FTS_DOC_ID` `FTS_DOC_ID_INDEX` Untuk informasi selengkapnya, lihat Bug #76012.

Jika Anda memiliki tabel yang dibuat pada versi MySQL sebelumnya di mana hal ini terjadi, ini dapat menyebabkan peningkatan ke Aurora MySQL versi 3 gagal. Precheck ini memverifikasi bahwa tidak ada indeks teks lengkap yatim piatu, atau "menggantung" yang ada di cluster DB Anda sebelum memutakhirkan ke MySQL 8.0. Jika precheck ini gagal, buat kembali tabel apa pun yang berisi indeks teks lengkap yang menggantung.

Contoh keluaran:

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "description": "Error: The following tables contain dangling FULLTEXT index which
 is not supported. It is recommended to rebuild the table before upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_fts_index",
        "description": "Table `test.table_with_fts_index` contains dangling FULLTEXT
 index. Kindly recreate the table before upgrade."
      }
  ]
},
```

Precheck melaporkan kesalahan untuk `test.table_with_fts_index` tabel karena berisi indeks teks lengkap yang menggantung. Untuk memungkinkan upgrade untuk melanjutkan, membangun kembali tabel untuk membersihkan tabel tambahan indeks teks lengkap. Gunakan `OPTIMIZE TABLE test.table_with_fts_index` atau`ALTER TABLE test.table_with_fts_index, ENGINE=INNODB`.

Setelah membangun kembali tabel, precheck berlalu.

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForDatafilePathInconsistency

Tingkat precheck: Kesalahan

Periksa ketidakkonsistenan yang terkait dengan jalur **`ibd`** file

Precheck ini hanya berlaku untuk Aurora MySQL versi 3.03.0 dan yang lebih rendah. Jika Anda mengalami kesalahan dengan precheck ini, tingkatkan ke Aurora MySQL versi 3.04 atau lebih tinggi.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForFtsSpaceIdZero

Tingkat precheck: Kesalahan

Periksa indeks teks lengkap dengan ID spasi sebagai nol

Di MySQL, saat Anda menambahkan indeks teks lengkap ke tabel InnoDB, sejumlah ruang tabel indeks tambahan dibuat. Karena bug di versi MySQL sebelumnya, yang diperbaiki di versi 5.6.20,

ada kemungkinan bahwa tabel indeks tambahan ini dibuat di tablespace sistem, bukan tablespace InnoDB mereka sendiri.

Jika ada ruang tabel tambahan seperti itu, pemutakhiran akan gagal. Buat ulang indeks teks lengkap yang disebutkan dalam kesalahan precheck, lalu coba lagi pemutakhiran.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForFtsSpaceIdZero",
  "title": "Check for fulltext index with space id as zero",
  "status": "OK",
  "description": "The auxiliary tables of FTS indexes on the table are created
 in system table-space. Due to this DDL queries executed on MySQL8.0 shall cause
 database unavailability. To avoid that, drop and recreate all the FTS indexes on
 the table or rebuild the table using ALTER TABLE query before the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.fts_space_zero_check",
        "description": " The auxiliary tables of FTS indexes on the table
 'test.fts_space_zero_check' are created in system table-space due to https://
bugs.mysql.com/bug.php?id=72132. In MySQL8.0, DDL queries executed on this table
 shall cause database unavailability. To avoid that, drop and recreate all the
 FTS indexes on the table or rebuild the table using ALTER TABLE query before the
 upgrade."
      }
  ]
},
```

Precheck melaporkan kesalahan untuk `test.fts_space_zero_check` tabel, karena memiliki tabel pencarian teks lengkap (FTS) tambahan di tablespace sistem.

Setelah Anda menjatuhkan dan membuat ulang indeks FTS yang terkait dengan tabel ini, precheck berhasil.

> **Note**
>
> Sebelum membangun kembali ruang tabel, lihat Mempartisi operasi dalam dokumentasi MySQL untuk memahami efek penguncian dan pergerakan data pada transaksi latar depan.

```
{
 "id": "auroraUpgradeCheckForFtsSpaceIdZero",
 "title": "Check for fulltext index with space id as zero",
 "status": "OK",
 "detectedProblems": []
}
```

auroraUpgradeCheckForIncompleteXATransactions

Tingkat precheck: Kesalahan

Periksa transaksi XA dalam keadaan siap

Saat menjalankan proses peningkatan versi utama, penting bahwa instance Aurora MySQL versi 2 DB mengalami shutdown yang bersih. Ini memastikan bahwa semua transaksi dilakukan atau dibatalkan, dan bahwa InnoDB telah membersihkan semua catatan log undo. Karena rollback transaksi diperlukan, jika database Anda memiliki transaksi XA dalam keadaan siap, itu dapat memblokir shutdown bersih dari proses. Untuk alasan ini, jika ada transaksi XA yang disiapkan terdeteksi, peningkatan tidak akan dapat dilanjutkan sampai Anda mengambil tindakan untuk melakukan atau mengembalikannya.

Untuk informasi selengkapnya tentang menemukan transaksi XA dalam keadaan siap menggunakan`XA RECOVER`, lihat pernyataan SQL transaksi XA dalam dokumentasi MySQL. Untuk informasi lebih lanjut tentang status transaksi XA, lihat status transaksi XA dalam dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "description": "Your cluster currently has XA transactions in the prepared state.
 To proceed with the upgrade, commit or rollback these transactions.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your cluster currently has XA transactions in the prepared
 state. To proceed with the upgrade, commit or rollback these transactions."
      }
  ]
```

```
}
```

Precheck ini melaporkan kesalahan karena ada transaksi dalam keadaan siap yang harus dilakukan atau dibatalkan.

Setelah masuk ke database, Anda dapat memeriksa tabel information_schema.innodb_trx dan output untuk informasi lebih lanjut. `XA RECOVER`

> **Important**
>
> Sebelum melakukan atau mengembalikan transaksi, kami sarankan Anda meninjau dokumentasi MySQL dan persyaratan aplikasi Anda.

```
mysql> select trx_started,
    trx_mysql_thread_id,
    trx_id,trx_state,
    trx_operation_state,
    trx_rows_modified,
    trx_rows_locked
from
    information_schema.innodb_trx;
+---------------------+--------------------+---------+-----------
+---------------------+-------------------+-----------------+
| trx_started         | trx_mysql_thread_id | trx_id  | trx_state |
 trx_operation_state | trx_rows_modified | trx_rows_locked |
+---------------------+--------------------+---------+-----------
+---------------------+-------------------+-----------------+
| 2024-08-12 01:09:39 |                   0 | 2849470 | RUNNING   | NULL
    |                 1 |               0 |
+---------------------+--------------------+---------+-----------
+---------------------+-------------------+-----------------+
1 row in set (0.00 sec)

mysql> xa recover;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        1 |            6 |            0 | xatest |
+----------+--------------+--------------+--------+
1 row in set (0.00 sec)
```

Dalam hal ini, kami memutar kembali transaksi yang disiapkan.

```
mysql> XA ROLLBACK 'xatest';
Query OK, 0 rows affected (0.00 sec)
v
mysql> xa recover;
Empty set (0.00 sec)
```

Setelah transaksi XA dibatalkan, precheck berhasil.

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInstanceLimit

Tingkat precheck: Kesalahan

Periksa apakah upgrade didukung pada kelas instance saat ini

Menjalankan pemutakhiran di tempat dari Aurora MySQL versi 2.12.0 atau 2.12.1, di mana kelas instans DB penulis adalah db.r6i.32xlarge, saat ini tidak didukung. Dalam hal ini, precheck mengembalikan kesalahan. Untuk memungkinkan pemutakhiran dilanjutkan, Anda dapat mengubah kelas instans DB Anda menjadi db.r6i.24xlarge atau lebih kecil. Atau Anda dapat meningkatkan ke Aurora MySQL versi 2.12.2 atau lebih tinggi, di mana upgrade di tempat ke Aurora MySQL versi 3 didukung pada db.r6i.32xlarge.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForInstanceLimit",
  "title": "Checks if upgrade is supported on the current instance class",
  "status": "OK",
  "description": "Upgrade from Aurora Version 2.12.0 and 2.12.1 may fail for 32.xl
 and above instance class.",
  "detectedProblems": [
      {
        "level": "Error",
```

```
        "dbObject": "all",
        "description": "Upgrade is not supported on this instance size for Aurora
 MySql Version 2.12.1. Before upgrading to Aurora MySql 3, please consider either:
 1. Changing the instance class to 24.xl or lower. -or- 2. Upgrading to patch
 version 2.12.2 or higher."
      }
  ]
},
```

Precheck mengembalikan kesalahan karena instance DB penulis menggunakan kelas instance db.r6i.32xlarge, dan berjalan pada Aurora MySQL versi 2.12.1.

auroraUpgradeCheckForInternalUsers

Tingkat precheck: Kesalahan

Periksa 8.0 pengguna internal

Precheck ini hanya berlaku untuk Aurora MySQL versi 3.03.0 dan yang lebih rendah. Jika Anda mengalami kesalahan dengan precheck ini, tingkatkan ke Aurora MySQL versi 3.04 atau lebih tinggi.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForInternalUsers",
  "title": "Check for 8.0 internal users.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForMasterUser

Tingkat precheck: Kesalahan

Periksa apakah pengguna master RDS ada

MySQL 8 telah menambahkan model hak istimewa baru dengan dukungan untuk peran dan hak istimewa dinamis untuk membuat manajemen hak istimewa lebih mudah dan lebih halus. Sebagai bagian dari perubahan ini, Aurora MySQL telah memperkenalkan yang baru`rds_superuser_role`, yang secara otomatis diberikan kepada pengguna master database pada peningkatan dari Aurora MySQL versi 2 ke versi 3.

Untuk informasi selengkapnya tentang peran dan hak istimewa yang diberikan kepada pengguna master di Aurora MySQL, lihat. Hak akses akun pengguna master Untuk informasi lebih lanjut tentang model hak istimewa berbasis peran di Aurora MySQL versi 3, lihat. Model hak akses berbasis peran

Precheck ini memverifikasi bahwa pengguna master ada dalam database. Jika pengguna master tidak ada, precheck gagal. Untuk memungkinkan pemutakhiran dilanjutkan, buat ulang pengguna master dengan mengatur ulang kata sandi pengguna master, atau dengan membuat pengguna secara manual. Kemudian coba lagi upgrade. Untuk informasi selengkapnya tentang mengatur ulang kata sandi pengguna utama, lihatMengubah kata sandi untuk pengguna master database.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  "title": "Check if master user exists",
  "status": "OK",
  "description": "Throws error if master user has been dropped!",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/UsingWithRDS.MasterAccounts.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your Master User on host '%' has been dropped. To proceed
 with the upgrade, recreate the master user `reinvent` on default host '%'"
      }
  ]
}
```

Setelah Anda mengatur ulang kata sandi pengguna utama Anda, precheck akan lulus, dan Anda dapat mencoba kembali upgrade.

Contoh berikut menggunakan AWS CLI untuk mengatur ulang kata sandi. Perubahan kata sandi diterapkan segera.

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --master-user-password my-new-password
```

Kemudian precheck berhasil.

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  title": "Check if master user exists",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForPrefixIndexOnGeometryColumns

Tingkat precheck: Kesalahan

Periksa kolom geometri pada indeks awalan

Pada MySQL 8.0.12, Anda tidak dapat lagi membuat indeks awalan pada kolom menggunakan tipe data GEOMETRY. Untuk informasi lebih lanjut, lihat WL #11808.

Jika ada indeks seperti itu, pemutakhiran Anda akan gagal. Untuk mengatasi masalah ini, letakkan `GEOMETRY` indeks awalan pada tabel yang disebutkan dalam kegagalan precheck.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "description": "Consider dropping the prefix Indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.geom_index_prefix",
        "description": "Table `test`.`geom_index_prefix` has an index `LatLon` on
 geometry column/s. Mysql 8.0 does not support this type of index on a geometry
 column https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run
 'DROP INDEX `LatLon` ON `test`.`geom_index_prefix`;"
      }
  ]
}
```

Precheck melaporkan kesalahan karena `test.geom_index_prefix` tabel berisi indeks dengan awalan pada kolom. `GEOMETRY`

Setelah Anda menjatuhkan indeks ini, precheck berhasil.

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForSpecialCharactersInProcedures

Tingkat precheck: Kesalahan

Periksa ketidakkonsistenan yang terkait dengan karakter khusus dalam prosedur tersimpan

Sebelum MySQL 8.0, nama database, nama tabel, dan objek lain berhubungan dengan file dalam direktori data, yaitu metadata berbasis file. Sebagai bagian dari upgrade ke MySQL 8.0, semua objek database dimigrasikan ke tabel kamus data internal baru yang disimpan dalam `mysql` skema untuk mendukung Kamus Data Atom yang baru diimplementasikan. Sebagai bagian dari migrasi prosedur tersimpan, definisi prosedur dan isi untuk setiap prosedur divalidasi karena dimasukkan ke dalam kamus data baru.

Sebelum MySQL 8, dalam beberapa kasus dimungkinkan untuk membuat rutinitas yang disimpan, atau memasukkan langsung ke dalam tabel, prosedur `mysql.proc` yang berisi karakter khusus. Misalnya, Anda dapat membuat prosedur tersimpan yang berisi komentar dengan karakter spasi yang tidak sesuai dan tidak melanggar. `\xa0` Jika ada prosedur seperti itu ditemui, upgrade gagal.

Precheck ini memvalidasi bahwa isi dan definisi prosedur tersimpan Anda tidak mengandung karakter seperti itu. Untuk memungkinkan peningkatan dilanjutkan, buat ulang prosedur tersimpan ini tanpa karakter tersembunyi atau khusus.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "description": "Following procedure(s) has special characters inconsistency.",
  "detectedProblems": [
      {
        "level": "Error",
```

```
        "dbObject": "information_schema.routines",
        "description": "Data Dictionary Metadata is inconsistent for the procedure
 `get_version_proc` in the database `test` due to usage of special characters in
 procedure body. To avoid that, drop and recreate the procedure without any special
 characters before proceeding with the Upgrade."
      }
  ]
}
```

Precheck melaporkan bahwa cluster DB berisi prosedur yang disebut `get_version_proc` dalam `test` database yang berisi karakter khusus dalam badan prosedur.

Setelah menjatuhkan dan membuat ulang prosedur yang disimpan, precheck berhasil, memungkinkan peningkatan untuk melanjutkan.

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForSysSchemaObjectTypeMismatch

Tingkat precheck: Kesalahan

Periksa ketidakcocokan tipe objek untuk skema **`sys`**

Skema sys adalah sekumpulan objek dan tampilan dalam database MySQL yang membantu pengguna memecahkan masalah, mengoptimalkan, dan memantau instance DB mereka. Saat melakukan upgrade versi utama dari Aurora MySQL versi 2 ke versi 3, tampilan `sys` skema dibuat ulang dan diperbarui ke definisi Aurora MySQL versi 3 yang baru.

Sebagai bagian dari peningkatan, jika ada objek dalam `sys` skema yang didefinisikan menggunakan mesin penyimpanan (`sys_config/BASE TABLE`di INFORMATION_SCHEMA.TABLES), bukan tampilan, pemutakhiran akan gagal. Tabel semacam itu dapat ditemukan di `information_schema.tables` tabel. Ini bukan perilaku yang diharapkan, tetapi dalam beberapa kasus dapat terjadi karena kesalahan pengguna.

Precheck ini memvalidasi semua objek `sys` skema untuk memastikan bahwa mereka menggunakan definisi tabel yang benar, dan bahwa tampilan tidak salah didefinisikan sebagai

tabel InnoDB atau MyISAM. Untuk mengatasi masalah ini, perbaiki objek yang dikembalikan secara manual dengan mengganti nama atau menjatuhkannya. Kemudian coba lagi upgrade.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "description": "Database contains objects with type mismatch for sys schema.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "sys.waits_global_by_latency",
        "description": "Your object sys.waits_global_by_latency has a type mismatch.
 To fix the inconsistency we recommend to rename or remove the object before
 upgrading (use RENAME TABLE command). "
      }
  ]
}
```

Precheck melaporkan bahwa tampilan sys.waits_global_by_latency dalam skema memiliki ketidakcocokan tipe yang memblokir pemutakhiran agar tidak melanjutkan. `sys`

Setelah masuk ke instance DB, Anda dapat melihat bahwa objek ini didefinisikan sebagai tabel InnoDB, ketika seharusnya tampilan.

```
mysql> show create table sys.waits_global_by_latency\G
*************************** 1. row ***************************
       Table: waits_global_by_latency
Create Table: CREATE TABLE `waits_global_by_latency` (
  `events` varchar(128) DEFAULT NULL,
  `total` bigint(20) unsigned DEFAULT NULL,
  `total_latency` text,
  `avg_latency` text,
  `max_latency` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

Untuk mengatasi masalah ini, kami dapat menghapus dan membuat ulang tampilan dengan definisi yang benar atau mengganti namanya. Selama proses upgrade, itu akan secara otomatis dibuat dengan definisi tabel yang benar.

```
mysql> RENAME TABLE sys.waits_global_by_latency to sys.waits_global_by_latency_old;
Query OK, 0 rows affected (0.01 sec)
```

Setelah melakukan ini, precheck berlalu.

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForViewColumnNameLength

Tingkat precheck: Kesalahan

Periksa batas atas untuk nama kolom dalam tampilan

Panjang maksimum yang diizinkan dari nama kolom di MySQL adalah 64 karakter. Sebelum MySQL 8.0, dalam beberapa kasus dimungkinkan untuk membuat tampilan dengan nama kolom yang lebih besar dari 64 karakter. Jika ada tampilan seperti itu pada instance database Anda, kesalahan precheck dikembalikan, dan pemutakhiran akan gagal. Untuk memungkinkan pemutakhiran dilanjutkan, Anda harus membuat ulang tampilan yang dimaksud, memastikan bahwa panjang kolomnya kurang dari 64 karakter. Kemudian coba lagi upgrade.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "description": "Following view(s) has column(s) with length greater than 64.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject":
 "test.colname_view_test.col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pa
        "description": "View `test`.`colname_view_test`has column
 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` with
 invalid column name length. To avoid Upgrade errors, view should be altered by
 renaming the column name so that its length is not 0 and doesn't exceed 64."
      }
```

```
  ]
}
```

Precheck melaporkan bahwa `test.colname_view_test` tampilan berisi kolom `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` yang melebihi panjang kolom maksimum yang diizinkan dari 64 karakter.

Melihat definisi tampilan, kita bisa melihat kolom yang menyinggung.

```
mysql> desc `test`.`colname_view_test`;
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| Field                                                             | Type        |
 Null | Key | Default | Extra |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| col1                                                              | varchar(20) |
 YES  |     | NULL    |       |
| col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad | int(11)     |
 YES  |     | NULL    |       |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Untuk memungkinkan pemutakhiran dilanjutkan, buat ulang tampilan, pastikan panjang kolom tidak melebihi 64 karakter.

```
mysql> drop view `test`.`colname_view_test`;
Query OK, 0 rows affected (0.01 sec)

mysql> create view `test`.`colname_view_test`(col1, col2_nopad) as select inf,
 fodcol from test;
Query OK, 0 rows affected (0.01 sec)

mysql> desc `test`.`colname_view_test`;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| col1       | varchar(20) | YES  |     | NULL    |       |
| col2_nopad | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Setelah melakukan ini, precheck berhasil.

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckIndexLengthLimitOnTinyColumns

Tingkat precheck: Kesalahan

Periksa tabel dengan indeks yang didefinisikan dengan panjang awalan lebih besar dari 255 byte pada kolom kecil

Saat membuat indeks pada kolom menggunakan tipe data biner di MySQL, Anda harus menambahkan panjang awalan dalam definisi indeks. Sebelum MySQL 8.0, dalam beberapa kasus dimungkinkan untuk menentukan panjang awalan yang lebih besar dari ukuran maksimum yang diizinkan dari tipe data tersebut. Contohnya adalah `TINYTEXT` dan `TINYBLOB` kolom, di mana ukuran data maksimum yang diizinkan adalah 255 byte, tetapi awalan indeks yang lebih besar dari ini diizinkan. Untuk informasi selengkapnya, lihat batas InnoDB dalam dokumentasi MySQL.

Jika precheck ini gagal, jatuhkan indeks yang menyinggung atau kurangi panjang awalan `TINYTEXT` dan `TINYBLOB` kolom indeks menjadi kurang dari 255 byte. Kemudian coba lagi upgrade.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "description": "Prefix length of the indexes defined on tiny columns cannot
 exceed 255 bytes. With utf8mb4 char set, this limits the prefix length supported
 upto 63 characters only. A larger prefix length was allowed in MySQL5.7 using
 innodb_large_prefix parameter. This parameter is deprecated in MySQL 8.0.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/innodb-limits.html,
 https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html",
  "detectedProblems": [
```

```
      {
        "level": "Error",
        "dbObject": "test.tintxt_prefixed_index.col1",
        "description": "Index 'PRIMARY' on tinytext/tinyblob column `col1` of table
 `test.tintxt_prefixed_index` is defined with prefix length exceeding 255 bytes.
 Reduce the prefix length to <=255 bytes depending on character set used. For
 utf8mb4, it should be <=63."
      }
  ]
}
```

Precheck melaporkan kesalahan untuk `test.tintxt_prefixed_index` tabel, karena memiliki Indeks `PRIMARY` yang memiliki awalan lebih besar dari 255 byte pada kolom TINYTEXT atau TINYBLOB.

Melihat definisi untuk tabel ini, kita dapat melihat bahwa kunci utama memiliki awalan 65 pada `TINYTEXT` kolom`col1`. Karena tabel didefinisikan menggunakan set `utf8mb4` karakter, yang menyimpan 4 byte per karakter, awalan melebihi batas 255-byte.

```
mysql> show create table `test`.`tintxt_prefixed_index`\G
*************************** 1. row ***************************
       Table: tintxt_prefixed_index
Create Table: CREATE TABLE `tintxt_prefixed_index` (
  `col1` tinytext NOT NULL,
  `col2` tinytext,
  `col_id` tinytext,
  PRIMARY KEY (`col1`(65))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC
1 row in set (0.00 sec)
```

Memodifikasi awalan indeks ke 63 saat menggunakan set `utf8mb4` karakter akan memungkinkan peningkatan untuk melanjutkan.

```
mysql> alter table `test`.`tintxt_prefixed_index` drop PRIMARY KEY, ADD  PRIMARY KEY
 (`col1`(63));
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Setelah melakukan ini, precheck berhasil.

```
{
```

```
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

Tingkat precheck: Kesalahan

Periksa inkonsistensi metadata InnoDB yang hilang untuk tabel **`mysql.host`**

Ini adalah precheck internal saja yang dilakukan oleh layanan RDS. Kesalahan apa pun akan ditangani secara otomatis saat upgrade dan dapat diabaikan dengan aman.

Jika Anda menemukan kesalahan dengan precheck ini, buka kasus dengan AWS Support untuk meminta agar inkonsistensi metadata diselesaikan. Atau, Anda dapat mencoba kembali upgrade dengan melakukan dump logis, kemudian mengembalikan ke cluster Aurora MySQL versi 3 DB baru.

auroraUpgradeCheckForInvalidUtf8mb3 ColumnComments

Tingkat precheck: Kesalahan

Periksa komentar kolom utf8mb3 yang tidak valid

Precheck ini mengidentifikasi tabel yang berisi komentar kolom dengan pengkodean karakter utf8mb3 yang tidak valid. Di MySQL 8.0, validasi yang lebih ketat diterapkan pada pengkodean karakter dalam metadata, termasuk komentar kolom. Jika ada komentar kolom yang berisi karakter yang tidak valid dalam set karakter utf8mb3, pemutakhiran akan gagal.

Penyebab paling umum dari masalah ini adalah adanya karakter non-BMP (Basic Multilingual Plane) di kolom komentar. Karakter non-BMP memerlukan pengkodean UTF-8 4-byte (utf8mb4), tetapi utf8mb3 hanya mendukung urutan 3-byte, yang tidak dapat mewakili karakter ini.

Untuk mengatasi masalah ini, Anda harus memodifikasi kolom komentar untuk menghapus atau mengganti karakter non-BMP sebelum mencoba upgrade. Anda dapat menggunakan `ALTER TABLE` pernyataan untuk memperbarui kolom komentar.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "description": "Following table(s) has/have invalid utf8mb3 comments on the
 column/columns.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.t2",
        "description": "Table test.t2 has invalid utf8mb3 comments in it's column/
columns. This is due to non-BMP characters in the comment field. To fix the
 inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
      }
  ]
}
```

Precheck melaporkan bahwa `test.t2` tabel berisi karakter utf8mb3 yang tidak valid dalam satu atau beberapa kolom komentar, khususnya karena adanya karakter non-BMP.

Untuk mengatasi masalah ini, Anda dapat mengidentifikasi kolom bermasalah dan memperbarui komentar mereka. Pertama, periksa struktur tabel untuk mengidentifikasi kolom dengan komentar:

```
mysql> SHOW CREATE TABLE test.t2\G
```

Setelah Anda mengidentifikasi kolom dengan komentar bermasalah, perbarui dengan menggunakan `ALTER TABLE` pernyataan. Misalnya:

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT 'Updated
 comment without non-BMP characters';
```

Atau, Anda dapat menghapus komentar sepenuhnya:

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT '';
```

Setelah memperbarui semua komentar kolom yang bermasalah, precheck akan lulus dan peningkatan dapat dilanjutkan:

```
{
```

```
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "detectedProblems": []
}
```

> **Note**
>
> Sebelum memodifikasi komentar kolom, pastikan bahwa kode aplikasi atau dokumentasi apa pun yang bergantung pada komentar ini diperbarui sesuai dengan itu. Pertimbangkan untuk bermigrasi ke set karakter utf8mb4 untuk dukungan Unicode yang lebih baik jika aplikasi Anda memerlukan karakter non-BMP.

Peringatan

Prechecks berikut menghasilkan peringatan ketika precheck gagal, tetapi upgrade dapat dilanjutkan.

Topik

- MySQL memeriksa sebelumnya yang melaporkan peringatan
- Aurora MySQL mengecek peringatan laporan itu

MySQL memeriksa sebelumnya yang melaporkan peringatan

Precheck berikut berasal dari MySQL Komunitas:

- defaultAuthenticationPlugin
- maxdbFlagCheck
- mysqlDollarSignNameCheck
- reservedKeywordsCheck
- UTF8MB3Periksa
- zeroDatesCheck

defaultAuthenticationPlugin

Tingkat precheck: Peringatan

Pertimbangan plugin otentikasi default baru

Di MySQL 8.0, plugin otentikasi diperkenalkan, `caching_sha2_password` menyediakan enkripsi kata sandi yang lebih aman dan kinerja yang lebih baik daripada plugin yang tidak digunakan lagi. `mysql_native_password` Untuk Aurora MySQL versi 3, plugin otentikasi default yang digunakan untuk pengguna database adalah plugin. `mysql_native_password`

Precheck ini memperingatkan bahwa plugin ini akan dihapus dan default diubah dalam rilis versi utama masa depan. Pertimbangkan untuk mengevaluasi kompatibilitas klien dan pengguna aplikasi Anda sebelum perubahan ini.

Untuk informasi selengkapnya, lihat masalah kompatibilitas caching_sha2_password dan solusi dalam dokumentasi MySQL.

Contoh keluaran:


```
{
  "id": "defaultAuthenticationPlugin",
  "title": "New default authentication plugin considerations",
  "description": "Warning: The new default authentication plugin
 'caching_sha2_password' offers more secure password hashing than previously
 used 'mysql_native_password' (and consequent improved client connection
 authentication). However, it also has compatibility implications that
 may affect existing MySQL installations. If your MySQL installation
 must serve pre-8.0 clients and you encounter compatibility issues after
 upgrading, the simplest way to address those issues is to reconfigure
 the server to revert to the previous default authentication plugin
 (mysql_native_password). For example, use these lines in the server option file:
\n\n[mysqld]\ndefault_authentication_plugin=mysql_native_password\n\nHowever, the
 setting should be viewed as temporary, not as a long term or permanent solution,
 because it causes new accounts created with the setting in effect to forego the
 improved authentication security.\nIf you are using replication please take time to
 understand how the authentication plugin changes may impact you.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://
dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-
sha2-password-replication"
},
```


maxdbFlagCheck

Tingkat precheck: Peringatan

Penggunaan bendera usang **`MAXDB` `sql_mode`**

Di MySQL 8.0, sejumlah opsi variabel sistem sql_mode yang tidak digunakan lagi telah dihapus, salah satunya adalah. `MAXDB` Precheck ini memeriksa semua sesi yang terhubung saat ini, bersama dengan rutinitas dan pemicu, untuk memastikan bahwa tidak ada yang `sql_mode` mengatur kombinasi apa pun yang disertakan. `MAXDB`

Contoh keluaran:

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "description": "Warning: The following DB objects have the obsolete MAXDB
 option persisted for sql_mode, which will be cleared during the upgrade. It can
 potentially change the datatype DATETIME into TIMESTAMP if it is used inside
 object's definition, and this in turn can change the behavior in case of dates
 earlier than 1970 or later than 2037. If this is a concern, please redefine these
 objects so that they do not rely on the MAXDB flag before running the upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.maxdb_stored_routine",
        "description": "PROCEDURE uses obsolete MAXDB sql_mode",
        "dbObjectType": "Routine"
      }
  ]
}
```

Precheck melaporkan bahwa `test.maxdb_stored_routine` rutinitas berisi opsi yang tidak didukung`sql_mode`.

Setelah masuk ke database, Anda dapat melihat dalam definisi rutin yang `sql_mode` berisi`MAXDB`.

```
 > SHOW CREATE PROCEDURE test.maxdb_stored_routine\G

*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode:
 PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,MAXDB,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPT
```

```
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Untuk mengatasi masalah, buat kembali prosedur setelah mengatur yang benar `sql_mode` pada klien.

> **Note**
>
> Menurut dokumentasi MySQL, MySQL menyimpan `sql_mode` pengaturan yang berlaku ketika rutinitas dibuat atau diubah. Itu selalu menjalankan rutinitas dengan pengaturan ini, terlepas dari `sql_mode` pengaturan ketika rutinitas dimulai.
> Sebelum mengubah`sql_mode`, lihat mode Server SQL dalam dokumentasi MySQL. Hati-hati mengevaluasi dampak potensial dari melakukannya pada aplikasi Anda.

Buat ulang prosedur tanpa yang tidak `sql_mode` didukung.

```
mysql > set sql_mode='PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql > DROP PROCEDURE test.maxdb_stored_routine\G
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER $$
mysql >
mysql > CREATE PROCEDURE test.maxdb_stored_routine()
    -> SQL SECURITY DEFINER
    -> BEGIN
    ->     SELECT * FROM test;
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql >
```

```
mysql > DELIMITER ;
mysql > show create procedure test.maxdb_stored_routine\G
*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode: PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Precheck berhasil.

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "detectedProblems": []
}
```

mysqlDollarSignNameCheck

Tingkat precheck: Peringatan

Periksa penggunaan tanda dolar tunggal yang tidak digunakan lagi dalam nama objek

Pada MySQL 8.0.32, penggunaan tanda dolar $ () sebagai karakter pertama dari pengidentifikasi yang tidak dikutip tidak digunakan lagi. Jika Anda memiliki skema, tabel, tampilan, kolom, atau rutinitas yang berisi $ sebagai karakter pertama, precheck ini mengembalikan peringatan. Meskipun peringatan ini tidak menghalangi peningkatan untuk melanjutkan, kami menyarankan Anda segera mengambil tindakan untuk menyelesaikannya. Dari MySQL 8.4 pengidentifikasi semacam itu akan mengembalikan kesalahan sintaks daripada peringatan.

Contoh keluaran:

```
{
  "id": "mysqlDollarSignNameCheck",
```

```
  "title": "Check for deprecated usage of single dollar signs in object names",
  "status": "OK",
  "description": "Warning: The following objects have names with deprecated usage
 of dollar sign ($) at the begining of the identifier. To correct this warning,
 ensure, that names starting with dollar sign, also end with it, similary to quotes
 ($example$). ",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.$deprecated_syntx",
        "description": " name starts with $ sign."
      }
  ]
},
```

Precheck melaporkan peringatan karena `$deprecated_syntx` tabel dalam `test` skema berisi a $ sebagai karakter pertama.

reservedKeywordsCheck

Tingkat precheck: Peringatan

Penggunaan objek database dengan nama yang bertentangan dengan kata kunci baru yang dicadangkan

Pemeriksaan ini mirip dengan routineSyntaxCheck, karena memeriksa penggunaan objek database dengan nama yang bertentangan dengan kata kunci baru yang dicadangkan. Meskipun mereka tidak memblokir peningkatan, Anda perlu mengevaluasi peringatan dengan hati-hati.

Contoh keluaran:

Menggunakan contoh sebelumnya dengan tabel bernama`except`, precheck mengembalikan peringatan:

```
{
  "id": "reservedKeywordsCheck",
  "title": "Usage of db objects with names conflicting with new reserved keywords",
  "status": "OK",
  "description": "Warning: The following objects have names that conflict with
 new reserved keywords. Ensure queries sent by your applications use `quotes` when
 referring to them or they will result in errors.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
```

```
      {
        "level": "Warning",
        "dbObject": "test.except",
        "description": "Table name",
        "dbObjectType": "Table"
      }
  ]
}
```

Peringatan ini memberi tahu Anda bahwa mungkin ada beberapa kueri aplikasi untuk ditinjau. Jika kueri aplikasi Anda tidak lolos dari literal string dengan benar, mereka mungkin mengalami kesalahan setelah memutakhirkan ke MySQL 8.0. Tinjau aplikasi Anda untuk mengonfirmasi, menguji klon atau snapshot dari cluster DB MySQL Aurora Anda yang berjalan pada versi 3.

Contoh kueri aplikasi yang tidak lolos yang akan gagal setelah memutakhirkan:

```
SELECT * FROM escape;
```

Contoh kueri aplikasi yang lolos dengan benar yang akan berhasil setelah memutakhirkan:

```
SELECT * FROM 'escape';
```

UTF8MB3Periksa

Tingkat precheck: Peringatan

Penggunaan set **`utf8mb3`** karakter

Di MySQL 8.0 `utf8mb3` set karakter tidak digunakan lagi, dan akan dihapus dalam versi utama MySQL future. Precheck ini diimplementasikan untuk memunculkan peringatan jika ada objek database yang menggunakan set karakter ini terdeteksi. Meskipun ini tidak akan memblokir peningkatan dari proses, kami sangat menyarankan Anda berpikir tentang memigrasikan tabel ke set `utf8mb4` karakter, yang merupakan default pada MySQL 8.0. Untuk informasi selengkapnya tentang utf8mb3 dan utf8mb4, lihat Mengonversi antara set karakter Unicode 3-byte dan 4-byte dalam dokumentasi MySQL.

Contoh keluaran:

```
{
```

```
  "id": "utf8mb3",
  "title": "Usage of utf8mb3 charset",
  "status": "OK",
  "description": "Warning: The following objects use the deprecated utf8mb3
 character set. It is recommended to convert them to use utf8mb4 instead, for
 improved Unicode support. The utf8mb3 character is subject to removal in the
 future.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-
utf8mb3.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.t1.col1",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      },
      {
        "level": "Warning",
        "dbObject": "test.t1.col2",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      }
  ]
}
```

Untuk mengatasi masalah ini, Anda membangun kembali objek dan tabel yang direferensikan. Untuk informasi selengkapnya dan prasyarat sebelum melakukannya, lihat Mengonversi antara set karakter Unicode 3-byte dan 4-byte dalam dokumentasi MySQL.

zeroDatesCheck

Tingkat precheck: Peringatan

Nilai tanggal nol, datetime, dan stempel waktu

MySQL sekarang memberlakukan aturan yang lebih ketat mengenai penggunaan nilai nol di kolom tanggal, datetime, dan timestamp. Kami menyarankan Anda menggunakan `NO_ZERO_DATE` SQL mode `NO_ZERO_IN_DATE` dan dalam hubungannya dengan `strict` mode, karena mereka akan digabungkan dengan `strict` mode dalam rilis MySQL future.

Jika `sql_mode` pengaturan untuk salah satu koneksi database Anda, pada saat menjalankan precheck, tidak menyertakan mode ini, peringatan akan muncul di precheck. Pengguna mungkin masih dapat menyisipkan nilai tanggal, datetime, dan stempel waktu yang berisi nilai nol. Namun,

kami sangat menyarankan untuk mengganti nilai nol dengan nilai yang valid, karena perilakunya mungkin berubah di masa mendatang dan mungkin tidak berfungsi dengan benar. Karena ini adalah peringatan, ini tidak akan memblokir peningkatan, tetapi kami menyarankan Anda mulai merencanakan untuk mengambil tindakan.

Contoh keluaran:


```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
  ]
}
```


Aurora MySQL mengecek peringatan laporan itu

Precheck berikut khusus untuk Aurora MySQL:

- auroraUpgradeCheckForRollbackSegmentHistoryLength
- auroraUpgradeCheckForUncommittedRowModifications

auroraUpgradeCheckForRollbackSegmentHistoryLength

Tingkat precheck: Peringatan

Memeriksa apakah panjang daftar riwayat segmen rollback untuk cluster tinggi

Seperti disebutkan dalam auroraUpgradeCheckForIncompleteXATransactions, saat menjalankan proses peningkatan versi utama, penting bahwa instance Aurora MySQL versi 2 DB mengalami shutdown yang bersih. Ini memastikan bahwa semua transaksi dilakukan atau dibatalkan, dan bahwa InnoDB telah membersihkan semua catatan log undo.

Jika cluster DB Anda memiliki panjang daftar riwayat segmen rollback (HLL) yang tinggi, itu dapat memperpanjang jumlah waktu yang dibutuhkan InnoDB untuk menyelesaikan pembersihan catatan log batal, yang menyebabkan waktu henti yang diperpanjang selama proses peningkatan versi utama. Jika precheck mendeteksi bahwa HLL pada cluster DB Anda tinggi, itu menimbulkan peringatan. Meskipun ini tidak menghalangi peningkatan Anda untuk melanjutkan, kami sarankan Anda memantau HLL di cluster DB Anda dengan cermat. Menjaga pada tingkat rendah mengurangi waktu henti yang diperlukan selama peningkatan versi utama. Untuk informasi lebih lanjut tentang pemantauan HLL, lihatPanjang daftar riwayat InnoDB meningkat secara signifikan.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForRollbackSegmentHistoryLength",
  "title": "Checks if the rollback segment history length for the cluster is high",
  "status": "OK",
  "description": "Rollback Segment History length is greater than 1M. Upgrade may
 take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_metrics",
        "description": "The InnoDB undo history list length('trx_rseg_history_len')
 is 82989114. Upgrade may take longer due to purging of undo information for old row
 versions."
      }
  ]
}
```

Precheck mengembalikan peringatan karena mendeteksi InnoDB undo HLL tinggi pada cluster database (82989114). Meskipun upgrade berlangsung, tergantung pada jumlah undo untuk membersihkan, itu dapat memperpanjang downtime yang diperlukan selama proses upgrade.

Kami menyarankan Anda menyelidiki transaksi terbuka pada cluster DB Anda sebelum menjalankan upgrade dalam produksi, untuk memastikan HLL Anda disimpan pada ukuran yang dapat dikelola.

auroraUpgradeCheckForUncommittedRowModifications

Tingkat precheck: Peringatan

Memeriksa apakah ada banyak modifikasi baris yang tidak berkomitmen

Seperti disebutkan dalam auroraUpgradeCheckForIncompleteXATransactions, saat menjalankan proses peningkatan versi utama, penting bahwa instance Aurora MySQL versi 2 DB mengalami shutdown yang bersih. Ini memastikan bahwa semua transaksi dilakukan atau dibatalkan, dan bahwa InnoDB telah membersihkan semua catatan log undo.

Jika cluster DB Anda memiliki transaksi yang telah memodifikasi sejumlah besar baris, itu dapat memperpanjang jumlah waktu yang dibutuhkan InnoDB untuk menyelesaikan rollback transaksi ini sebagai bagian dari proses shutdown bersih. Jika precheck menemukan transaksi yang berjalan lama, dengan sejumlah besar baris yang dimodifikasi pada cluster DB Anda, itu menimbulkan peringatan. Meskipun ini tidak menghalangi peningkatan Anda untuk melanjutkan, kami sarankan Anda memantau dengan cermat ukuran transaksi aktif di cluster DB Anda. Menjaga pada tingkat rendah mengurangi waktu henti yang diperlukan selama peningkatan versi utama.

Contoh keluaran:

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "description": "Database contains uncommitted row changes greater than 10M.
 Upgrade may take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_trx",
        "description": "The database contains 11000000 uncommitted row change(s) in
 1 transaction(s). Upgrade may take longer due to transaction rollback."
      }
  ]
},
```

Precheck melaporkan bahwa cluster DB berisi transaksi dengan 11.000.000 perubahan baris tanpa komitmen yang perlu dibatalkan selama proses shutdown bersih. Upgrade akan dilanjutkan, tetapi untuk mengurangi downtime selama proses upgrade, kami sarankan Anda memantau dan menyelidiki ini sebelum menjalankan upgrade pada cluster produksi Anda.

Untuk melihat transaksi aktif pada instance DB penulis Anda, Anda dapat menggunakan tabel information_schema.innodb_trx. Kueri berikut pada instance DB penulis menunjukkan transaksi saat ini, waktu berjalan, status, dan baris yang dimodifikasi untuk cluster DB.

```
# Example of uncommitted transaction
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state | rows_modified |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| 2024-08-12 18:32:52 |                          1592 |
 20041 | 52866130 | RUNNING   |      11000000 |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
1 row in set (0.01 sec)

# Example of transaction rolling back
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+-------------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state    | rows_modified |
+---------------------+-----------------------------
+---------------------------------+----------+-------------+---------------+
```

```
| 2024-08-12 18:32:52 |                          1719 |
 20041 | 52866130 | ROLLING BACK |      10680479 |
+---------------------+-------------------------------
+--------------------------------+---------+-------------+--------------+
1 row in set (0.01 sec)
```

Setelah transaksi dilakukan atau dibatalkan, precheck tidak lagi mengembalikan peringatan. Konsultasikan dokumentasi MySQL dan tim aplikasi Anda sebelum mengembalikan transaksi besar apa pun, karena rollback dapat memakan waktu untuk diselesaikan, tergantung pada ukuran transaksi.

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "detectedProblems": []
},
```

Untuk informasi selengkapnya tentang mengoptimalkan manajemen transaksi InnoDB, dan dampak potensial dari menjalankan dan mengembalikan transaksi besar pada instance database MySQL, lihat Mengoptimalkan manajemen transaksi InnoDB dalam dokumentasi MySQL.

Pemberitahuan

Precheck berikut menghasilkan pemberitahuan ketika precheck gagal, tetapi upgrade dapat dilanjutkan.

sqlModeFlagMemeriksa

Tingkat precheck: Pemberitahuan

Penggunaan bendera usang **`sql_mode`**

Selain ituMAXDB, sejumlah `sql_mode` opsi lain telah dihapus:DB2,,MSSQL,MYSQL323,MYSQL40,ORACLE,POSTGRESQL, NO_FIELD_OPTIONSNO_KEY_OPTIONS, danNO_TABLE_OPTIONS. Pada MySQL 8.0, tidak satu pun dari nilai-nilai ini dapat ditetapkan ke variabel sistem. `sql_mode` Jika precheck ini menemukan sesi terbuka menggunakan `sql_mode` pengaturan ini, pastikan bahwa instans DB dan grup parameter cluster DB Anda, serta aplikasi dan konfigurasi klien, diperbarui untuk menonaktifkannya.- Untuk informasi lebih lanjut, lihat dokumentasi MySQL.

Contoh keluaran:

```
{
  "id": "sqlModeFlagCheck",
  "title": "Usage of obsolete sql_mode flags",
  "status": "OK",
  "detectedProblems": []
}
```

Untuk mengatasi salah satu kegagalan precheck ini, lihat maxdbFlagCheck.

Kesalahan, peringatan, atau pemberitahuan

Precheck berikut dapat mengembalikan kesalahan, peringatan, atau pemberitahuan tergantung pada output precheck.

checkTableOutput

Tingkat precheck: Kesalahan, Peringatan, atau Pemberitahuan

Masalah yang dilaporkan oleh **`check table x for upgrade`** perintah

Sebelum memulai upgrade ke Aurora MySQL versi 3, `check table for upgrade` dijalankan pada setiap tabel dalam skema pengguna di cluster DB Anda. Precheck ini tidak sama dengan checkTableMysqlSchema.

`check table for upgrade`Perintah memeriksa tabel untuk setiap potensi masalah yang mungkin timbul selama upgrade ke versi MySQL yang lebih baru. Menjalankan perintah ini sebelum mencoba upgrade dapat membantu mengidentifikasi dan menyelesaikan ketidakcocokan sebelumnya, membuat proses upgrade yang sebenarnya lebih lancar.

Perintah ini melakukan berbagai pemeriksaan pada setiap tabel, seperti berikut ini:

- Memverifikasi bahwa struktur tabel dan metadata kompatibel dengan versi MySQL target
- Memeriksa fitur yang tidak digunakan lagi atau dihapus yang digunakan oleh tabel
- Memastikan bahwa tabel dapat ditingkatkan dengan benar tanpa kehilangan data

Tidak seperti prechecks lainnya, ini dapat mengembalikan kesalahan, peringatan, atau pemberitahuan tergantung pada `check table` output. Jika precheck ini mengembalikan tabel apa pun, tinjau dengan cermat, bersama dengan kode pengembalian dan pesan sebelum

memulai pemutakhiran. Untuk informasi selengkapnya, lihat pernyataan CHECK TABLE dalam dokumentasi MySQL.

Di sini kami memberikan contoh kesalahan dan contoh peringatan.

Contoh kesalahan:

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.parent",
        "description": "Table 'test.parent' doesn't exist"
      }
  ]
},
```

Precheck melaporkan kesalahan bahwa `test.parent` tabel tidak ada.

`mysql-error.log`File untuk instance DB penulis menunjukkan bahwa ada kesalahan kunci asing.

```
2024-08-13T15:32:10.676893Z 62 [Warning] InnoDB: Load table `test`.`parent` failed,
 the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try
 again.
2024-08-13T15:32:10.676905Z 62 [Warning] InnoDB: Cannot open table test/parent from
 the internal data dictionary of InnoDB though the .frm file for the table exists.
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-troubleshooting.html
 for how to resolve the issue.
```

Masuk ke instans DB penulis dan jalankan `show engine innodb status\G` untuk mendapatkan informasi lebih lanjut tentang kesalahan kunci asing.

```
mysql> show engine innodb status\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
```

```
2024-08-13 15:33:33 0x14ef7b8a1700 INNODB MONITOR OUTPUT
=====================================
.
.
.
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2024-08-13 15:32:10 0x14ef6dbbb700 Error in foreign key constraint of table test/
child:
there is no index in referenced table which would contain
the columns as the first columns, or the data types in the
referenced table do not match the ones in table. Constraint:
,
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
The index in the foreign key in table is p_name_idx
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-foreign-key-
constraints.html for correct foreign key definition.
.
.
```

`LATEST FOREIGN KEY ERROR`Pesan melaporkan bahwa kendala kunci `fk_pname` asing dalam `test.child` tabel, yang mereferensikan `test.parent` tabel, memiliki indeks yang hilang atau ketidakcocokan tipe data. Dokumentasi MySQL pada batasan kunci asing menyatakan bahwa kolom yang direferensikan dalam kunci asing harus memiliki indeks terkait, dan kolom induk/anak harus menggunakan tipe data yang sama.

Untuk memverifikasi apakah ini terkait dengan ketidakcocokan indeks atau tipe data yang hilang, masuk ke database dan periksa definisi tabel dengan menonaktifkan sementara variabel sesi foreign_key_checks. Setelah melakukannya, kita dapat melihat bahwa batasan anak di question (`fk_pname`) digunakan `p_name varchar(20) CHARACTER SET latin1 DEFAULT NULL` untuk mereferensikan tabel induk. `name varchar(20) NOT NULL` Tabel induk menggunakan`DEFAULT CHARSET=utf8`, tetapi `p_name` kolom tabel anak menggunakan`latin1`, sehingga kesalahan ketidakcocokan tipe data dilemparkan.

```
mysql> show create table parent\G
ERROR 1146 (42S02): Table 'test.parent' doesn't exist

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
```

```
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> set foreign_key_checks=0;
Query OK, 0 rows affected (0.00 sec)

mysql> show create table parent\G
*************************** 1. row ***************************
       Table: parent
Create Table: CREATE TABLE `parent` (
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

Untuk mengatasi masalah ini, kita dapat mengubah tabel anak untuk menggunakan set karakter yang sama dengan induk, atau mengubah induk untuk menggunakan set karakter yang sama dengan tabel anak. Di sini, karena tabel anak secara eksplisit menggunakan `latin1` dalam definisi p_name kolom, kami menjalankan `ALTER TABLE` untuk memodifikasi set karakter ke. `utf8`

```
mysql> alter table child modify p_name varchar(20) character set utf8 DEFAULT NULL;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> flush tables;
Query OK, 0 rows affected (0.01 sec)
```

Setelah melakukannya, precheck berlalu, dan peningkatan dapat dilanjutkan.

Contoh peringatan:

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.orders",
        "description": "Trigger test.orders.delete_audit_trigg does not have CREATED
 attribute."
      }
  ]
}
```

Precheck melaporkan peringatan untuk `delete_audit_trigg` pemicu di `test.orders` atas meja karena tidak memiliki `CREATED` atribut. Menurut Memeriksa kompatibilitas versi dalam dokumentasi MySQL, pesan ini bersifat informasi, dan dicetak untuk pemicu yang dibuat sebelum MySQL 5.7.2.

Karena ini adalah peringatan, itu tidak menghalangi peningkatan untuk melanjutkan. Namun, jika Anda ingin menyelesaikan masalah, Anda dapat membuat ulang pemicu yang dimaksud, dan precheck berhasil tanpa peringatan.

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": []
},
```

Cara melakukan peningkatan di tempat

Sebaiknya Anda meninjau materi latar belakang dalam Cara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL.

Lakukan perencanaan dan pengujian pra-upgrade, seperti yang dijelaskan dalamMerencanakan peningkatan versi mayor untuk klaster Aurora MySQL.

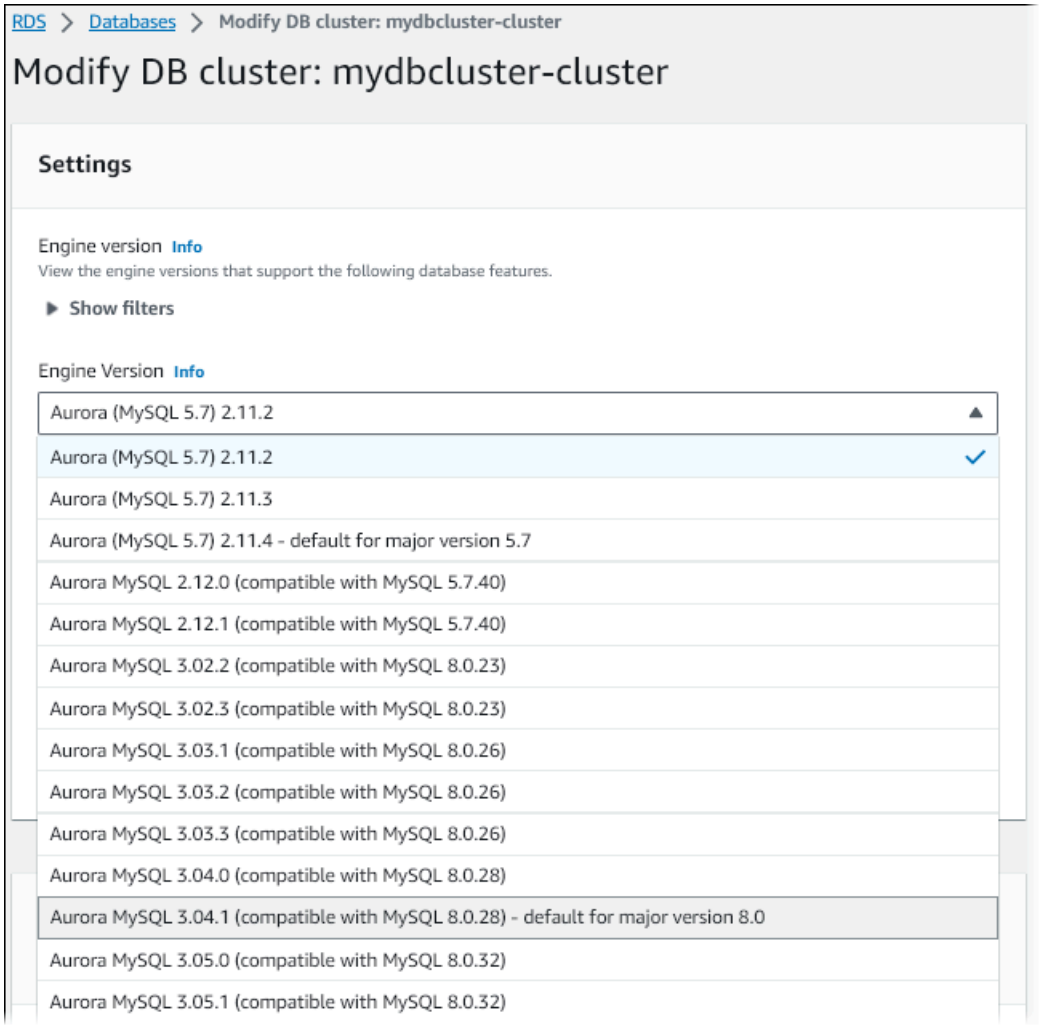
Konsol

Contoh berikut meningkatkan cluster `mydbcluster-cluster` DB ke Aurora MySQL versi 3.04.1.

Untuk meningkatkan versi mayor klaster DB Aurora MySQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Jika Anda menggunakan grup parameter kustom untuk klaster DB asli, buat grup parameter terkait yang kompatibel dengan versi mayor baru. Buat penyesuaian yang diperlukan untuk parameter konfigurasi di grup parameter baru tersebut. Untuk informasi selengkapnya, lihat Bagaimana peningkatan di tempat memengaruhi grup parameter untuk klaster.
3. Di panel navigasi, pilih Basis Data.
4. Dalam daftar, pilih klaster DB yang ingin Anda ubah.
5. Pilih Ubah.
6. Untuk Versi, pilih versi mayor Aurora MySQL baru.

   Biasanya, kami merekomendasikan untuk menggunakan versi minor terbaru dari versi mayor. Di sini, kami memilih versi default saat ini.




7. Pilih Lanjutkan.
8. Di halaman berikutnya, tentukan kapan harus melakukan peningkatan. Pilih Selama jendela pemeliharaan terjadwal berikutnya atau Segera.
9. (Opsional) Periksa secara berkala halaman Peristiwa di konsol RDS selama peningkatan. Tindakan ini akan membantu Anda memantau progres peningkatan dan identifikasi masalah apa pun. Jika peningkatan mengalami masalah apa pun, lihat Pemecahan masalah untuk Aurora Peningkatan di tempat saya SQL untuk langkah-langkah yang harus diambil.
10. Jika Anda membuat grup parameter baru di awal prosedur ini, kaitkan grup parameter kustom dengan klaster yang ditingkatkan. Untuk informasi selengkapnya, lihat Bagaimana peningkatan di tempat memengaruhi grup parameter untuk klaster.

> **Note**
>
> Langkah ini mengharuskan Anda untuk memulai ulang klaster lagi untuk menerapkan grup parameter baru.

11. (Opsional) Setelah Anda menyelesaikan setiap pengujian pasca-peningkatan, hapus snapshot manual yang dibuat Aurora pada awal peningkatan.

AWS CLI

Untuk memutakhirkan versi utama cluster DB MySQL Aurora, gunakan AWS CLI modify-db-clusterperintah dengan parameter yang diperlukan berikut:

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` atau `--no-apply-immediately`

Jika klaster Anda menggunakan grup parameter kustom, sertakan juga salah satu atau kedua opsi berikut ini:

- `--db-cluster-parameter-group-name`, jika klaster menggunakan grup parameter klaster kustom
- `--db-instance-parameter-group-name`, jika ada instans di klaster yang menggunakan grup parameter DB kustom

Contoh berikut meningkatkan cluster `sample-cluster` DB ke Aurora MySQL versi 3.04.1. Peningkatan akan segera terjadi, bukan menunggu periode pemeliharaan berikutnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
          --allow-major-version-upgrade \
```

```
          --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
          --allow-major-version-upgrade ^
          --apply-immediately
```

Anda dapat menggabungkan perintah CLI lainnya `modify-db-cluster` untuk membuat end-to-end proses otomatis untuk melakukan dan memverifikasi peningkatan. Untuk informasi selengkapnya dan contoh tambahan, lihat Aurora Tutorial peningkatan di tempat SQL saya.

> **Note**
>
> Jika klaster Anda adalah bagian dari basis data global Aurora, prosedur peningkatan di tempat ini sedikit berbeda. Anda memanggil operasi modify-global-clusterperintah alih-alih`modify-db-cluster`. Untuk informasi selengkapnya, lihat Peningkatan besar di tempat untuk basis data global.

API RDS

Untuk memutakhirkan versi utama cluster DB MySQL Aurora, gunakan operasi RDS API Modify dengan parameter yang diperlukan berikut: DBCluster

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (atur ke `true` atau `false`)

> **Note**
>
> Jika klaster Anda adalah bagian dari basis data global Aurora, prosedur peningkatan di tempat ini sedikit berbeda. Anda memanggil ModifyGlobalClusteroperasi alih-

alih`ModifyDBCluster`. Untuk informasi selengkapnya, lihat Peningkatan besar di tempat untuk basis data global.

Bagaimana peningkatan di tempat memengaruhi grup parameter untuk klaster

Grup parameter Aurora memiliki kumpulan pengaturan konfigurasi yang berbeda untuk klaster yang kompatibel dengan MySQL 5.7 atau 8.0. Saat Anda melakukan peningkatan di tempat, klaster yang ditingkatkan dan semua instans harus menggunakan grup parameter klaster dan instans yang sesuai:

Klaster dan instans Anda mungkin menggunakan grup parameter default yang kompatibel dengan 5.7. Jika demikian, klaster dan instans yang ditingkatkan akan dimulai dengan grup parameter default yang kompatibel dengan 8.0. Jika klaster dan instans Anda menggunakan grup parameter kustom apa pun, pastikan untuk membuat grup parameter yang sesuai atau yang kompatibel dengan 8.0. Pastikan juga untuk menentukannya selama proses peningkatan.

**Note**

Untuk sebagian besar pengaturan parameter, Anda dapat memilih grup parameter kustom di dua titik. Titik ini adalah saat Anda membuat klaster atau mengaitkan grup parameter dengan klaster nanti.
Namun, jika Anda menggunakan pengaturan nondefault untuk parameter `lower_case_table_names`, Anda harus mengatur grup parameter kustom dengan pengaturan ini terlebih dahulu. Kemudian, tentukan grup parameter saat Anda melakukan pemulihan snapshot untuk membuat klaster. Setiap perubahan pada parameter `lower_case_table_names` tidak akan berpengaruh setelah klaster dibuat.
Kami menyarankan Anda menggunakan pengaturan yang sama untuk `lower_case_table_names` ketika Anda meningkatkan dari Aurora MySQL versi 2 ke versi 3.
Dengan database global Aurora berdasarkan Aurora MySQL, Anda dapat melakukan peningkatan di tempat dari Aurora MySQL versi 2 ke versi 3 hanya jika Anda mengatur parameter ke default dan me-reboot database global Anda. `lower_case_table_names`
Untuk informasi selengkapnya tentang metode yang dapat Anda gunakan, lihat Peningkatan versi utama.

> **Important**
>
> Jika Anda menentukan grup parameter kustom apa pun selama proses peningkatan, pastikan untuk mem-boot ulang klaster secara manual setelah peningkatan selesai. Tindakan ini akan membuat klaster mulai menggunakan pengaturan parameter kustom Anda.

Perubahan pada properti klaster di antara versi Aurora MySQL

Saat Anda meningkatkan dari Aurora MySQL versi 2 ke versi 3, pastikan untuk memeriksa aplikasi atau skrip apa pun yang Anda gunakan untuk menyiapkan atau mengelola klaster dan instans DB Aurora MySQL.

Selain itu, ubah kode Anda yang memanipulasi grup parameter untuk memperhitungkan fakta bahwa nama grup parameter default berbeda untuk klaster yang kompatibel dengan 5.7 dan 8.0. Nama grup parameter default untuk klaster Aurora MySQL versi 2 dan 3 masing-masing adalah `default.aurora-mysql5.7` dan `default.aurora-mysql8.0`.

Misalnya, Anda mungkin memiliki kode seperti berikut yang berlaku untuk klaster Anda sebelum peningkatan.

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

Setelah meningkatkan versi mayor klaster, ubah kode tersebut sebagai berikut.

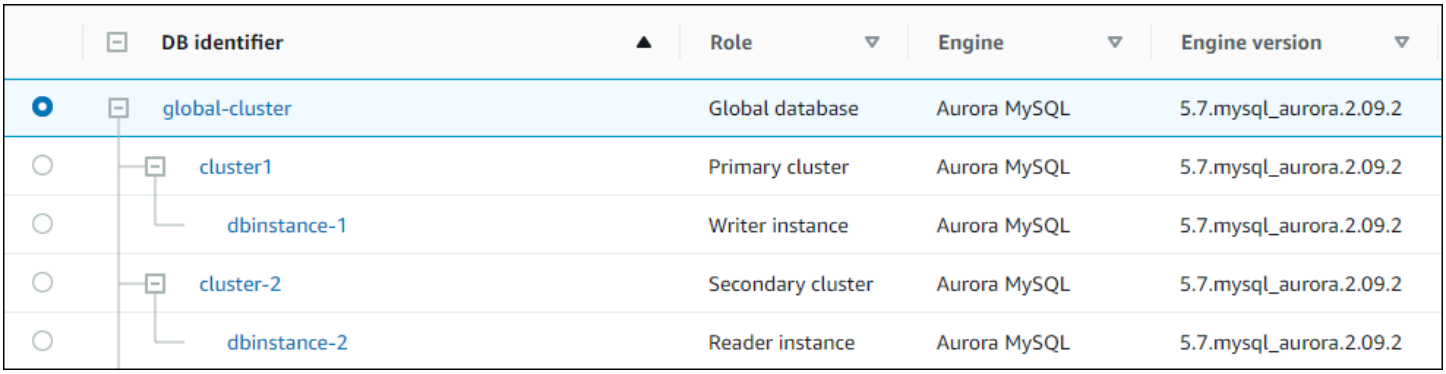
```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

Peningkatan besar di tempat untuk basis data global

Untuk basis data global Aurora, Anda meningkatkan klaster basis data global. Aurora secara otomatis meningkatkan semua klaster pada saat yang sama dan memastikan bahwa semua klaster ini menjalankan versi mesin yang sama. Persyaratan ini berlaku karena setiap perubahan pada tabel sistem, format file data, dan sebagainya akan secara otomatis direplikasi ke semua klaster sekunder.

Ikuti petunjuk dalam Cara kerja peningkatan di tempat terhadap versi mayor Aurora MySQL. Saat Anda menentukan hal yang akan ditingkatkan, pastikan untuk memilih klaster basis data global, bukan salah satu klaster yang terdapat di dalamnya.

Jika Anda menggunakan AWS Management Console, pilih item dengan peran Database global.



| DB identifier | Role | Engine | Engine version |
|---|---|---|---|
| global-cluster | Global database | Aurora MySQL | 5.7.mysql_aurora.2.09.2 |
| cluster1 | Primary cluster | Aurora MySQL | 5.7.mysql_aurora.2.09.2 |
| dbinstance-1 | Writer instance | Aurora MySQL | 5.7.mysql_aurora.2.09.2 |
| cluster-2 | Secondary cluster | Aurora MySQL | 5.7.mysql_aurora.2.09.2 |
| dbinstance-2 | Reader instance | Aurora MySQL | 5.7.mysql_aurora.2.09.2 |

Jika Anda menggunakan AWS CLI atau RDS API, mulai proses upgrade dengan memanggil modify-global-clusterperintah atau ModifyGlobalClusteroperasi. Anda akan menggunakan salah satunya, bukan `modify-db-cluster` atau `ModifyDBCluster`.

> **Note**
>
> Anda tidak dapat menentukan grup parameter kustom untuk klaster basis data global saat Anda melakukan peningkatan versi mayor basis data global Aurora tersebut. Buat grup parameter kustom Anda di setiap Wilayah klaster global. Kemudian, terapkan secara manual ke klaster Regional setelah peningkatan.

Untuk meng-upgrade versi utama dari sebuah cluster database global MySQL Aurora dengan menggunakan AWS CLI, gunakan modify-global-clusterperintah dengan parameter yang diperlukan berikut:

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

Contoh berikut meningkatkan cluster database global ke Aurora MySQL versi 3.04.2.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-global-cluster \
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 8.0.mysql_aurora.3.04.2 \
          --allow-major-version-upgrade
```

Untuk Windows:

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 8.0.mysql_aurora.3.04.2 ^
          --allow-major-version-upgrade
```

Upgrade di tempat untuk cluster DB dengan replika baca lintas wilayah

Anda dapat memutakhirkan cluster Aurora DB yang memiliki replika baca lintas wilayah menggunakan prosedur peningkatan di tempat, tetapi ada pertimbangan tertentu:

- Anda harus memutakhirkan cluster DB replika baca terlebih dahulu. Jika Anda mencoba memutakhirkan klaster utama terlebih dahulu, Anda akan menerima pesan kesalahan seperti berikut ini:

  Tidak dapat memutakhirkan cluster DB test-xr-primary-cluster karena replika Aurora Cross-region terkait test-xr-replica-cluster belum ditambal. Tingkatkan replika Aurora Cross-region dan coba lagi.

  Ini berarti bahwa cluster DB primer tidak dapat memiliki versi mesin DB yang lebih tinggi daripada cluster replika.
- Sebelum Anda memutakhirkan cluster DB primer, hentikan beban kerja tulis dan nonaktifkan permintaan koneksi baru apa pun ke instance DB penulis dari cluster utama.
- Saat Anda memutakhirkan klaster utama, pilih grup parameter cluster DB kustom dengan `binlog_format` parameter yang disetel ke nilai yang mendukung replikasi logging biner, seperti`MIXED`.

  Untuk informasi selengkapnya tentang menggunakan pencatatan log biner dengan Aurora MySQL, lihat Replikasi antara Aurora dan SQL My atau antara Aurora dan cluster Aurora DB lainnya

(replikasi log biner). Untuk informasi selengkapnya tentang memodifikasi parameter konfigurasi Aurora MySQL, lihat Parameter konfigurasi Aurora MySQL dan .

- Jangan menunggu lama untuk memutakhirkan cluster DB utama setelah Anda memutakhirkan cluster replika. Kami menyarankan untuk tidak menunggu lebih lama dari jendela pemeliharaan berikutnya.
- Setelah Anda memutakhirkan cluster DB primer, reboot instance DB penulisnya. Grup parameter cluster DB kustom yang memungkinkan replikasi binlog tidak berlaku sampai instance DB penulis di-boot ulang.
- Jangan melanjutkan beban kerja tulis atau mengaktifkan koneksi ke instans DB penulis sampai Anda mengonfirmasi bahwa replikasi lintas wilayah telah dimulai ulang, dan bahwa jeda replika di sekunder adalah 0. Wilayah AWS

Aurora Tutorial peningkatan di tempat SQL saya

Contoh Linux berikut menunjukkan cara Anda dapat melakukan langkah-langkah umum untuk prosedur peningkatan di tempat menggunakan AWS CLI.

Contoh pertama ini membuat cluster Aurora DB yang menjalankan Aurora My versi 2.x. SQL Klaster ini mencakup instans DB penulis dan instans DB pembaca. Perintah `wait db-instance-available` dijeda hingga instans DB penulis tersedia. Pada saat itulah, klaster siap untuk ditingkatkan.

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.11.2
...
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

Versi Aurora My SQL 3.x yang dapat Anda tingkatkan cluster agar bergantung pada versi 2.x yang sedang dijalankan cluster dan di Wilayah AWS mana cluster berada. Perintah pertama, dengan `--output text`, hanya menunjukkan versi target yang tersedia. Perintah kedua menunjukkan JSON output penuh dari respons. Dalam respons tersebut, Anda dapat melihat detail seperti nilai `aurora-mysql` yang Anda gunakan untuk parameter `engine`. Anda juga dapat melihat fakta bahwa memutakhirkan ke 3.04.0 merupakan peningkatan versi utama.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.11.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.04.0",
    "Description": "Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false,
    "SupportsIntegrations": false
},
...
```

Contoh ini menunjukkan bahwa jika Anda memasukkan nomor versi target yang bukan merupakan target peningkatan yang valid untuk klaster, Aurora tidak akan melakukan peningkatan. Aurora juga tidak melakukan peningkatan versi mayor kecuali jika Anda menyertakan parameter `--allow-major-version-upgrade`. Dengan demikian, Anda tidak dapat tanpa disengaja melakukan peningkatan yang berpotensi akan memerlukan pengujian dan perubahan ekstensif terhadap kode aplikasi Anda.

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.11.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.
```

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --apply-immediately --allow-major-version-
upgrade
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.11.2"
}
```

Dibutuhkan beberapa saat hingga status klaster dan instans DB terkait berubah menjadi `upgrading`. Nomor versi untuk klaster dan instans DB hanya berubah saat peningkatan selesai. Sekali lagi, Anda dapat menggunakan perintah `wait db-instance-available` untuk instans DB penulis untuk menunggu hingga peningkatan selesai sebelum melanjutkan.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
upgrading 5.7.mysql_aurora.2.11.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}

aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

Pada tahap ini, nomor versi untuk klaster akan cocok dengan nomor versi yang ditentukan untuk peningkatan.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.04.0
```

Contoh sebelumnya melakukan peningkatan langsung dengan menentukan parameter `--apply-immediately`. Agar peningkatan terjadi pada waktu yang tepat saat klaster diperkirakan tidak sibuk, Anda dapat menentukan parameter `--no-apply-immediately`. Tindakan ini akan membuat

peningkatan dimulai selama periode pemeliharaan berikutnya untuk klaster. Periode pemeliharaan menentukan periode saat operasi pemeliharaan dapat dimulai. Operasi yang berjalan lama mungkin tidak selesai selama periode pemeliharaan. Dengan demikian, Anda tidak perlu menentukan periode pemeliharaan yang lebih panjang bahkan jika Anda memperkirakan bahwa peningkatan mungkin akan memakan waktu lama.

Contoh berikut memutakhirkan cluster yang awalnya menjalankan Aurora SQL My versi 2.11.2. Dalam output `describe-db-engine-versions`, nilai `False` dan `True` merepresentasikan properti `IsMajorVersionUpgrade`. Dari versi 2.11.2, Anda dapat meningkatkan ke beberapa versi 2.* lainnya. Peningkatan tersebut tidak dianggap sebagai peningkatan versi mayor sehingga tidak memerlukan peningkatan di tempat. Peningkatan di tempat hanya tersedia untuk peningkatan ke versi 3.* yang ditampilkan dalam daftar.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget][0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text

5.7.mysql_aurora.2.11.3 False
5.7.mysql_aurora.2.11.4 False
5.7.mysql_aurora.2.11.5 False
5.7.mysql_aurora.2.11.6 False
5.7.mysql_aurora.2.12.0 False
5.7.mysql_aurora.2.12.1 False
5.7.mysql_aurora.2.12.2 False
5.7.mysql_aurora.2.12.3 False
8.0.mysql_aurora.3.04.0 True
8.0.mysql_aurora.3.04.1 True
8.0.mysql_aurora.3.04.2 True
8.0.mysql_aurora.3.04.3 True
8.0.mysql_aurora.3.05.2 True
8.0.mysql_aurora.3.06.0 True
8.0.mysql_aurora.3.06.1 True
8.0.mysql_aurora.3.07.1 True

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --no-apply-immediately --allow-major-
version-upgrade
```

```
...
```

Ketika sebuah klaster dibuat tanpa periode pemeliharaan yang ditentukan, Aurora akan memilih hari acak dalam seminggu. Dalam hal ini, perintah `modify-db-cluster` dikirimkan pada hari Senin. Dengan demikian, kita mengubah periode pemeliharaan menjadi Selasa pagi. Semua waktu diwakili dalam zona UTC waktu. Periode `tue:10:00-tue:10:30` sama dengan 02.00-02.30 waktu Pasifik. Perubahan periode pemeliharaan akan langsung berlaku.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

Contoh berikut menunjukkan cara mendapatkan laporan dari peristiwa yang dihasilkan oleh peningkatan. Argumen `--duration` merepresentasikan jumlah menit untuk mengambil informasi peristiwa. Argumen ini diperlukan, karena secara default `describe-events` hanya mengembalikan peristiwa dari jam terakhir.

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
                "creation"
```

```
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:01:45.428000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
```

```
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        }
    ]
```

```
}
```

Menemukan alasan Aurora Kegagalan peningkatan versi SQL utama saya

Dalam tutorial, upgrade dari Aurora My SQL versi 2 ke versi 3 berhasil. Tetapi jika pemutakhiran gagal, Anda pasti ingin tahu alasannya.

Anda dapat mulai dengan menggunakan `describe-events` AWS CLI perintah untuk melihat peristiwa cluster DB. Contoh ini menunjukkan peristiwa `mydbcluster` selama 10 jam terakhir.

```
aws rds describe-events \
    --source-type db-cluster \
    --source-identifier mydbcluster \
    --duration 600
```

Dalam hal ini, kami mengalami kegagalan precheck upgrade.

```
{
    "Events": [
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:22.846000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        },
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster is in a state that cannot be upgraded: Upgrade
 prechecks failed. For more details, see the
             upgrade-prechecks.log file. For more information on troubleshooting the
 cause of the upgrade failure, see
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.Upgrading.Troubleshooting.html",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:24.373000+00:00",
```

```
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        }
    ]
}
```

Untuk mendiagnosis penyebab pasti masalah, periksa log database untuk instance DB penulis. Ketika upgrade ke Aurora My SQL version 3 gagal, instance writer berisi file log dengan nama. `upgrade-prechecks.log` Contoh ini menunjukkan cara mendeteksi keberadaan log tersebut lalu mengunduhnya ke file lokal untuk diperiksa.

```
aws rds describe-db-log-files --db-instance-identifier mydbcluster-instance \
    --query '*[].[LogFileName]' --output text

error/mysql-error-running.log
error/mysql-error-running.log.2024-04-11.20
error/mysql-error-running.log.2024-04-11.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

aws rds download-db-log-file-portion --db-instance-identifier mydbcluster-instance \
    --log-file-name upgrade-prechecks.log \
    --starting-token 0 \
    --output text >upgrade_prechecks.log
```

`upgrade-prechecks.log`File dalam JSON format. Kami mengunduhnya menggunakan `--output text` opsi untuk menghindari pengkodean JSON output dalam JSON pembungkus lain. Untuk upgrade Aurora My SQL versi 3, log ini selalu menyertakan pesan informasi dan peringatan tertentu. Log ini hanya berisi pesan kesalahan jika peningkatan gagal. Jika peningkatan berhasil, file log tidak dibuat sama sekali.

Untuk meringkas semua kesalahan dan menampilkan bidang objek dan deskripsi terkait, Anda dapat menjalankan perintah `grep -A 2 '"level": "Error"'` pada isi `upgrade-prechecks.log` file. Tindakan ini akan menampilkan setiap baris kesalahan dan dua baris setelahnya. Baris ini berisi nama objek basis data yang sesuai dan panduan tentang cara memperbaiki masalahnya.

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'

"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
```

```
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling
 FULLTEXT index. Kindly recreate the table before upgrade."
```

Dalam contoh ini, Anda dapat menjalankan SQL perintah berikut pada tabel yang menyinggung untuk mencoba memperbaiki masalah, atau Anda dapat membuat ulang tabel tanpa indeks yang menggantung.

```
OPTIMIZE TABLE problematic_upgrade.dangling_fulltext_index;
```

Kemudian coba lagi upgrade.

Pemecahan masalah untuk Aurora Peningkatan di tempat saya SQL

Gunakan tips berikut untuk membantu memecahkan masalah dengan peningkatan Aurora My in-place. SQL Tips ini tidak berlaku untuk Aurora Serverless klaster DB.

| Alasan peningkatan di tempat dibatalkan atau lambat | Efek | Solusi untuk memungkinkan peningkatan di tempat selesai dalam periode pemeliharaan |
|---|---|---|
| Replika Lintas wilayah Aurora Terkait belum ditambal | Aurora membatalkan peningkatan. | Tingkatkan replika Aurora Cross-region dan coba lagi. |
| Klaster memiliki transaksi XA dalam status siap | Aurora membatalkan peningkatan. | Komit atau kembalikan semua transaksi XA yang disiapkan. |
| Cluster memproses pernyataan bahasa definisi data (DDL) apa pun | Aurora membatalkan peningkatan. | Pertimbangkan untuk menunggu dan melakukan peningkatan setelah semua DDL pernyataan selesai. |
| Klaster memiliki perubahan yang tidak dikomit untuk banyak baris | Peningkatan mungkin memerlukan waktu lama. | Proses peningkatan mengembalikan perubahan yang tidak dikomit. Indikator untuk kondisi ini adalah nilai `TRX_ROWS_MODIFIED` dalam tabel `INFORMATION_SCHEMA.INNODB_TRX` . |

| Alasan peningkatan di tempat dibatalkan atau lambat | Efek | Solusi untuk memungkinkan peningkatan di tempat selesai dalam periode pemeliharaan |
| --- | --- | --- |
| | | Pertimbangkan untuk melakukan peningkatan hanya setelah semua transaksi besar dilakukan atau dikembalikan. |
| Klaster memiliki jumlah catatan undo yang tinggi | Peningkatan mungkin memerlukan waktu lama. | Bahkan jika transaksi yang tidak dikomit tidak memengaruhi sejumlah besar baris, transaksi ini mungkin berkaitan dengan volume besar data. Misalnya, Anda mungkin memasukkan besarBLOBs. Aurora tidak akan secara otomatis mendeteksi atau menghasilkan peristiwa untuk jenis aktivitas transaksi ini. Indikator untuk kondisi ini adalah panjang daftar riwayat (HLL). Proses peningkatan mengembalikan perubahan yang tidak dikomit.<br><br>Anda dapat memeriksa HLL output dari `SHOW ENGINE INNODB STATUS` SQL perintah, atau langsung dengan menggunakan SQL query berikut:<br><br>`SELECT count FROM information_schema .innodb_metrics WHERE name = 'trx_rseg_history_len';`<br><br>Anda juga dapat memantau `RollbackSegmentHistoryListLength` metrik di Amazon CloudWatch.<br><br>Pertimbangkan untuk melakukan upgrade hanya setelah HLL lebih kecil. |

| Alasan peningkatan di tempat dibatalkan atau lambat | Efek | Solusi untuk memungkinkan peningkatan di tempat selesai dalam periode pemeliharaan |
| --- | --- | --- |
| Klaster sedang dalam proses melakukan komit transaksi log biner besar | Peningkatan mungkin memerlukan waktu lama. | Proses peningkatan menunggu sampai perubahan log biner diterapkan. Lebih banyak transaksi atau DDL pernyataan dapat dimulai selama periode ini, yang selanjutnya memperlambat proses peningkatan.<br><br>Jadwalkan proses peningkatan saat klaster tidak sibuk menghasilkan perubahan replikasi log biner. Aurora tidak akan secara otomatis mendeteksi atau membuat peristiwa untuk kondisi ini. |

| Alasan peningkatan di tempat dibatalkan atau lambat | Efek | Solusi untuk memungkinkan peningkatan di tempat selesai dalam periode pemeliharaan |
| --- | --- | --- |
| Inkonsistensi skema yang dihasilkan dari penghapusan atau kerusakan file | Aurora membatalkan peningkatan. | Ubah mesin penyimpanan default untuk tabel sementara dari My ISAM ke InnoDB. Lakukan langkah-langkah berikut ini:<br>1. Ubah parameter DB `default_tmp_storage_engine` menjadi `InnoDB`.<br>2. Boot ulang klaster DB.<br>3. Setelah boot ulang, konfirmasikan bahwa parameter DB `default_tmp_storage_engine` diatur ke `InnoDB`. Gunakan perintah berikut ini.<br>`show global variables like 'default_tmp_storage_engine';`<br>4. Pastikan untuk tidak membuat tabel sementara yang menggunakan mesin ISAM penyimpanan Saya. Kami menyarankan agar Anda menjeda beban kerja basis data apa pun dan tidak membuat koneksi basis data baru karena Anda akan segera melakukan peningkatan.<br>5. Coba lagi peningkatan di tempat. |

| Alasan peningkatan di tempat dibatalkan atau lambat | Efek | Solusi untuk memungkinkan peningkatan di tempat selesai dalam periode pemeliharaan |
|---|---|---|
| Pengguna master dihapus | Aurora membatalkan peningkatan. | **Important**<br>Jangan hapus pengguna master.<br><br>Namun, jika karena alasan tertentu Anda kebetulan menghapus pengguna master, kembalikan menggunakan SQL perintah berikut:<br><br>`CREATE USER 'master_username '@'% IDENTIFIED BY ' master_user_password ' REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK;`<br><br>`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO S3, INVOKE LAMBDA, INVOKE SAGEMAKER , INVOKE COMPREHEND ON *.* TO 'master_username '@'% WITH GRANT OPTION;` |

Untuk detail selengkapnya tentang masalah pemecahan masalah yang menyebabkan precheck upgrade gagal, lihat blog berikut:

- Amazon Aurora SQL Versi saya 2 (dengan kompatibilitas SQL 5.7 Saya) ke versi 3 (dengan kompatibilitas SQL 8.0 Saya) daftar periksa pemutakhiran, Bagian 1

- Amazon Aurora SQL Versi saya 2 (dengan kompatibilitas SQL 5.7 Saya) ke versi 3 (dengan kompatibilitas SQL 8.0 Saya) daftar periksa pemutakhiran, Bagian 2

Anda dapat menggunakan langkah-langkah berikut untuk melakukan pemeriksaan Anda sendiri untuk beberapa kondisi di tabel sebelumnya. Dengan demikian, Anda dapat menjadwalkan peningkatan pada saat Anda mengetahui basis data dalam keadaan yang memungkinkan peningkatan berhasil diselesaikan dengan cepat.

- Anda dapat memeriksa transaksi XA terbuka dengan menjalankan pernyataan `XA RECOVER`. Anda kemudian dapat meng-commit atau mengembalikan transaksi XA sebelum memulai peningkatan.
- Anda dapat memeriksa DDL pernyataan dengan mengeksekusi `SHOW PROCESSLIST` pernyataan dan mencari`CREATE`,,`DROP`, `ALTER``RENAME`, dan `TRUNCATE` pernyataan dalam output. Izinkan semua DDL pernyataan selesai sebelum memulai peningkatan.
- Anda dapat memeriksa jumlah total baris yang tidak dikomit dengan mengueri tabel `INFORMATION_SCHEMA.INNODB_TRX`. Tabel ini berisi satu baris untuk setiap transaksi. Kolom `TRX_ROWS_MODIFIED` berisi jumlah baris yang diubah atau dimasukkan oleh transaksi.
- Anda dapat memeriksa panjang daftar riwayat InnoDB dengan menjalankan pernyataan `SHOW ENGINE INNODB STATUS SQL` dan mencari `History list length` dalam output. Anda juga dapat memeriksa nilai secara langsung dengan menjalankan kueri berikut:

  ```
  SELECT count FROM information_schema.innodb_metrics WHERE name =
   'trx_rseg_history_len';
  ```

  Panjang daftar riwayat sesuai dengan jumlah informasi undo yang disimpan oleh database untuk menerapkan kontrol konkurensi multi-versi (). MVCC

Pembersihan pasca-upgrade untuk Aurora Versi saya 3 SQL

Setelah Anda selesai memutakhirkan klaster Aurora SQL My version 2 ke SQL Aurora My version 3, Anda dapat melakukan tindakan pembersihan lainnya:

- Buat versi baru yang SQL kompatibel dengan My 8.0 dari grup parameter kustom apa pun. Terapkan nilai parameter kustom yang diperlukan ke grup parameter baru.
- Perbarui CloudWatch alarm, skrip penyiapan, dan sebagainya untuk menggunakan nama baru untuk metrik apa pun yang namanya dipengaruhi oleh perubahan bahasa inklusif. Untuk daftar metrik tersebut, lihat Perubahan bahasa inklusif untuk Aurora Versi saya 3 SQL.

- Perbarui apa saja AWS CloudFormation template untuk menggunakan nama baru untuk parameter konfigurasi apa pun yang namanya dipengaruhi oleh perubahan bahasa inklusif. Untuk daftar parameter tersebut, lihat Perubahan bahasa inklusif untuk Aurora Versi saya 3 SQL.

Indeks spasial

Setelah memutakhirkan ke Aurora SQL My versi 3, periksa apakah Anda perlu menjatuhkan atau membuat ulang objek dan indeks yang terkait dengan indeks spasial. Sebelum My SQL 8.0, Aurora dapat mengoptimalkan kueri spasial menggunakan indeks yang tidak berisi pengenal sumber daya spasial (). SRID Aurora SQL Versi saya 3 hanya menggunakan indeks spasial yang berisi. SRIDs Selama peningkatan, Aurora secara otomatis menjatuhkan indeks spasial apa pun tanpa SRIDs dan mencetak pesan peringatan di log database. Jika Anda mengamati pesan peringatan tersebut, buat indeks spasial baru SRIDs setelah pemutakhiran. Untuk informasi selengkapnya tentang perubahan fungsi spasial dan tipe data di My SQL 8.0, lihat Perubahan pada SQL 8.0 Saya di Manual SQL Referensi Saya.

## Pembaruan dan perbaikan mesin database untuk Amazon Aurora MySQL

Anda dapat menemukan informasi berikut di Catatan rilis untuk Amazon Aurora Edisi yang kompatibel dengan MySQL:

- Pembaruan mesin database untuk Amazon Aurora MySQL versi 3
- Pembaruan mesin database untuk Amazon Aurora MySQL versi 2
- Pembaruan mesin database untuk Amazon Aurora MySQL versi 1 (Usang)
- Bug MySQL diperbaiki oleh pembaruan mesin database Aurora MySQL
- Kerentanan keamanan diperbaiki di Amazon Aurora MySQL

# Menggunakan Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL adalah mesin basis data relasional yang terkelola sepenuhnya, kompatibel dengan PostgreSQL, dan memenuhi standar ACID. Mesin basis data ini menggabungkan kecepatan, keandalan, dan pengelolaan Amazon Aurora dengan kesederhanaan dan efektivitas biaya basis data sumber terbuka. Aurora PostgreSQL adalah pengganti "drop-in" untuk PostgreSQL serta memudahkan dan menghemat biaya untuk menyiapkan, mengoperasikan, dan menskalakan deployment PostgreSQL baru dan yang sudah ada, sehingga Anda bebas untuk fokus pada bisnis dan aplikasi Anda. Untuk mempelajari selengkapnya tentang Aurora secara umum, lihat Apa itu Amazon Aurora?.

Selain manfaat Aurora, Aurora PostgreSQL menawarkan jalur migrasi praktis dari Amazon RDS ke Aurora, dengan alat migrasi yang hanya memerlukan beberapa klik untuk mengonversi aplikasi RDS for PostgreSQL Anda yang sudah ada menjadi aplikasi Aurora PostgreSQL. Tugas basis data rutin seperti penyediaan, patching, pencadangan, pemulihan, deteksi kegagalan, dan perbaikan juga mudah dikelola dengan Aurora PostgreSQL.

Aurora PostgreSQL dapat memenuhi berbagai standar industri. Misalnya, Anda dapat menggunakan basis data Aurora PostgreSQL untuk membuat aplikasi yang memenuhi standar HIPAA dan untuk menyimpan informasi terkait layanan kesehatan, termasuk informasi kesehatan yang dilindungi (PHI), berdasarkan Perjanjian Rekanan Bisnis (BAA) dengan AWS.

Aurora PostgreSQL memenuhi syarat FedRAMP tingkat TINGGI. Untuk detail tentang AWS dan upaya kepatuhan, lihat AWS layanan dalam cakupan berdasarkan program kepatuhan.

**Important**

Jika Anda mengalami masalah dengan cluster Aurora PostgreSQL DB Anda, agen AWS dukungan Anda mungkin memerlukan informasi lebih lanjut tentang kesehatan database Anda. Tujuannya adalah untuk memastikan bahwa AWS Support mendapatkan sebanyak mungkin informasi yang diperlukan dalam waktu sesingkat mungkin.
Anda dapat menggunakan PG Collector untuk membantu mengumpulkan informasi database yang berharga dalam file HTML terkonsolidasi. Untuk informasi lebih lanjut tentang PG Collector, cara menjalankannya, dan cara mengunduh laporan HTML, lihat PG Collector.
Setelah berhasil diselesaikan, dan kecuali dinyatakan lain, skrip mengembalikan output dalam format HTML yang dapat dibaca. Skrip ini dirancang untuk mengecualikan data atau detail keamanan apa pun dari HTML yang mungkin membahayakan bisnis Anda. Skrip ini

juga tidak melakukan modifikasi pada basis data Anda atau lingkungannya. Namun, jika Anda menemukan informasi apa pun dalam HTML yang tidak nyaman Anda bagikan, jangan ragu untuk menghapus informasi yang bermasalah sebelum mengunggah HTML. Jika HTML dapat diterima, unggah menggunakan bagian lampiran dalam detail kasus kasus dukungan Anda.

Topik


- Bekerja dengan Lingkungan Pratinjau Basis Data
- Keamanan dengan Amazon Aurora PostgreSQL
- Memperbarui aplikasi untuk terhubung ke klaster DB Aurora PostgreSQL menggunakan sertifikat SSL/TLS baru
- Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL
- Migrasi data ke Amazon Aurora dengan kompatibilitas Postgre SQL
- Mengoptimalkan kinerja kueri di Aurora PostgreSQL
- Bekerja dengan tabel yang tidak tercatat di Aurora PostgreSQL
- Bekerja dengan PostgreSQL autovacuum di Amazon RDS Aurora PostgreSQL
- Menggunakan Babelfish for Aurora PostgreSQL
- Kinerja dan penskalaan untuk Amazon Aurora PostgreSQL
- Menyetel dengan peristiwa tunggu di Aurora PostgreSQL
- Menyetel dengan wawasan proaktif Amazon Guru DevOps
- Praktik terbaik dengan Amazon Aurora PostgreSQL
- Replikasi dengan Amazon Aurora PostgreSQL
- Penerusan tulis lokal di Aurora PostgreSQL
- Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock
- Mengintegrasikan Amazon Aurora SQL Postgre dengan layanan lain AWS
- Memantau rencana eksekusi kueri dan memori puncak untuk Aurora Postgre SQL
- Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL
- Menangani ekstensi dan pembungkus data asing
- Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL
- Referensi Amazon Aurora PostgreSQL

- Pembaruan mesin database untuk Amazon Aurora Postgre SQL

# Bekerja dengan Lingkungan Pratinjau Basis Data

SQLKomunitas Postgre merilis versi utama baru SQL Postgre setiap tahun. Demikian pula, Amazon Aurora membuat versi SQL utama Postgre tersedia sebagai rilis Pratinjau. Hal ini memungkinkan Anda membuat klaster DB menggunakan versi Pratinjau dan menguji fiturnya di Lingkungan Pratinjau Basis Data.

Cluster Aurora Postgre SQL DB di Lingkungan Pratinjau Database secara fungsional mirip dengan cluster DB Aurora Postgre lainnya. SQL Namun, Anda tidak dapat menggunakan versi Pratinjau untuk produksi.

Perhatikan batasan penting berikut:

- Semua instans DB dan klaster DB dihapus 60 hari setelah Anda membuatnya, beserta semua cadangan dan snapshot.
- Anda hanya dapat membuat instans DB di cloud pribadi virtual (VPC) berdasarkan VPC layanan Amazon.
- Anda tidak dapat menyalin snapshot instans DB ke lingkungan produksi.

Opsi berikut didukung oleh Pratinjau.

- Anda dapat membuat instance DB menggunakan jenis instans r5, r6g, r6i, r7g, r7i, r8g, x2g, t3 dan t4g saja. Untuk informasi selengkapnya tentang kelas instans, lihat Kelas instans Amazon Aurora DB.
- Anda dapat menggunakan deployment AZ tunggal dan multi-AZ.
- Anda dapat menggunakan fungsi SQL dump dan load Postgre standar untuk mengekspor database dari atau mengimpor database ke Lingkungan Pratinjau Database.

## Jenis kelas instans DB yang didukung

Amazon Aurora Postgre SQL mendukung kelas instans DB berikut di wilayah pratinjau:

Kelas Memori yang Dioptimalkan

- db.r5 – kelas instans memori yang dioptimalkan

- db.r6g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton2 AWS
- db.r6i – kelas instans dengan memori yang dioptimalkan
- db.r7g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton3 AWS
- db.r7i - kelas instance yang dioptimalkan untuk memori
- db.r8g — kelas instans yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton4 AWS
- db.x2g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton2 AWS

> **Note**
>
> Untuk informasi selengkapnya tentang daftar kelas instans, lihat Jenis kelas instans DB.

Kelas yang dapat melonjak

- db.t3.medium
- db.t3.large
- db.t4g.medium
- db.t4g.large

## Fitur yang tidak didukung di Lingkungan Pratinjau

Fitur berikut ini tidak tersedia di Lingkungan Pratinjau:

- Aurora Serverless v1 dan v2
- Upgrade versi utama () MVU
- Tidak ada versi minor baru yang akan dirilis di wilayah pratinjau
- RDSuntuk replikasi masuk Postgre ke SQL Aurora Postgre SQL
- Penyebaran Amazon RDS Biru/Hijau
- Salinan snapshot lintas-wilayah

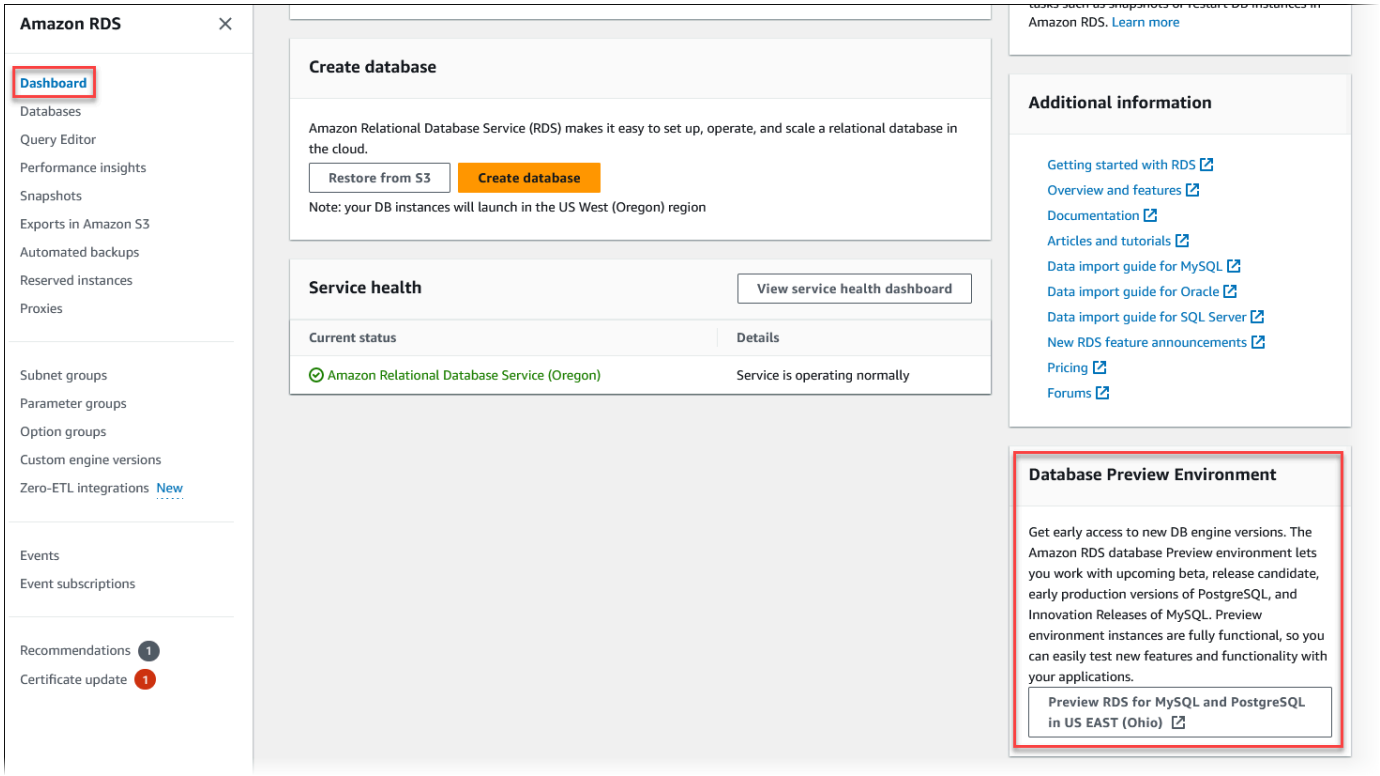
- Basis data global Aurora
- Aliran aktivitas database (DAS), RDS Proxy, dan AWS DMS
- Replika baca penskalaan otomatis
- AWS Batuan dasar
- RDSekspor
- Wawasan Performa
- Titik akhir kustom
- Salinan snapshot
- Nol- ETL
- Babelfish
- Semua ekstensi

## Membuat klaster DB baru di Preview Environment

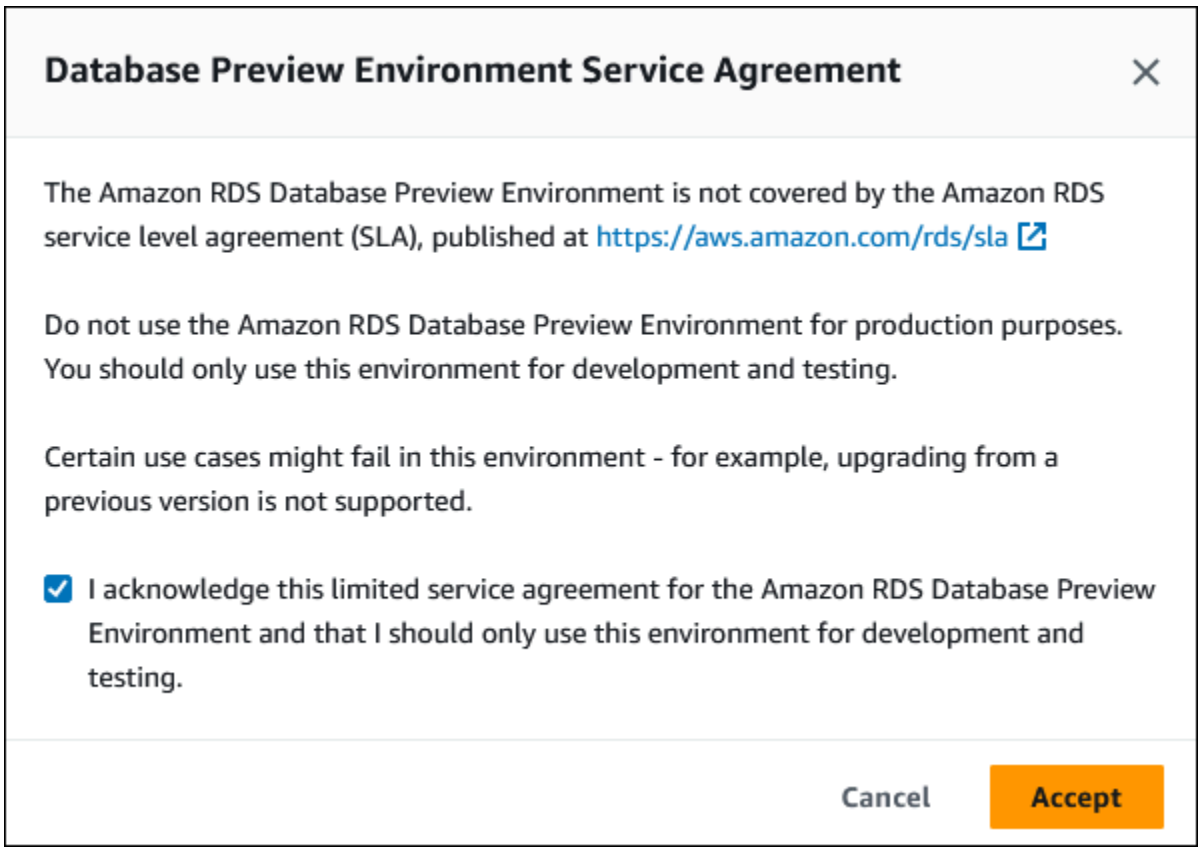
Gunakan prosedur berikut untuk membuat klaster DB di Preview Environment.

Untuk membuat klaster DB di Preview Environment

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Dasbor dari panel navigasi.
3. Di halaman Dasbor, temukan bagian Lingkungan Pratinjau Basis Data, seperti yang ditunjukkan pada gambar berikut.




Anda dapat langsung menuju Lingkungan pratinjau basis data. Sebelum dapat melanjutkan, Anda harus mengakui dan menerima batasan.




4. Untuk membuat cluster Aurora Postgre SQL DB, ikuti proses yang sama seperti untuk membuat cluster Aurora DB apa pun. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Untuk membuat instance di Lingkungan Pratinjau Database menggunakan Aurora API atau AWS CLI, gunakan endpoint berikut.

```
rds-preview.us-east-2.amazonaws.com
```

## Postgre SQL versi 17 di lingkungan Pratinjau Database

⚠ Ini adalah dokumentasi pratinjau untuk Aurora SQL Postgre versi 17. Dokumentasi dapat berubah.

Postgre SQL versi 17.0 sekarang tersedia di lingkungan Amazon RDS Database Preview. Postgre SQL versi 17 berisi beberapa perbaikan yang dijelaskan dalam dokumentasi SQL Postgre berikut:

- Postgre 17 Dirilis SQL

Untuk informasi tentang Lingkungan Pratinjau Basis Data, lihat Bekerja dengan Lingkungan Pratinjau Basis Data. Untuk mengakses Lingkungan Pratinjau dari konsol, pilih https://console.aws.amazon.com/rds-preview/.

# Keamanan dengan Amazon Aurora PostgreSQL

Untuk gambaran umum tentang keamanan Aurora, lihat. Keamanan dalam Amazon Aurora Anda dapat mengelola keamanan untuk Amazon Aurora PostgreSQL pada beberapa level yang berbeda:

- Untuk mengontrol siapa yang dapat melakukan tindakan pengelolaan Amazon RDS pada cluster DB PostgreSQL Aurora dan instans DB PostgreSQL, gunakan (IAM). AWS Identity and Access Management IAM menangani autentikasi identitas pengguna sebelum pengguna dapat mengakses layanan. IAM juga menangani otorisasi, yaitu, apakah pengguna diizinkan untuk melakukan apa yang mereka coba lakukan. Autentikasi basis data IAM adalah metode autentikasi tambahan yang dapat dipilih saat Anda membuat klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.

  Jika Anda menggunakan IAM dengan cluster Aurora PostgreSQL DB Anda, masuk AWS Management Console ke dengan kredenal IAM Anda terlebih dahulu, sebelum membuka konsol Amazon RDS di. https://console.aws.amazon.com/rds/
- Pastikan untuk membuat klaster DB Aurora di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Untuk mengontrol perangkat dan EC2 instans Amazon mana yang dapat membuka koneksi ke titik akhir dan port instans DB untuk cluster Aurora DB di VPC, gunakan grup keamanan VPC. Anda dapat membuat titik akhir dan koneksi port ini menggunakan Lapisan Soket Aman (SSL). Selain itu, aturan firewall di perusahaan Anda dapat mengendalikan perangkat yang berjalan di perusahaan Anda untuk dapat membuka koneksi ke instans DB. Untuk informasi lebih lanjut tentang VPCs, lihatAmazon VPC dan Aurora.

  Penghunian VPC yang didukung tergantung pada kelas instans DB yang digunakan oleh klaster DB Aurora PostgreSQL Anda. Dengan penghunian VPC `default`, klaster DB berjalan di perangkat keras bersama. Dengan penghunian VPC `dedicated`, klaster DB berjalan di instans perangkat keras khusus. Kelas instans DB performa yang dapat melonjak hanya mendukung penghunian VPC default. Kelas instans DB performa yang dapat melonjak mencakup kelas instans DB db.t3 dan db.t4g. Semua kelas instans DB Aurora PostgreSQL lainnya mendukung penghunian VPC default dan khusus.

Untuk informasi selengkapnya tentang kelas instans, lihat Kelas instans Amazon Aurora DB. Untuk informasi selengkapnya tentang penghunian VPC `default` dan `dedicated`, lihat Instans khusus dalam Panduan Pengguna Amazon Elastic Compute Cloud.

- Guna memberikan izin ke basis data PostgreSQL yang berjalan di klaster DB Amazon Aurora, Anda dapat menggunakan pendekatan umum yang sama seperti instans PostgreSQL yang berdiri sendiri. Perintah seperti `CREATE ROLE`, `ALTER ROLE`, `GRANT`, dan `REVOKE` bekerja sama seperti dalam basis data on-premise, seperti mengubah langsung basis data, skema, dan tabel.

  PostgreSQL mengelola hak istimewa dengan menggunakan peran. Peran `rds_superuser` adalah peran yang paling istimewa di klaster DB Aurora PostgreSQL. Peran ini dibuat secara otomatis, dan diberikan kepada pengguna yang membuat klaster DB (akun pengguna master, `postgres` secara default). Untuk mempelajari selengkapnya, lihat Memahami peran dan izin PostgreSQL.

Semua versi Aurora PostgreSQL yang tersedia, termasuk versi 10, 11, 12, 13, 14, dan rilis yang lebih tinggi mendukung Salted Challenge Response Authentication Mechanism (SCRAM) untuk kata sandi sebagai alternatif untuk message digest (). MD5 Kami menyarankan Anda menggunakan SCRAM karena lebih aman daripada MD5. Untuk informasi selengkapnya, termasuk cara memigrasikan kata sandi pengguna database dari SCRAM MD5 ke SCRAM, lihat. Menggunakan SCRAM untuk enkripsi kata sandi PostgreSQL

## Memahami peran dan izin PostgreSQL

Bila Anda membuat Aurora PostgreSQL DB cluster DB menggunakan, akun administrator dibuat pada saat yang sama. AWS Management Console Secara default, akun tersebut akan dinamakan `postgres`, seperti yang ditunjukkan pada gambar berikut:

Anda dapat memilih nama lain daripada menerima nama default (`postgres`). Jika ya, nama yang Anda pilih harus dimulai dengan huruf dan terdiri atas 1 hingga 16 karakter alfanumerik. Untuk menyederhanakan, kami merujuk ke akun pengguna utama ini dengan nilai default-nya (`postgres`) di seluruh panduan ini.

Jika Anda menggunakan `create-db-cluster` AWS CLI bukan AWS Management Console, Anda membuat nama pengguna dengan meneruskannya dengan `master-username` parameter. Untuk informasi selengkapnya, lihat Langkah 2: Membuat klaster DB Aurora PostgreSQL.

Baik Anda menggunakan AWS Management Console, API AWS CLI, atau Amazon RDS, dan apakah Anda menggunakan `postgres` nama default atau memilih nama yang berbeda, akun pengguna database pertama ini adalah anggota `rds_superuser` grup dan memiliki `rds_superuser` hak istimewa.

Topik

- Memahami peran rds_superuser
- Mengontrol akses pengguna ke basis data PostgreSQL
- Menyerahkan dan mengendalikan manajemen kata sandi pengguna
- Menggunakan SCRAM untuk enkripsi kata sandi PostgreSQL

## Memahami peran rds_superuser

Dalam PostgreSQL, peran dapat menentukan pengguna, grup, atau sekumpulan izin tertentu yang diberikan kepada grup atau pengguna untuk berbagai objek dalam basis data. Perintah PostgreSQL untuk `CREATE USER` dan `CREATE GROUP` telah digantikan oleh `CREATE ROLE` yang lebih umum dengan properti khusus untuk membedakan pengguna basis data. Pengguna basis data dapat dianggap sebagai peran dengan hak akses LOGIN.

> **Note**
>
> Perintah `CREATE USER` dan `CREATE GROUP` masih dapat digunakan. Untuk informasi selengkapnya, lihat Database Roles dalam dokumentasi PostgreSQL.

Pengguna `postgres` adalah pengguna basis data dengan hak akses tertinggi di klaster DB Aurora PostgreSQL . Pengguna tersebut memiliki karakteristik yang ditentukan oleh pernyataan `CREATE ROLE` berikut.

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

PropertiNOSUPERUSER, NOREPLICATION, INHERIT, dan VALID UNTIL 'infinity' merupakan opsi default untuk CREATE ROLE, kecuali ditentukan lain.

Secara default, postgres memiliki hak akses yang diberikan kepada peran rds_superuser, serta izin untuk membuat peran dan basis data. Peran rds_superuser mengizinkan pengguna postgres untuk melakukan tindakan berikut:

- Menambahkan ekstensi yang tersedia untuk digunakan dengan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menangani ekstensi dan pembungkus data asing.
- Buat peran dan berikan hak akses kepada pengguna. Untuk informasi selengkapnya, lihat CREATE ROLE dan GRANT dalam dokumentasi PostgreSQL.
- Membuat basis data. Untuk informasi selengkapnya, lihat CREATE DATABASE dalam dokumentasi PostgreSQL.
- Berikan hak akses rds_superuser untuk peran pengguna yang tidak memiliki hak akses ini, dan cabut hak akses sesuai kebutuhan. Sebaiknya berikan peran ini hanya untuk pengguna yang menjalankan tugas superuser. Dengan kata lain, Anda dapat memberikan peran ini kepada administrator database (DBAs) atau administrator sistem.
- Berikan (dan cabut) peran rds_replication kepada pengguna basis data yang tidak memiliki peran rds_superuser.
- Berikan (dan cabut) peran rds_password kepada pengguna basis data yang tidak memiliki peran rds_superuser.
- Dapatkan informasi status tentang semua koneksi basis data dengan menggunakan tampilan pg_stat_activity. Bila diperlukan, rds_superuser dapat menghentikan semua koneksi menggunakan pg_terminate_backend atau pg_cancel_backend.

Dalam pernyataan CREATE ROLE postgres..., Anda dapat melihat bahwa peran pengguna postgres secara khusus melarang izin superuser PostgreSQL. Aurora PostgreSQL adalah layanan terkelola, jadi Anda tidak dapat mengakses OS host, dan terhubung menggunakan akun superuser PostgreSQL. Banyak tugas yang memerlukan akses superuser pada PostgreSQL mandiri dikelola secara otomatis oleh Aurora.

Untuk informasi selengkapnya tentang pemberian hak akses, lihat GRANT dalam dokumentasi PostgreSQL.

Peran `rds_superuser` adalah satu dari beberapa peran yang telah ditentukan dalam klaster DB Aurora PostgreSQL.

> **Note**
>
> Dalam PostgreSQL 13 dan rilis sebelumnya, peran yang telah ditentukan dikenal sebagai peran default.

Dalam daftar berikut, Anda akan menemukan beberapa peran standar lainnya yang dibuat secara otomatis untuk klaster DB Aurora PostgreSQL yang baru. Peran yang telah ditentukan dan hak aksesnya tidak dapat diubah. Anda tidak dapat menghapus, mengganti nama, atau memodifikasi hak akses untuk peran yang telah ditentukan ini. Mencoba untuk melakukannya akan mengakibatkan kesalahan.

- rds_password — Peran yang dapat mengubah kata sandi dan mengatur batasan kata sandi untuk pengguna basis data. `rds_superuser`Peran diberikan dengan peran ini secara default, dan dapat memberikan peran tersebut kepada pengguna database. Untuk informasi selengkapnya, lihat Mengontrol akses pengguna ke basis data PostgreSQL.
  - Untuk RDS untuk PostgreSQL versi yang lebih tua dari 14`rds_password`, peran dapat mengubah kata sandi dan mengatur batasan kata sandi untuk pengguna database dan pengguna dengan peran. `rds_superuser` Dari RDS untuk PostgreSQL versi 14 dan yang lebih baru`rds_password`, peran dapat mengubah kata sandi dan mengatur batasan kata sandi hanya untuk pengguna database. Hanya pengguna dengan `rds_superuser` peran yang dapat melakukan tindakan ini pada pengguna lain yang memiliki `rds_superuser` peran.
- rdsadmin — Peran yang dibuat untuk menangani banyak tugas manajemen yang dilakukan oleh administrator dengan hak akses `superuser` pada basis data PostgreSQL mandiri. Peran ini digunakan secara internal oleh Aurora PostgreSQL untuk banyak tugas manajemen.

Melihat peran dan hak istimewa mereka

Anda dapat melihat peran yang telah ditentukan dan hak istimewanya di RDS untuk instans PostgreSQL DB menggunakan perintah yang berbeda tergantung pada versi PostgreSQL Anda.

Untuk PostgreSQL 15 dan versi sebelumnya

Connect ke RDS Anda untuk PostgreSQL DB instance dan gunakan perintah di psql: `\du`

```
postgres=> \du
                                                              List of roles
    Role name    |                         Attributes                          |
                 Member of
-----------------+-----------------------------------------------------------------
+-------------------------------------------------
 postgres        | Create role, Create DB                                      +|
 {rds_superuser}
                 | Password valid until infinity                               |
 rds_ad          | Cannot login                                                | {}
 rds_iam         | Cannot login                                                | {}
 rds_password    | Cannot login                                                | {}
 rds_replication | Cannot login                                                | {}
 rds_superuser   | Cannot login                                                |
 {pg_monitor,pg_signal_backend,rds_password,rds_replication}
 rdsadmin        | Superuser, Create role, Create DB, Replication, Bypass RLS+| {}
                 | Password valid until infinity                               |
```

Untuk PostgreSQL 16 dan versi yang lebih baru

Dimulai dengan PostgreSQL 16, gunakan perintah untuk melihat informasi `\drg+` keanggotaan peran rinci:

```
postgres=> \drg+
                            List of role grants
   Role name   |          Member of          |       Options       | Grantor
---------------+-----------------------------+---------------------+----------
 postgres      | rds_superuser               | INHERIT, SET        | rdsadmin
 rds_superuser | pg_checkpoint               | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_monitor                  | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_signal_backend           | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_use_reserved_connections | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_password                | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_replication             | ADMIN, INHERIT, SET | rdsadmin
```

Untuk memeriksa keanggotaan peran terlepas dari versi PostgreSQL, Anda dapat menggunakan kueri SQL berikut:

```
SELECT m.rolname AS "Role name", r.rolname AS "Member of"
FROM pg_catalog.pg_roles m
JOIN pg_catalog.pg_auth_members pam ON (pam.member = m.oid)
LEFT JOIN pg_catalog.pg_roles r ON (pam.roleid = r.oid)
```

```
LEFT JOIN pg_catalog.pg_roles g ON (pam.grantor = g.oid)
WHERE m.rolname !~ '^pg_'
ORDER BY 1, 2;
```

Dalam output, Anda dapat melihat bahwa `rds_superuser` itu bukan peran pengguna basis data (tidak dapat masuk), tetapi memiliki hak akses dari banyak peran lainnya. Anda juga dapat melihat bahwa pengguna basis data `postgres` adalah anggota peran `rds_superuser`. Seperti disebutkan sebelumnya, `postgres` adalah nilai default di halaman Buat basis data konsol Amazon RDS. Jika Anda memilih nama lain, nama itu akan ditampilkan dalam daftar peran.

> **Note**
>
> Aurora PostgreSQL versi 15.2 dan 14.7 memperkenalkan perilaku restriktif peran `rds_superuser`. Pengguna Aurora PostgreSQL harus diberi hak akses `CONNECT` pada basis data terkait agar dapat menghubungkan, meskipun pengguna tersebut diberi peran `rds_superuser`. Sebelum Aurora PostgreSQL versi 14.7 dan 15.2, pengguna dapat terhubung dengan basis data dan tabel sistem mana pun jika diberi peran `rds_superuser`. Perilaku restriktif ini sejalan dengan komitmen Amazon Aurora AWS dan Amazon Aurora untuk peningkatan keamanan yang berkelanjutan.
> Perbarui logika masing-masing dalam aplikasi Anda jika peningkatan di atas berdampak.

## Mengontrol akses pengguna ke basis data PostgreSQL

Basis data baru di PostgreSQL selalu dibuat dengan serangkaian hak akses default dalam skema `public` basis data yang memungkinkan semua pengguna dan peran basis data untuk membuat objek. Hak akses ini memungkinkan pengguna basis data untuk terhubung ke basis data, misalnya, dan membuat tabel sementara saat terhubung.

Agar dapat mengontrol dengan lebih baik akses pengguna ke instans basis data yang Anda buat di simpul primer klaster DB Aurora PostgreSQL , sebaiknya Anda mencabut hak akses `public` default. Setelah melakukannya, Anda kemudian memberikan hak akses khusus untuk pengguna basis data secara lebih rinci, seperti yang diperlihatkan dalam prosedur berikut ini.

Untuk menyiapkan peran dan hak akses untuk instans basis data baru

Misalnya, Anda sedang menyiapkan basis data pada klaster DB Aurora PostgreSQL yang baru dibuat untuk digunakan oleh beberapa peneliti, yang semuanya membutuhkan akses baca-tulis ke basis data.

1. Gunakan `psql` (atau pgAdmin) untuk menghubungkan ke instans DB primer di klaster DB Aurora PostgreSQL:

```
psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

   Saat diminta, masukkan kata sandi Anda. Klien `psql` menghubungkan dan menampilkan basis data koneksi administratif default, `postgres=>`, sebagai perintah.

2. Untuk mencegah pengguna basis data membuat objek dalam skema `public`, lakukan tindakan berikut:

```
postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

3. Selanjutnya, buat instans basis data baru:

```
postgres=> CREATE DATABASE lab_db;
CREATE DATABASE
```

4. Cabut semua hak akses dari skema `PUBLIC` pada basis data baru ini.

```
postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
REVOKE
```

5. Buat peran untuk pengguna basis data.

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. Berikan pengguna basis data yang memiliki peran ini kemampuan untuk terhubung ke basis data.

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. Berikan semua hak akses pada basis data ini kepada semua pengguna dengan peran `lab_tech`.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. Buat pengguna basis data, sebagai berikut:

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. Beri hak akses terkait peran lab_tech kepada dua pengguna ini:

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

Pada titik ini, `lab_user1` dan `lab_user2` dapat terhubung ke basis data `lab_db`. Contoh ini tidak mengikuti praktik terbaik untuk penggunaan perusahaan, yang mungkin termasuk membuat beberapa instans basis data, skema yang berbeda, dan pemberian izin terbatas. Untuk informasi yang lebih lengkap dan skenario tambahan lainnya, lihat Mengelola Pengguna dan Peran PostgreSQL.

Untuk informasi selengkapnya tentang hak akses di PostgreSQL, lihat perintah GRANT dalam dokumentasi PostgreSQL.

### Menyerahkan dan mengendalikan manajemen kata sandi pengguna

Sebagai DBA, Anda mungkin perlu menyerahkan manajemen kata sandi pengguna. Atau, Anda juga dapat mencegah pengguna basis data mengubah kata sandi mereka atau mengonfigurasi ulang batasan kata sandi, seperti masa pakai kata sandi. Untuk memastikan bahwa hanya pengguna basis data yang Anda pilih yang dapat mengubah pengaturan kata sandi, Anda dapat mengaktifkan fitur manajemen kata sandi terbatas. Bila mengaktifkan fitur ini, hanya pengguna basis data yang telah diberikan peran `rds_password` yang dapat mengelola kata sandi.

> **Note**
>
> Untuk menggunakan manajemen kata sandi terbatas, klaster DB Aurora PostgreSQL harus menjalankan Amazon Aurora PostgreSQL 10.6 atau versi lebih tinggi.

Secara default, fitur ini dalam keadaan `off`, seperti yang ditunjukkan berikut:

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

Untuk mengaktifkan fitur ini, Anda harus menggunakan grup parameter kustom dan mengubah pengaturan `rds.restrict_password_commands` ke 1. Pastikan untuk melakukan booting ulang instans DB primer Aurora PostgreSQL agar pengaturan dapat berlaku.

Dalam keadaan fitur ini aktif, hak akses `rds_password` diperlukan untuk perintah SQL berikut:

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

Mengganti nama role (`ALTER ROLE myrole RENAME TO newname`) juga dibatasi jika password menggunakan algoritma MD5 hashing.

Jika fitur ini dalam keadaan aktif, mencoba salah satu perintah SQL ini tanpa izin peran `rds_password` akan menghasilkan kesalahan berikut:

```
ERROR: must be a member of rds_password to alter passwords
```

Sebaiknya Anda memberikan `rds_password` hanya untuk beberapa peran yang Anda gunakan semata untuk manajemen kata sandi. Jika memberikan hak akses `rds_password` kepada pengguna basis data yang tidak memiliki hak akses `rds_superuser`, Anda juga harus memberi mereka atribut `CREATEROLE`.

Pastikan Anda memverifikasi persyaratan kata sandi seperti kedaluwarsa dan kompleksitas yang diperlukan di sisi klien. Jika Anda menggunakan utilitas sisi klien Anda sendiri untuk perubahan terkait kata sandi, utilitas tersebut harus menjadi anggota `rds_password` dan memiliki hak akses `CREATE ROLE`.

## Menggunakan SCRAM untuk enkripsi kata sandi PostgreSQL

The Salted Challenge Response Authentication Mechanism (SCRAM) adalah alternatif dari algoritma message digest (MD5) default PostgreSQL untuk mengenkripsi kata sandi. Mekanisme otentikasi SCRAM dianggap lebih aman daripada. MD5 Untuk mempelajari selengkapnya tentang dua pendekatan berbeda untuk mengamankan kata sandi ini, lihat Password Authentication dalam dokumentasi PostgreSQL.

Kami menyarankan Anda menggunakan SCRAM daripada MD5 sebagai skema enkripsi kata sandi untuk cluster Aurora PostgreSQL DB Anda. SCRAM didukung di Aurora PostgreSQL versi 10 dan semua versi mayor dan minor yang lebih tinggi. Ini adalah mekanisme respons tantangan kriptografi yang menggunakan algoritma scram-sha-256 untuk autentikasi dan enkripsi kata sandi.

Anda mungkin perlu memperbarui pustaka untuk aplikasi klien Anda agar dapat mendukung SCRAM. Misalnya, versi JDBC sebelum 42.2.0 tidak mendukung SCRAM. Untuk informasi selengkapnya, lihat PostgreSQL JDBC Driver dalam dokumentasi Driver JDBC PostgreSQL. Untuk daftar driver PostgreSQL dan dukungan SCRAM lainnya, lihat List of drivers dalam dokumentasi PostgreSQL.

Aurora PostgreSQL versi 14 dan yang lebih tinggi mendukung scram-sha-256 untuk enkripsi kata sandi untuk klaster DB baru secara default. Untuk versi ini, grup parameter cluster DB default (`default.aurora-postgresql14`) memiliki `password_encryption` nilai yang disetel ke scram-sha-256. SCRAM tidak didukung untuk Aurora Serverless v1.

Mengatur klaster DB Aurora PostgreSQL agar meminta SCRAM

Untuk Aurora PostgreSQL 14.3 dan versi yang lebih tinggi, Anda dapat meminta klaster DB Aurora PostgreSQL untuk hanya menerima kata sandi yang menggunakan algoritma scram-sha-256.

> **Important**
>
> Untuk Proksi RDS yang ada dengan basis data PostgreSQL, jika Anda mengubah autentikasi basis data untuk hanya menggunakan `SCRAM`, proksi akan menjadi tidak tersedia selama maksimal 60 detik. Untuk menghindari masalah ini, lakukan salah satu tindakan berikut:
>
> - Pastikan basis data mengizinkan autentikasi `SCRAM` dan `MD5`.
> - Untuk hanya menggunakan autentikasi `SCRAM`, buat proksi baru, migrasikan lalu lintas aplikasi Anda ke proksi baru, lalu hapus proksi yang sebelumnya terkait dengan basis data.

Sebelum melakukan perubahan pada sistem, pastikan Anda telah memahami proses lengkapnya, sebagai berikut:

- Dapatkan informasi semua peran dan enkripsi kata sandi untuk semua pengguna basis data.
- Periksa kembali pengaturan parameter klaster DB Aurora PostgreSQL untuk parameter yang mengontrol enkripsi kata sandi.
- Jika klaster DB Aurora PostgreSQL menggunakan grup parameter default, maka Anda harus membuat grup parameter klaster DB , lalu menerapkannya ke klaster DB Aurora PostgreSQL agar Anda dapat memodifikasi parameter saat diperlukan. Jika klaster DB Aurora PostgreSQL menggunakan grup parameter kustom, Anda dapat memodifikasi parameter yang diperlukan nanti selama prosesnya sesuai kebutuhan.
- Ubah parameter `password_encryption` ke `scram-sha-256`.
- Beri tahu semua pengguna basis data bahwa mereka perlu memperbarui kata sandi. Lakukan hal yang sama untuk akun `postgres` Anda. Kata sandi baru telah dienkripsi dan disimpan menggunakan algoritma scram-sha-256.
- Verifikasikan bahwa semua kata sandi telah dienkripsi menggunakan jenis enkripsi tersebut.
- Jika semua kata sandi menggunakan scram-sha-256, Anda dapat mengubah parameter `rds.accepted_password_auth_method` dari `md5+scram` ke `scram-sha-256`.

> **Warning**
>
> Setelah Anda mengubah `rds.accepted_password_auth_method` ke scram-sha-256 saja, setiap pengguna (peran) dengan kata sandi terenkripsi `md5` tidak dapat terhubung.

Penyiapan meminta SCRAM untuk klaster DB Aurora PostgreSQL

Sebelum membuat perubahan apa pun pada klaster DB Aurora PostgreSQL periksa semua akun pengguna basis data yang ada. Periksa juga jenis enkripsi yang digunakan untuk kata sandi. Anda dapat melakukan semua tugas ini menggunakan ekstensi `rds_tools`. Untuk melihat versi PostgreSQL yang `rds_tools` didukung, lihat Versi ekstensi untuk Amazon RDS for PostgreSQL.

Untuk mendapatkan daftar pengguna (peran) basis data dan metode enkripsi kata sandi

1. Gunakan `psql` untuk terhubung ke instans primer klaster DB Aurora PostgreSQL , sebagaimana yang ditunjukkan di bawah ini.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. Instal ekstensi `rds_tools`.

```
postgres=> CREATE EXTENSION rds_tools;
CREATE EXTENSION
```

3. Dapatkan daftar peran dan enkripsi.

```
postgres=> SELECT * FROM
      rds_tools.role_password_encryption_type();
```

Anda akan melihat output yang mirip dengan berikut ini.

```
       rolname        | encryption_type
----------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
 postgres             | md5
(8 rows)
```

Membuat grup parameter klaster DB

> **Note**
>
> Jika klaster DB Aurora PostgreSQL sudah menggunakan grup parameter kustom, Anda tidak perlu membuat grup parameter yang baru.

Untuk ringkasan grup parameter Aurora, lihat Membuat grup parameter cluster DB di Amazon Aurora.

Jenis enkripsi kata sandi yang digunakan untuk kata sandi yang diatur dalam satu parameter, `password_encryption`. Enkripsi yang diizinkan oleh klaster DB Aurora PostgreSQL diatur dalam

parameter lain, `rds.accepted_password_auth_method`. Mengubah salah satu dari nilai default ini mengharuskan Anda membuat grup parameter klaster DB , lalu menerapkannya ke klaster.

Anda juga dapat menggunakan AWS Management Console atau RDS API untuk membuat grup parameter cluster DB kustom grup parameter . Untuk informasi lebih lanjut, lihat Membuat grup parameter cluster DB di Amazon Aurora.

Anda kini dapat mengaitkan grup parameter kustom dengan instans DB.

Untuk membuat grup parameter klaster DB

1. Gunakan perintah CLI `create-db-cluster-parameter-group` untuk membuat grup parameter kustom untuk klaster. Contoh berikut menggunakan `aurora-postgresql13` sebagai sumber untuk grup parameter kustom ini.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
     --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster parameter group for SCRAM'
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
     --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster parameter group for SCRAM"
   ```

   Anda kini dapat mengaitkan grup parameter kustom dengan klaster.

2. Gunakan perintah CLI `modify-db-cluster` untuk menerapkan grup parameter kustom ini ke klaster DB Aurora PostgreSQL Anda.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

   Untuk Windows:

```
aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
        --db-cluster-parameter-group-name "docs-lab-scram-passwords
```

Untuk menyinkronkan ulang klaster DB Aurora PostgreSQL Anda dengan grup parameter klaster DB kustom, boot ulang instans primer dan semua instans lain klaster.

Mengonfigurasi enkripsi kata sandi agar menggunakan SCRAM

Mekanisme enkripsi kata sandi yang digunakan oleh klaster DB Aurora PostgreSQL diatur dalam grup parameter klaster DB dalam parameter `password_encryption`. Nilai yang diizinkan tidak ditentukan, `md5`, atau `scram-sha-256`. Nilai default bergantung pada versi Aurora PostgreSQL , sebagai berikut:

- Aurora PostgreSQL 14 – Default adalah `scram-sha-256`
- Aurora PostgreSQL 13 – Default adalah `md5`

Dengan grup parameter klaster DB yang terhubung ke klaster DB Aurora PostgreSQL, Anda dapat memodifikasi nilai untuk parameter enkripsi kata sandi.

| Name | Values | Allowed values | Modifiable | Source | Apply type |
|---|---|---|---|---|---|
| password_encryption | scram-sha-256 | md5, scram-sha-256 | true | system | dynamic |
| rds.accepted_password_auth_method | md5+scram | md5+scram, scram | true | system | dynamic |

Untuk mengubah pengaturan enkripsi kata sandi menjadi scram-sha-256

- Ubah nilai enkripsi kata sandi menjadi scram-sha-256, sebagaimana ditunjukkan berikut ini. Perubahan dapat langsung diterapkan karena parameternya dinamis, jadi mulai ulang tidak diperlukan agar perubahan diterapkan.

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
```

```
  'docs-lab-scram-passwords' --parameters
 'ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate'
```

Untuk Windows:

```
aws rds modify-db-parameter-group --db-parameter-group-name ^
  "docs-lab-scram-passwords" --parameters
 "ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate"
```

Memigrasikan kata sandi untuk peran pengguna ke SCRAM

Anda dapat memigrasikan kata sandi untuk peran pengguna ke SCRAM sebagaimana dijelaskan berikut.

Untuk memigrasikan kata sandi pengguna (peran) basis data dari MD5 ke SCRAM

1. Masuk sebagai pengguna administrator (nama pengguna default, `postgres`) sebagaimana ditunjukkan berikut.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. Periksa pengaturan parameter `password_encryption` pada instans DB RDS for PostgreSQL menggunakan perintah berikut.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. Ubah nilai parameter ini menjadi scram-sha-256. Ini adalah parameter dinamis, jadi Anda tidak perlu mem-boot ulang instans setelah melakukan perubahan ini. Periksa kembali nilainya untuk memastikan bahwa sekarang diatur ke`scram-sha-256`, sebagai berikut.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
```

```
scram-sha-256
(1 row)
```

4. Beri tahu semua pengguna basis data untuk mengubah kata sandi mereka. Pastikan juga untuk mengubah kata sandi Anda sendiri untuk akun `postgres` (pengguna basis data dengan hak akses `rds_superuser`).

```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

5. Ulangi proses tersebut untuk semua basis data di klaster DB Aurora PostgreSQL Anda.

Mengubah parameter untuk mengharuskan SCRAM

Ini adalah langkah terakhir dalam proses. Setelah Anda membuat perubahan dalam prosedur berikut, setiap akun pengguna (peran) yang masih menggunakan enkripsi `md5` untuk kata sandi tidak dapat masuk ke klaster DB PostgreSQL Aurora.

`rds.accepted_password_auth_method` menentukan metode enkripsi yang klaster DB Aurora PostgreSQL DB menerima untuk kata sandi pengguna selama proses masuk. Nilai default-nya adalah `md5+scram`, yang berarti bahwa salah satu metode diterima. Pada gambar berikut, Anda dapat menemukan pengaturan default untuk parameter ini.

| Name | Values | Allowed values | Modifiable | Source | Apply type |
|---|---|---|---|---|---|
| password_encryption | scram-sha-256 | md5, scram-sha-256 | true | system | dynamic |
| rds.accepted_password_auth_method | md5+scram | md5+scram, scram | true | system | dynamic |

Nilai yang diizinkan untuk parameter ini adalah `md5+scram` atau `scram` saja. Mengubah nilai parameter ini ke `scram` akan menjadikannya sebagai persyaratan.

Untuk mengubah nilai parameter agar mengharuskan autentikasi SCRAM untuk kata sandi

1. Verifikasikan bahwa semua kata sandi pengguna basis data untuk semua basis data di klaster DB Aurora PostgreSQL menggunakan `scram-sha-256` untuk enkripsi kata sandi. Untuk melakukannya, buat kueri `rds_tools` untuk peran (pengguna) dan jenis enkripsi, sebagai berikut.

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname        | encryption_type
  ---------------------+-----------------
  pg_monitor           |
  pg_read_all_settings |
  pg_read_all_stats    |
  pg_stat_scan_tables  |
  pg_signal_backend    |
  lab_tester           | scram-sha-256
  user_465             | scram-sha-256
  postgres             | scram-sha-256
  ( rows)
```

2. Ulangi kueri di semua instans DB dalam klaster DB Aurora PostgreSQL Anda.

   Jika semua kata sandi menggunakan scram-sha-256, Anda dapat melanjutkan.

3. Ubah nilai autentikasi kata sandi yang diterima menjadi scram-sha-256, sebagai berikut.

   Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-
lab-scram-passwords' \
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

   Untuk Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

## Mengamankan data Aurora PostgreSQL dengan SSL/TLS

Amazon RDS mendukung enkripsi Lapisan Soket Aman (SSL) dan Keamanan Lapisan Pengangkutan (TLS) untuk klaster DB Aurora PostgreSQL. MenggunakanSSL/TLS, you can encrypt a connection between your applications and your Aurora PostgreSQL DB clusters. You can also force all connections to your Aurora PostgreSQL DB cluster to use SSL/TLS. Amazon Aurora PostgreSQL mendukung Keamanan Lapisan Pengangkutan (TLS) versi 1.1 dan 1.2. Kami merekomendasikan

menggunakan TLS 1.2 untuk koneksi terenkripsi. Kami telah menambahkan dukungan untuk TLSv1 .3 dari versi Aurora PostgreSQL berikut:

- Versi 15.3 dan semua yang lebih tinggi
- Versi 14.8 dan versi 14 yang lebih tinggi
- 13.11 dan versi 13 yang lebih tinggi
- 12.15 dan versi 12 yang lebih tinggi
- Versi 11.20 dan versi 11 yang lebih tinggi

Untuk informasi umum tentang dukungan SSL/TLS dan basis data PostgreSQL, lihat SSL support dalam dokumentasi PostgreSQL. Untuk informasi tentang menggunakan koneksi SSL/TLS melalui JDBC, lihat Configuring the client dalam dokumentasi PostgreSQL.

Topik

- Memerlukan klaster DB Aurora PostgreSQL melalui SSL/TLS
- Menentukan status koneksi SSL/TLS
- Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL

Dukungan SSL/TLS tersedia di semua Wilayah untuk AWS Aurora PostgreSQL. Amazon RDS membuat SSL/TLS certificate for your Aurora PostgreSQL DB cluster when the DB cluster is created. If you enable SSL/TLS certificate verification, then the SSL/TLS certificate includes the DB cluster endpoint as the Common Name (CN) for the SSL/TLS sertifikat untuk mencegah serangan spoofing.

Untuk menghubungkan ke klaster DB Aurora PostgreSQL melalui SSL/TLS

1. Unduh sertifikatnya.

   Untuk informasi tentang mengunduh sertifikat, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .
2. Impor sertifikat ke dalam sistem operasi Anda.
3. Hubungkan ke klaster DB Aurora PostgreSQL melalui SSL/TLS.

   Saat Anda menghubungkan menggunakan SSL/TLS, klien Anda dapat memilih untuk memverifikasi rantai sertifikat atau tidak. Jika parameter koneksi Anda menentukan `sslmode=verify-ca` atau `sslmode=verify-full`, maka klien Anda mengharuskan sertifikat

RDS CA berada di penyimpanan kepercayaan atau direferensikan di URL koneksi. Persyaratan ini untuk memverifikasi rantai sertifikat yang menandatangani sertifikat basis data Anda.

Ketika klien, seperti psql atau JDBC, dikonfigurasi dengan. SSL/TLS support, the client first tries to connect to the database with SSL/TLS by default. If the client can't connect with SSL/TLS, it reverts to connecting without SSL/TLS Secara default, opsi `sslmode` untuk klien berbasis JDBC dan libpq diatur ke `prefer`.

Gunakan parameter `sslrootcert` untuk mereferensikan sertifikat, misalnya `sslrootcert=rds-ssl-ca-cert.pem`.

Berikut ini adalah contoh penggunaan psql untuk terhubung ke klaster DB Aurora PostgreSQL.

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

## Memerlukan klaster DB Aurora PostgreSQL melalui SSL/TLS

Untuk memerlukan koneksi SSL/TLS ke cluster Aurora PostgreSQL DB Anda, gunakan parameter. `rds.force_ssl`

- Untuk memerlukan koneksi SSL/TLS, atur nilai `rds.force_ssl` parameter ke 1 (on).
- Untuk mematikan koneksi SSL/TLS yang diperlukan, atur nilai `rds.force_ssl` parameter ke 0 (off).

Nilai default parameter ini tergantung pada versi Aurora PostgreSQL:

- Untuk Aurora PostgreSQL versi 17 dan yang lebih baru: Nilai default adalah 1 (on).
- Untuk Aurora PostgreSQL versi 16 dan yang lebih lama: Nilai default adalah 0 (off).

> **Note**
>
> Saat Anda melakukan upgrade versi mayor dari Aurora PostgreSQL versi 16 atau versi sebelumnya ke versi 17 atau yang lebih baru, nilai default parameter berubah dari 0 (off) menjadi 1 (on). Perubahan ini dapat menyebabkan kegagalan konektivitas untuk aplikasi

yang tidak dikonfigurasi untuk SSL. Anda dapat kembali ke perilaku default sebelumnya dengan menyetel parameter ini ke 0 (off).

Untuk informasi selengkapnya tentang penanganan parameter, lihat.

Memperbarui parameter `rds.force_ssl` juga mengatur parameter `ssl` PostgreSQL ke 1 (aktif) dan mengubah file `pg_hba.conf` klaster DB Anda untuk mendukung konfigurasi SSL/TLS baru.

Saat `rds.force_ssl` parameter diatur ke 1 untuk klaster DB, Anda melihat output yang serupa dengan yang berikut ini saat Anda terhubung, menunjukkan bahwa SSL/TLS sekarang diperlukan:

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

## Menentukan status koneksi SSL/TLS

Status terenkripsi dari koneksi Anda ditunjukkan pada banner logon saat Anda terhubung ke klaster DB.

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

Anda juga dapat memuat ekstensi `sslinfo`, lalu memanggil fungsi `ssl_is_used()` untuk menentukan apakah SSL/TLS digunakan. Fungsi akan kembali `t` jika koneksi menggunakan SSL/TLS, jika tidak akan menghasilkan `f`.

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

Anda dapat menggunakan perintah `select ssl_cipher()` untuk menentukan cipher SSL/TLS:

```
postgres=> select ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

Jika Anda mengaktifkan `set rds.force_ssl` dan memulai ulang klaster DB, koneksi non-SSL ditolak dengan pesan berikut ini:

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

Untuk informasi tentang opsi `sslmode`, lihat Database connection control functions dalam dokumentasi PostgreSQL.

## Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS/SSL klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Melakukan hal ini membantu mencegah penggunaan cipher yang tidak aman atau usang.

Suite cipher yang dapat dikonfigurasi didukung di Aurora PostgreSQL versi 11.8 dan lebih tinggi.

Untuk menentukan daftar cipher yang diizinkan untuk mengenkripsi koneksi, modifikasi parameter klaster `ssl_ciphers`. Setel `ssl_ciphers` parameter ke string nilai cipher yang dipisahkan koma dalam grup parameter cluster menggunakan AWS Management Console, AWS CLI, atau RDS API. Untuk mengatur parameter klaster, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.

Tabel berikut menunjukkan cipher yang didukung untuk versi mesin Aurora PostgreSQL yang valid.

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| 9.6, 10.20 dan lebih rendah, 11.15 dan lebih rendah, 12.10 dan lebih rendah, 13.6 dan lebih rendah | DHE-RSA- -SHA AES128 | Ya | Tidak | Tidak |
| | DHE-RSA- - AES128 SHA256 | Tidak | Ya | Tidak |
| | DHE-RSA- -GCM- AES128 SHA256 | Tidak | Ya | Tidak |
| | DHE-RSA- -SHA AES256 | Tidak | Ya | Tidak |
| | DHE-RSA- - AES256 SHA256 | Tidak | Ya | Tidak |
| | DHE-RSA- -GCM- AES256 SHA384 | Tidak | Ya | Tidak |
| | ECDHE-ECDSA- - SHA AES256 | Tidak | Ya | Tidak |
| | ECDHE-ECDSA- -GCM- AES256 SHA384 | Tidak | Ya | Tidak |
| | ECDHE-RSA- - AES256 SHA384 | Ya | Ya | Tidak |
| | ECDHE-RSA- -SHA AES128 | Tidak | Ya | Tidak |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| | ECDHE-RSA- -AES128 SHA256<br>ECDHE-RSA- -GCM-AES128 SHA256<br>ECDHE-RSA- -SHA AES256<br>ECDHE-RSA- -GCM-AES256 SHA384 | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| 10.21, 11.16, 12.11, 13.7, 14.3, dan 14.4 | ECDHE-RSA- -SHATLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | Ya | Ya | Tidak |
| | AES128 | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | Tidak | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | | | |
| | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | | | |
| | TLS_ECDHE_RSA_WITH | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| | _AES_256_GCM_ SHA384<br><br>TLS_RSA_W ITH_AES_256_GCM_ SHA384<br><br>TLS_RSA_W ITH_AES_2 56_CBC_SHA<br><br>TLS_RSA_W ITH_AES_128_GCM_ SHA256<br><br>TLS_RSA_W ITH_AES_1 28_CBC_SHA<br><br>TLS_ECDHE _RSA_DENGAN_ 0_ 05_ CHACHA2 POLY13 SHA256 | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| 10.22, 11.17, 12.12, 13.8, 14.5, dan 15.2 | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | Tidak | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | Tidak | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | Ya | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| | TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA<br><br>TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384<br><br>TLS_RSA_W ITH_AES_256_GCM_ SHA384<br><br>TLS_RSA_W ITH_AES_2 56_CBC_SHA<br><br>TLS_RSA_W ITH_AES_128_GCM_ SHA256<br><br>TLS_RSA_W ITH_AES_128_CBC_ SHA256<br><br>TLS_RSA_W ITH_AES_1 28_CBC_SHA<br><br>TLS_ECDHE _RSA_DENGAN_ 0_ 05_ CHACHA2 POLY13 SHA256 | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| 11.20, 12.15, 13.11, 14.8, 15.3, 16.1 dan lebih tinggi | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | Tidak | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | Tidak | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | | Tidak | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | Ya | Ya | Tidak |
| | | Ya | Ya | Tidak |
| | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | Tidak | Ya | Tidak |
| | | Tidak | Tidak | Ya |
| | | Tidak | Tidak | Ya |
| | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| | TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA<br><br>TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384<br><br>TLS_RSA_W ITH_AES_256_GCM_ SHA384<br><br>TLS_RSA_W ITH_AES_2 56_CBC_SHA<br><br>TLS_RSA_W ITH_AES_128_GCM_ SHA256<br><br>TLS_RSA_W ITH_AES_128_CBC_ SHA256<br><br>TLS_RSA_W ITH_AES_1 28_CBC_SHA<br><br>TLS_ECDHE _RSA_DENGAN_ 0_ 05_ CHACHA2 POLY13 SHA256<br><br>TLS_AES_128_GCM_ SHA256 | | | |

| Versi mesin Aurora PostgreSQL | Cipher yang didukung | TLS 1.1 | TLS 1.2 | TLS 1.3 |
|---|---|---|---|---|
| | TLS_AES_256_GCM_SHA384 | | | |

Anda juga dapat menggunakan perintah describe-engine-default-cluster-parameter CLI untuk menentukan rangkaian sandi mana yang saat ini didukung untuk keluarga grup parameter tertentu. Contoh berikut menunjukkan cara mendapatkan nilai yang diizinkan untuk parameter klaster `ssl_cipher` untuk Aurora PostgreSQL 11.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

Parameter default `ssl_ciphers` ke semua suite cipher yang diizinkan. Untuk informasi selengkapnya tentang cipher, lihat variabel ssl_ciphers dalam dokumentasi PostgreSQL.

# Memperbarui aplikasi untuk terhubung ke klaster DB Aurora PostgreSQL menggunakan sertifikat SSL/TLS baru

Sejak 13 Januari 2023, Amazon RDS telah menerbitkan sertifikat Otoritas Sertifikat (CA) baru untuk terhubung ke klaster DB Aurora menggunakan Lapisan Soket Aman atau Keamanan Lapisan Pengangkutan (SSL/TLS). Setelah itu, Anda dapat menemukan informasi tentang pembaruan aplikasi untuk menggunakan sertifikat baru.

Topik ini dapat membantu Anda menentukan apakah aplikasi klien menggunakan SSL/TLS untuk terhubung ke klaster DB Anda. Jika demikian, Anda dapat memeriksa lebih lanjut apakah aplikasi tersebut memerlukan verifikasi sertifikat untuk terhubung.

> **Note**
>
> Beberapa aplikasi dikonfigurasi untuk terhubung ke klaster DB Aurora PostgreSQL hanya jika aplikasi tersebut berhasil memverifikasi sertifikat pada server.
> Untuk aplikasi tersebut, Anda harus memperbarui penyimpanan kepercayaan aplikasi klien untuk menyertakan sertifikat CA baru.

Setelah memperbarui sertifikat CA di penyimpanan kepercayaan aplikasi klien, Anda dapat merotasi sertifikat di klaster DB Anda. Sebaiknya Anda menguji prosedur ini di lingkungan pengembangan dan pementasan sebelum menerapkannya di lingkungan produksi Anda.

Untuk informasi selengkapnya tentang rotasi sertifikat, lihat Merotasi sertifikat SSL/TLS. Untuk informasi selengkapnya tentang cara mengunduh sertifikat, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke . Untuk informasi tentang menggunakan SSL/TLS dengan klaster DB PostgreSQL, lihat Mengamankan data Aurora PostgreSQL dengan SSL/TLS.

Topik


- Menentukan apakah aplikasi terhubung ke klaster DB Aurora PostgreSQL menggunakan SSL
- Menentukan apakah klien memerlukan verifikasi sertifikat agar dapat terhubung
- Memperbarui penyimpanan kepercayaan aplikasi Anda
- Menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi

## Menentukan apakah aplikasi terhubung ke klaster DB Aurora PostgreSQL menggunakan SSL

Periksa konfigurasi klaster DB untuk nilai parameter `rds.force_ssl`. Secara default, parameter `rds.force_ssl` diatur ke `0` (nonaktif). Jika parameter `rds.force_ssl` diatur ke `1` (aktif), klien harus menggunakan SSL/TLS untuk koneksi. Untuk informasi selengkapnya tentang grup parameter, lihat .

Jika `rds.force_ssl` tidak diatur ke `1` (aktif), buat kueri ke tampilan `pg_stat_ssl` untuk memeriksa koneksi menggunakan SSL. Misalnya, kueri berikut hanya menampilkan koneksi SSL dan informasi tentang klien menggunakan SSL.

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

Hanya baris yang menggunakan koneksi SSL/TLS yang ditampilkan dengan informasi mengenai koneksi. Berikut ini adalah contoh output-nya.

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

Kueri sebelumnya hanya menampilkan koneksi saat ini pada waktu kueri. Tidak adanya hasil tidak menunjukkan bahwa tidak ada aplikasi yang menggunakan koneksi SSL. Koneksi SSL lainnya mungkin dibangun pada waktu yang lain.

## Menentukan apakah klien memerlukan verifikasi sertifikat agar dapat terhubung

Ketika klien, seperti psql atau JDBC, dikonfigurasikan dengan dukungan SSL, klien pertama kali mencoba untuk terhubung ke basis data dengan SSL secara default. Jika klien tidak dapat terhubung dengan SSL, maka akan kembali ke koneksi tanpa SSL. `sslmode`Mode default yang digunakan untuk klien berbasis libpq (seperti psql) dan JDBC diatur ke. `prefer` Sertifikat di server diverifikasi hanya jika `sslrootcert` disediakan dengan `sslmode` diatur ke `verify-ca` atau `verify-full`. Kesalahan akan muncul jika sertifikat tidak valid.

Gunakan PGSSLROOTCERT untuk memverifikasi sertifikat dengan variabel lingkungan PGSSLMODE, dengan PGSSLMODE diatur ke `verify-ca` atau `verify-full`.

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

Gunakan argumen `sslrootcert` untuk memverifikasi sertifikat dengan `sslmode` dalam format string koneksi, dengan `sslmode` diatur ke `verify-ca` atau `verify-full`.

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

Misalnya, dalam kasus sebelumnya, jika Anda menggunakan sertifikat root yang tidak valid, Anda akan melihat kesalahan yang serupa dengan yang berikut pada klien Anda.

```
psql: SSL error: certificate verify failed
```

## Memperbarui penyimpanan kepercayaan aplikasi Anda

Untuk informasi tentang cara memperbarui penyimpanan kepercayaan untuk aplikasi PostgreSQL, lihat Secure TCP/IP connections with SSL dalam dokumentasi PostgreSQL.

> **Note**
>
> Saat memperbarui penyimpanan kepercayaan, Anda dapat mempertahankan sertifikat lama selain menambahkan sertifikat baru.

### Memperbarui penyimpanan kepercayaan aplikasi Anda untuk JDBC

Anda dapat memperbarui penyimpanan kepercayaan untuk aplikasi yang menggunakan JDBC untuk koneksi SSL/TLS.

Untuk informasi tentang cara mengunduh sertifikat root, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

Untuk contoh skrip yang mengimpor sertifikat, lihat Contoh skrip untuk mengimpor sertifikat ke trust store Anda.

# Menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi

Berikut ini berisi informasi tentang cara menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi:

- psql

  Klien diinvokasi dari baris perintah dengan menentukan opsi sebagai string koneksi atau sebagai variabel lingkungan. Untuk koneksi SSL/TLS, opsi yang relevan adalah `sslmode` (variabel lingkungan `PGSSLMODE`), `sslrootcert` (variabel lingkungan `PGSSLROOTCERT`).

  Untuk daftar lengkap opsi, lihat Parameter key words dalam dokumentasi PostgreSQL. Untuk daftar lengkap variabel lingkungan, lihat Environment variables dalam dokumentasi PostgreSQL.

- pgAdmin

  Klien berbasis browser ini adalah antarmuka yang lebih mudah untuk terhubung ke basis data PostgreSQL.

  Untuk informasi tentang cara mengonfigurasi koneksi, lihat dokumentasi pgAdmin.

- JDBC

  JDBC memungkinkan koneksi basis data dengan aplikasi Java.

  Untuk informasi umum tentang cara terhubung ke basis data PostgreSQL dengan JDBC, lihat Connecting to the database dalam dokumentasi PostgreSQL. Untuk informasi tentang terhubung dengan SSL/TLS, lihat Configuring the client dalam dokumentasi PostgreSQL.

- Python

  Pustaka Python yang populer untuk terhubung ke basis data PostgreSQL adalah `psycopg2`.

  Untuk informasi tentang cara menggunakan `psycopg2`, lihat dokumentasi psycopg2. Untuk tutorial singkat tentang cara terhubung ke basis data PostgreSQL, lihat Tutorial Psycopg2. Anda dapat menemukan informasi tentang opsi penerimaan perintah koneksi di Konten modul psycopg2.

> **Important**
>
> Setelah Anda menentukan bahwa koneksi database Anda menggunakan SSL/TLS dan telah memperbarui toko kepercayaan aplikasi Anda, Anda dapat memperbarui database Anda

untuk menggunakan sertifikat 2048-g1. rds-ca-rsa Untuk petunjuk, lihat langkah 3 dalam Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB Anda.

# Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL

Anda dapat menggunakan Kerberos untuk mengautentikasi pengguna saat terhubung ke klaster DB Anda yang menjalankan PostgreSQL. Untuk melakukannya, konfigurasikan cluster DB Anda untuk digunakan AWS Directory Service for Microsoft Active Directory untuk otentikasi Kerberos. AWS Directory Service for Microsoft Active Directory disebut juga AWS Managed Microsoft AD. Ini adalah fitur yang tersedia dengan AWS Directory Service. Untuk mempelajari lebih lanjut, lihat Apa itu AWS Directory Service? dalam Panduan AWS Directory Service Administrasi.

Untuk memulai, buat AWS Managed Microsoft AD direktori untuk menyimpan kredensyal pengguna. Kemudian, berikan domain Active Directory dan informasi lainnya ke klaster DB PostgreSQL Anda. Saat pengguna mengautentikasi dengan klaster DB PostgreSQL, permintaan autentikasi diteruskan ke direktori AWS Managed Microsoft AD .

Dengan menyimpan semua kredensial Anda di direktori yang sama, Anda dapat menghemat waktu dan tenaga. Anda memiliki sebuah lokasi terpusat untuk menyimpan dan mengelola kredensial bagi beberapa klaster DB. Penggunaan direktori juga dapat meningkatkan profil keamanan Anda secara keseluruhan.

Selain itu, Anda dapat mengakses kredensial dari Microsoft Active Directory on-premise Anda sendiri. Untuk melakukannya, buat hubungan domain tepercaya sehingga direktori AWS Managed Microsoft AD mempercayai Microsoft Active Directory on-premise Anda. Dengan cara ini, pengguna Anda dapat mengakses klaster PostgreSQL Anda dengan pengalaman masuk tunggal (SSO) Windows yang sama seperti ketika mereka mengakses beban kerja di jaringan on-premise Anda.

Database dapat menggunakan Kerberos, AWS Identity and Access Management (IAM), atau keduanya Kerberos dan otentikasi IAM. Namun, karena autentikasi Kerberos dan IAM menyediakan metode autentikasi yang berbeda, pengguna basis data tertentu dapat masuk ke basis data hanya menggunakan salah satu metode autentikasi, dan tidak bisa keduanya. Untuk informasi selengkapnya tentang autentikasi IAM, lihat Autentikasi basis data IAM.

Topik

- Ketersediaan Wilayah dan versi
- Ikhtisar autentikasi Kerberos untuk klaster DB PostgreSQL

- Menyiapkan autentikasi Kerberos untuk klaster DB PostgreSQL
- Mengelola Aurora PostgreSQL DB cluster instance PostgreSQL DB dalam domain Active Directory
- Menghubungkan ke PostgreSQL dengan autentikasi Kerberos
- Menggunakan grup keamanan AD untuk kontrol akses Aurora Postgre SQL

## Ketersediaan Wilayah dan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin basis data, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang Ketersediaan wilayah dan versi Aurora PostgreSQL dengan autentikasi Kerberos, lihat Autentikasi Kerberos dengan Aurora PostgreSQL.

## Ikhtisar autentikasi Kerberos untuk klaster DB PostgreSQL

Untuk menyiapkan autentikasi Kerberos untuk klaster DB PostgreSQL, lakukan langkah-langkah berikut, yang dijelaskan secara lebih mendetail nanti:

1. Gunakan AWS Managed Microsoft AD untuk membuat AWS Managed Microsoft AD direktori. Anda dapat menggunakan AWS Management Console, AWS CLI, atau AWS Directory Service API untuk membuat direktori. Pastikan untuk membuka port keluar yang relevan pada grup keamanan direktori sehingga direktori dapat berkomunikasi dengan klaster.
2. Buat peran yang menyediakan akses Amazon Aurora Amazon untuk melakukan panggilan ke direktori Anda. AWS Managed Microsoft AD Untuk melakukannya, buat peran AWS Identity and Access Management (IAM) yang menggunakan kebijakan IAM terkelola. `AmazonRDSDirectoryServiceAccess`

   Agar peran IAM mengizinkan akses, titik akhir AWS Security Token Service (AWS STS) harus diaktifkan di AWS Wilayah yang benar untuk akun Anda AWS . AWS STS endpoint aktif secara default di semua Wilayah AWS, dan Anda dapat menggunakannya tanpa tindakan lebih lanjut. Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan AWS STS di AWS Wilayah di Panduan Pengguna IAM.
3. Buat dan konfigurasikan pengguna di AWS Managed Microsoft AD direktori menggunakan alat Microsoft Active Directory. Untuk informasi selengkapnya tentang membuat pengguna di Active Directory, lihat Mengelola pengguna dan grup di Microsoft AD yang AWS dikelola di Panduan AWS Directory Service Administrasi.

4. Jika Anda berencana untuk menemukan direktori dan instans DB di AWS akun yang berbeda atau virtual private cloud (VPCs), konfigurasikan peering VPC. Untuk informasi selengkapnya, lihat Apa yang dimaksud peering VPC? di Panduan Peering Amazon VPC.
5. Buat atau modifikasi klaster DB PostgreSQL dari konsol, CLI, atau RDS API menggunakan salah satu metode berikut:
   - Membuat dan terhubung ke klaster DB Aurora PostgreSQL
   - Memodifikasi klaster DB Amazon Aurora
   - Memulihkan dari snapshot klaster DB
   - Memulihkan klaster DB ke waktu tertentu

   Anda dapat menemukan cluster di Amazon Virtual Private Cloud (VPC) yang sama dengan direktori atau di akun AWS atau VPC yang berbeda. Saat membuat atau memodifikasi klaster DB PostgreSQL, lakukan hal berikut:
   - Sediakan pengidentifikasi domain (pengidentifikasi `d-*`) yang dihasilkan saat Anda membuat direktori.
   - Beri nama peran IAM yang Anda buat.
   - Pastikan bahwa grup keamanan instans DB dapat menerima lalu lintas masuk dari grup keamanan direktori.
6. Gunakan kredensial pengguna utama RDS untuk terhubung ke klaster DB PostgreSQL. Buat pengguna dalam PostgreSQL untuk diidentifikasi secara eksternal. Pengguna yang diidentifikasi secara eksternal dapat masuk ke klaster DB PostgreSQL menggunakan autentikasi Kerberos.

## Menyiapkan autentikasi Kerberos untuk klaster DB PostgreSQL

Untuk menyiapkan autentikasi Kerberos, lakukan langkah berikut.

Topik

- Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD
- Langkah 2: (Opsional) Buat hubungan kepercayaan antara Active Directory on-premise Anda dan AWS Directory Service
- Langkah 3: Buat peran IAM untuk Amazon RDS untuk mengakses AWS Directory Service
- Langkah 4: Buat dan konfigurasikan pengguna
- Langkah 5: Aktifkan lalu lintas antar-VPC antara direktori dan instans DB
- Langkah 6: Buat atau modifikasi klaster DB PostgreSQL

- Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda
- Langkah 8: Konfigurasikan klien PostgreSQL

## Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD

AWS Directory Service membuat Direktori Aktif yang dikelola sepenuhnya di AWS Cloud. Saat Anda membuat AWS Managed Microsoft AD direktori, AWS Directory Service buat dua pengontrol domain dan server DNS untuk Anda. Server-server direktori dibuat di subnet yang berbeda di VPC. Redundansi ini membantu memastikan bahwa direktori Anda tetap dapat diakses meskipun terjadi kegagalan.

Saat Anda membuat AWS Managed Microsoft AD AWS direktori, Directory Service melakukan tugas-tugas berikut atas nama Anda:

- Menyiapkan Active Directory di dalam VPC Anda.
- Membuat akun administrator direktori dengan nama pengguna `Admin` dan kata sandi yang ditentukan. Anda menggunakan akun ini untuk mengelola direktori.

  > **Important**
  >
  > Pastikan untuk menyimpan kata sandi ini. AWS Directory Service tidak menyimpan kata sandi ini, dan tidak dapat diambil atau diatur ulang.

- Membuat grup keamanan untuk pengendali direktori. Grup keamanan harus mengizinkan komunikasi dengan klaster DB PostgreSQL.

Saat Anda meluncurkan AWS Directory Service for Microsoft Active Directory, AWS buat Unit Organisasi (OU) yang berisi semua objek direktori Anda. OU ini memiliki nama NetBIOS yang Anda masukkan saat membuat direktori, dan terletak di root domain. Root domain dimiliki dan dikelola oleh AWS.

`Admin`Akun yang dibuat dengan AWS Managed Microsoft AD direktori Anda memiliki izin untuk kegiatan administratif yang paling umum untuk OU Anda:

- Membuat, memperbarui, atau menghapus pengguna
- Menambahkan sumber daya ke domain Anda seperti server file atau cetak, lalu menetapkan izin untuk sumber daya tersebut kepada pengguna di OU Anda

- Buat tambahan OUs dan wadah
- Melimpahkan kewenangan
- Memulihkan objek-objek yang dihapus dari Keranjang Sampah Active Directory
- Jalankan modul Active Directory dan Domain Name Service (DNS) untuk Windows PowerShell pada Layanan Active Directory Web

Akun `Admin` juga memiliki hak untuk melakukan aktivitas di seluruh domain berikut:

- Mengelola konfigurasi DNS (menambahkan, menghapus, atau memperbarui catatan, zona, dan penerus)
- Melihat log peristiwa DNS
- Melihat log peristiwa keamanan

Untuk membuat direktori dengan AWS Managed Microsoft AD

1. Di panel navigasi konsol AWS Directory Service, pilih Direktori, lalu pilih Siapkan direktori.
2. Pilih AWS Managed Microsoft AD. AWS Managed Microsoft AD adalah satu-satunya pilihan yang saat ini didukung untuk digunakan dengan Amazon Aurora  RDS.
3. Pilih Berikutnya.
4. Di halaman Masukkan informasi direktori, berikan informasi berikut:

   Edisi

   Pilih edisi sesuai kebutuhan Anda.

   Nama DNS direktori

   Nama berkualifikasi penuh untuk direktori, seperti **`corp.example.com`**.

   Nama NetBIOS direktori

   Nama pendek opsional untuk direktori, seperti `CORP`.

   Deskripsi direktori

   Deskripsi opsional untuk direktori.

Kata sandi admin

Kata sandi administrator direktori. Proses pembuatan direktori menciptakan akun administrator dengan nama pengguna `Admin` dan kata sandi ini.

Kata sandi administrator direktori tidak boleh menyertakan kata "admin". Kata sandi peka terhadap huruf besar/kecil dan harus terdiri dari 8–64 karakter. Kata sandi juga harus berisi setidaknya satu karakter dari tiga di antara empat kategori berikut:

- Huruf kecil (a-z)
- Huruf besar (A-Z)
- Angka (0–9)
- Karakter non-alfanumerik (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Konfirmasi kata sandi

Ketik ulang kata sandi administrator.

> **Important**
>
> Pastikan Anda menyimpan kata sandi ini. AWS Directory Service tidak menyimpan kata sandi ini, dan tidak dapat diambil atau diatur ulang.

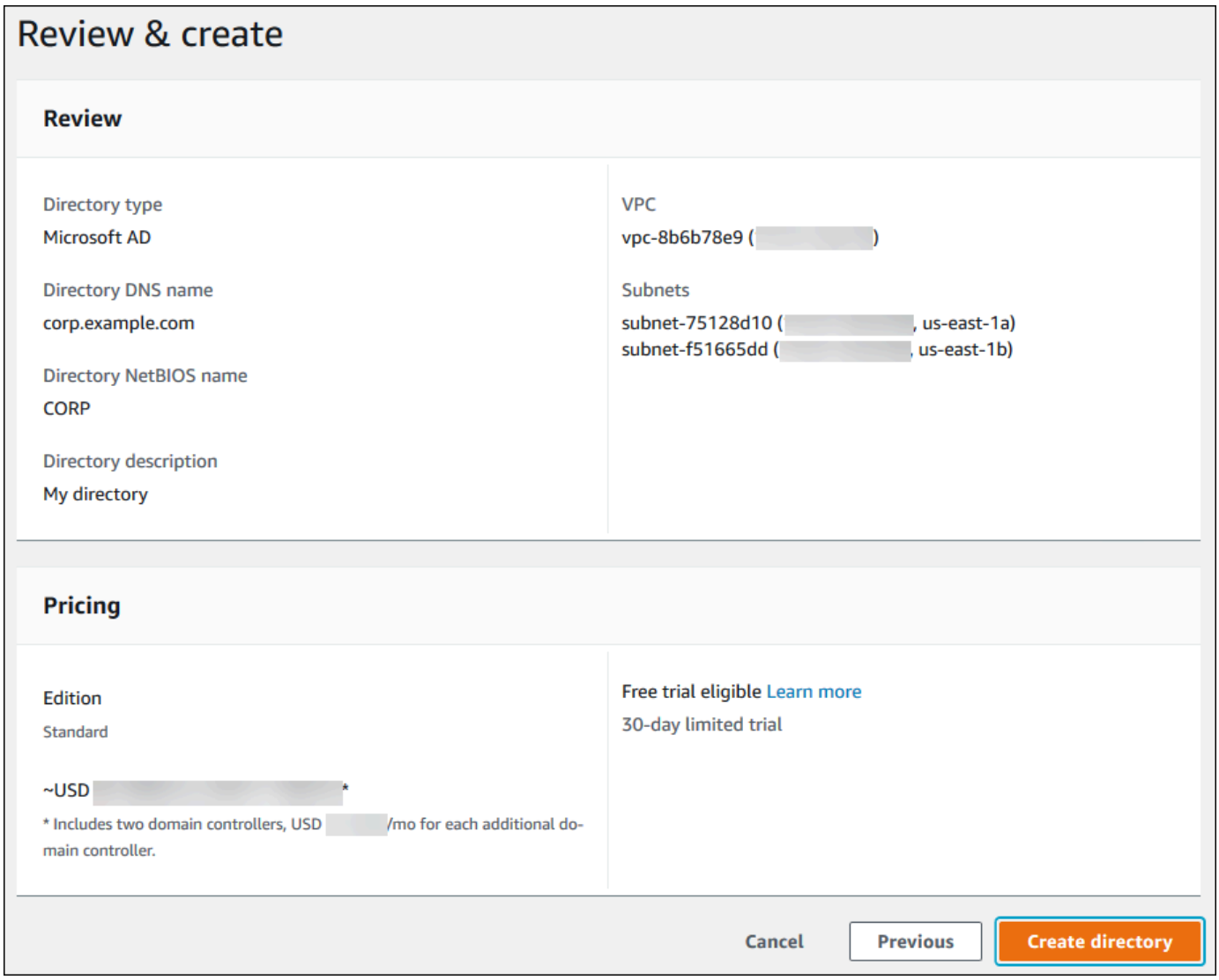
5. Pilih Berikutnya.
6. Di halaman Pilih VPC dan subnet, berikan informasi berikut:

   VPC

   Pilih VPC untuk direktori. Anda dapat membuat klaster DB PostgreSQL dalam VPC yang sama ini atau dalam VPC yang berbeda.

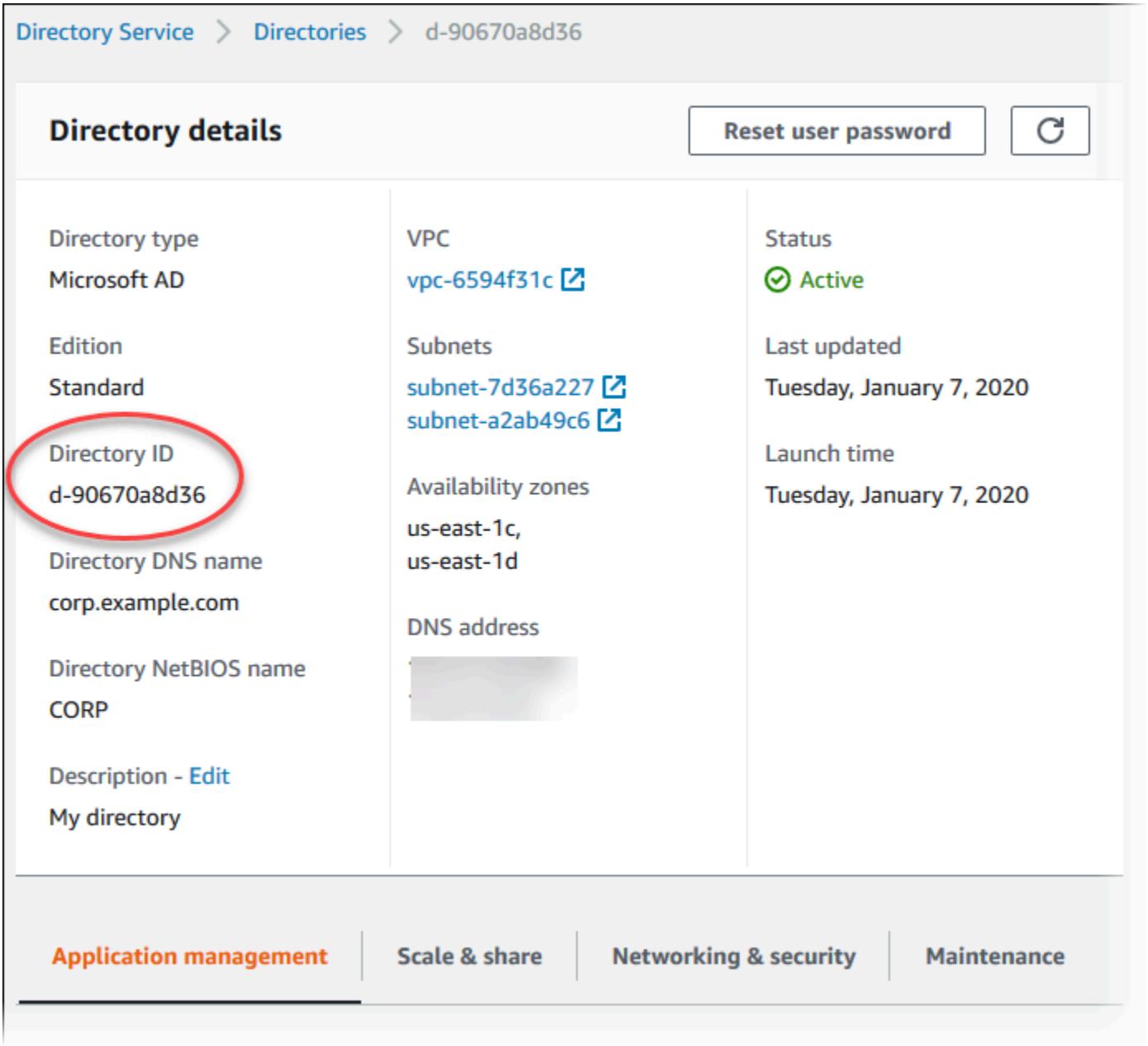
   Subnet

   Pilih subnet untuk server direktori. Kedua subnet harus berada di Zona Ketersediaan yang berbeda.
7. Pilih Berikutnya.
8. Tinjau informasi direktori. Jika ada yang perlu diubah, pilih Sebelumnya dan lakukan perubahan. Jika informasi sudah benar, pilih Buat direktori.




Pembuatan direktori memerlukan waktu beberapa menit. Setelah berhasil dibuat, nilai Status berubah menjadi Aktif.

Untuk melihat informasi tentang direktori Anda, pilih ID direktori di daftar direktori. Buat catatan tentang nilai ID Direktori. Anda memerlukan nilai ini saat membuat atau mengubah instans DB PostgreSQL.




## Langkah 2: (Opsional) Buat hubungan kepercayaan antara Active Directory on-premise Anda dan AWS Directory Service

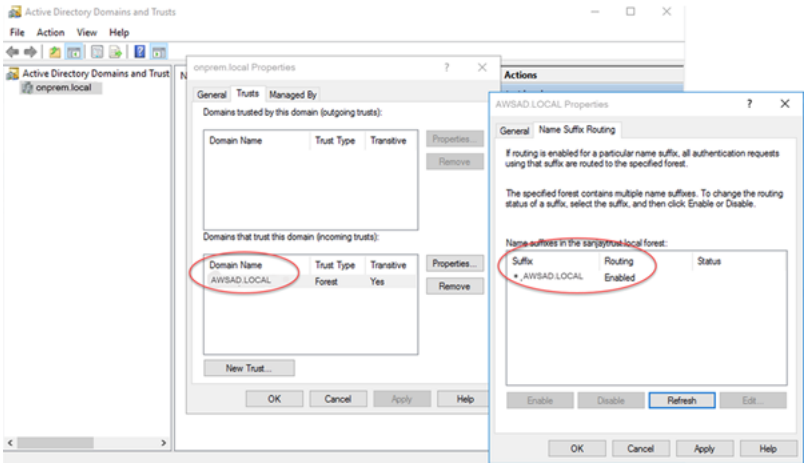
Jika Anda tidak berencana untuk menggunakan Microsoft Active Directory on-premise Anda sendiri, langsung ke Langkah 3: Buat peran IAM untuk Amazon  RDS untuk mengakses AWS Directory Service.

Untuk mendapatkan autentikasi Kerberos menggunakan Active Directory lokal, Anda perlu membuat relasi domain kepercayaan menggunakan trust hutan antara Microsoft Active Directory lokal dan direktori (dibuat di AWS Managed Microsoft AD ). Langkah 1: Buat direktori menggunakan AWS

Managed Microsoft AD Kepercayaan bisa satu arah, di mana AWS Managed Microsoft AD direktori mempercayai Microsoft Active Directory lokal. Kepercayaan juga dapat bersifat dua arah, di mana kedua Active Directory saling mempercayai. Untuk informasi selengkapnya tentang menyiapkan trust menggunakan AWS Directory Service, lihat Kapan membuat hubungan kepercayaan di Panduan AWS Directory Service Administrasi.

> **Note**
>
> Jika Anda menggunakan Microsoft Active Directory on-premise:
>
> - Klien Windows harus terhubung menggunakan nama domain AWS Directory Service di titik akhir daripada rds.amazonaws.com. Untuk informasi selengkapnya, lihat Menghubungkan ke PostgreSQL dengan autentikasi Kerberos.
> - Klien Windows tidak dapat terhubung menggunakan titik akhir kustom Aurora. Untuk mempelajari selengkapnya, lihat Koneksi titik akhir Amazon Aurora.
> - Untuk basis data global:
>   - Klien Windows dapat terhubung menggunakan titik akhir instance atau titik akhir cluster di primer Wilayah AWS database global saja.
>   - Klien Windows tidak dapat terhubung menggunakan titik akhir cluster di sekunder Wilayah AWS.

Pastikan bahwa nama domain Microsoft Active Directory on-premise Anda mencakup perutean akhiran DNS yang sesuai dengan hubungan kepercayaan yang baru dibuat. Tangkapan layar berikut menunjukkan sebuah contoh.



## Langkah 3: Buat peran IAM untuk Amazon RDS untuk mengakses AWS Directory Service

Agar Amazon Aurora Amazon memanggil AWS Directory Service Anda, AWS akun Anda memerlukan peran IAM yang menggunakan kebijakan IAM terkelola. `AmazonRDSDirectoryServiceAccess` Peran ini membuat Amazon Aurora dapat melakukan panggilan ke AWS Directory Service. (Perhatikan bahwa peran IAM ini untuk mengakses berbeda dari peran IAM yang digunakan.) AWS Directory Service Autentikasi basis data IAM

Saat Anda membuat instans DB menggunakan AWS Management Console dan akun pengguna konsol Anda memiliki `iam:CreateRole` izin, konsol akan membuat peran IAM yang diperlukan secara otomatis. Dalam hal ini, nama perannya adalah `rds-directoryservice-kerberos-access-role`. Jika tidak, Anda harus membuat peran IAM secara manual. Saat Anda membuat peran IAM ini, pilih`Directory Service`, dan lampirkan kebijakan AWS terkelola `AmazonRDSDirectoryServiceAccess` padanya.

Untuk informasi selengkapnya tentang membuat peran IAM untuk layanan, lihat Membuat peran untuk mendelegasikan izin ke AWS layanan di Panduan Pengguna IAM.

> **Note**
>
> Peran IAM yang digunakan untuk Autentikasi Windows untuk RDS for Microsoft SQL Server tidak dapat digunakan untuk Amazon Aurora.

Sebagai alternatif untuk menggunakan kebijakan terkelola `AmazonRDSDirectoryServiceAccess`, Anda dapat membuat kebijakan dengan izin yang diperlukan. Dalam hal ini, peran IAM harus memiliki kebijakan kepercayaan IAM berikut.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
```

```
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Peran ini juga harus memiliki kebijakan peran IAM berikut.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Untuk keikutsertaan Wilayah AWS, gunakan prinsip layanan khusus Wilayah dalam kebijakan kepercayaan peran IAM. Saat Anda membuat kebijakan kepercayaan untuk layanan di Wilayah ini, tentukan kode Wilayah di prinsipal layanan.

Contoh berikut menunjukkan kebijakan kepercayaan yang mencakup prinsip layanan khusus Wilayah:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.REGION-CODE.amazonaws.com",
          "rds.REGION-CODE.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Ganti REGION-CODE dengan kode untuk Wilayah spesifik Anda. Misalnya, gunakan prinsip layanan berikut untuk Wilayah Asia Pasifik (Melbourne):

```
"Service": [
  "directoryservice.rds.ap-southeast-4.amazonaws.com",
  "rds.ap-southeast-4.amazonaws.com"
]
```

### Langkah 4: Buat dan konfigurasikan pengguna

Anda dapat membuat pengguna dengan alat Active Directory Users and Computers. Alat ini merupakan salah satu alat Active Directory Domain Services dan Active Directory Lightweight Directory Services. Untuk informasi selengkapnya, lihat Add Users and Computers to the Active Directory domain dalam dokumentasi Microsoft. Dalam hal ini, pengguna adalah individu atau entitas lain, seperti komputer mereka yang merupakan bagian dari domain dan yang identitasnya dipertahankan dalam direktori.

Untuk membuat pengguna di AWS Directory Service direktori, Anda harus terhubung ke EC2 instance Amazon berbasis Windows yang merupakan anggota direktori. AWS Directory Service Pada saat yang sama, Anda harus masuk sebagai pengguna yang memiliki hak untuk membuat pengguna. Untuk informasi selengkapnya, lihat Membuat pengguna dalam Panduan Administrasi AWS Directory Service .

## Langkah 5: Aktifkan lalu lintas antar-VPC antara direktori dan instans DB

Jika Anda ingin menemukan direktori dan klaster DB dalam VPC yang sama, lewati langkah ini dan lanjutkan ke Langkah 6: Buat atau modifikasi klaster DB PostgreSQL.

Jika Anda berencana untuk menemukan direktori dan instans DB yang berbeda VPCs, konfigurasikan lalu lintas lintas VPC menggunakan pengintip VPC atau Transit Gateway.AWS

Prosedur berikut memungkinkan lalu lintas antara VPCs menggunakan VPC peering. Ikuti petunjuk di Apa yang dimaksud dengan peering VPC? dalam Panduan Peering Amazon Virtual Private Cloud.

Untuk mengaktifkan lalu lintas VPC menggunakan peering VPC

1. Siapkan aturan perutean VPC yang sesuai untuk memastikan lalu lintas jaringan dapat berjalan dua arah.
2. Pastikan bahwa grup keamanan instans DB dapat menerima lalu lintas masuk dari grup keamanan direktori.
3. Pastikan tidak ada aturan daftar kontrol akses (ACL) jaringan yang memblokir lalu lintas.

Jika AWS akun lain memiliki direktori, Anda harus berbagi direktori.

Untuk berbagi direktori antar AWS akun

1. Mulai berbagi direktori dengan AWS akun tempat instans DB akan dibuat dengan mengikuti petunjuk di Tutorial: Berbagi direktori AD Microsoft AWS Terkelola Anda untuk EC2 Domain yang mulus-Bergabung dalam Panduan Administrasi.AWS Directory Service
2. Masuk ke AWS Directory Service konsol menggunakan akun untuk instans DB, dan pastikan domain memiliki `SHARED` status sebelum melanjutkan.
3. Saat masuk ke AWS Directory Service konsol menggunakan akun untuk instans DB, perhatikan nilai ID Direktori. Anda menggunakan ID direktori ini untuk menggabungkan instans DB ke domain.

## Langkah 6: Buat atau modifikasi klaster DB PostgreSQL

Buat atau modifikasi klaster DB PostgreSQL untuk digunakan dengan direktori Anda. Anda dapat menggunakan konsol, CLI, atau RDS API untuk mengaitkan klaster DB dengan direktori. Anda dapat melakukannya dengan salah satu cara berikut:

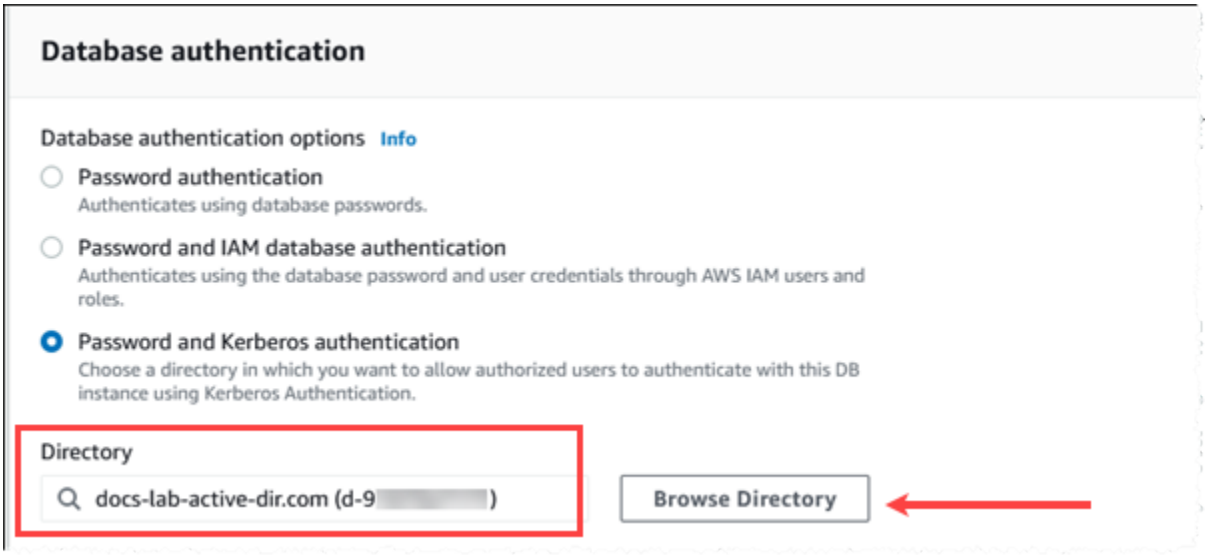
- Buat cluster PostgreSQL DB baru menggunakan konsol, perintah create-db-clusterCLI, atau operasi Create RDS API. DBCluster Untuk petunjuk, lihat Membuat dan terhubung ke klaster DB Aurora PostgreSQL.
- Ubah cluster PostgreSQL DB yang ada menggunakan konsol, perintah modify-db-clusterCLI, atau operasi Modify RDS API. DBCluster Untuk petunjuk, lihat Memodifikasi klaster DB Amazon Aurora.
- Pulihkan cluster PostgreSQL DB dari snapshot DB menggunakan konsol, perintah CLI restore-db-cluster-from-db-snapshot, atau operasi Restore From RDS API. DBCluster DBSnapshot Untuk petunjuk, lihat Memulihkan dari snapshot klaster DB.
- Kembalikan cluster PostgreSQL DB ke point-in-time menggunakan konsol, perintah restore-db-instance-to- point-in-time CLI, atau operasi Restore RDS API. DBCluster ToPointInTime Untuk petunjuk, lihat Memulihkan klaster DB ke waktu tertentu.

Autentikasi Kerberos hanya didukung untuk klaster DB PostgreSQL dalam sebuah VPC. Klaster DB boleh berada dalam VPC yang sama dengan direktori, atau dalam VPC yang berbeda. Klaster DB harus menggunakan grup keamanan yang memungkinkan ingress and egress di dalam VPC direktori, sehingga klaster DB dapat berkomunikasi dengan direktori.

> **Note**
>
> Mengaktifkan otentikasi Kerberos saat ini tidak didukung di klaster Aurora PostgreSQL DB selama migrasi dari RDS untuk PostgreSQL. Anda dapat mengaktifkan otentikasi Kerberos hanya pada cluster Aurora PostgreSQL DB mandiri.

Konsol

Saat Anda menggunakan konsol untuk membuat, memodifikasi, atau memulihkan klaster DB, pilih Autentikasi Kerberos di bagian Autentikasi basis data. Kemudian pilih Cari Direktori. Pilih direktori atau pilih Buat direktori baru untuk menggunakan Directory Service.




AWS CLI

Saat Anda menggunakan AWS CLI, parameter berikut diperlukan untuk cluster DB agar dapat menggunakan direktori yang Anda buat:

- Untuk parameter `--domain`, gunakan pengidentifikasi domain (pengidentifikasi "d-*") yang dihasilkan saat Anda membuat direktori.
- Untuk parameter `--domain-iam-role-name`, gunakan peran yang Anda buat yang menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`.

Misalnya, perintah CLI berikut memodifikasi klaster DB untuk menggunakan direktori.

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

> **Important**
>
> Jika Anda memodifikasi klaster DB untuk mengaktifkan autentikasi Kerberos, boot ulang klaster DB setelah membuat perubahan.

## Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda

Pada titik ini, klaster DB Aurora PostgreSQL Anda digabungkan ke domain AWS Managed Microsoft AD . Pengguna yang Anda buat di direktori Langkah 4: Buat dan konfigurasikan pengguna

perlu diatur sebagai pengguna basis data PostgreSQL dan diberi hak istimewa untuk masuk ke basis data. Anda dapat melakukannya dengan masuk sebagai pengguna basis data dengan hak istimewa `rds_superuser`. Misalnya, jika Anda menerima default saat membuat klaster DB Aurora PostgreSQL, , gunakan `postgres`, seperti yang ditunjukkan dalam langkah berikut.

Untuk membuat pengguna basis data PostgreSQL untuk pengguna utama Kerberos

1. Gunakan `psql` untuk menghubungkan ke titik akhir klaster DB Aurora PostgreSQL instans DB menggunakan `psql`. Contoh berikut menggunakan akun `postgres` default untuk peran `rds_superuser`.

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. Buat nama pengguna basis data untuk setiap pengguna utama Kerberos (nama pengguna Active Directory) yang ingin Anda beri akses ke basis data. Gunakan nama pengguna (identitas) kanonik seperti yang didefinisikan dalam instans Active Directory, yaitu `alias` dalam huruf kecil (nama pengguna di Active Directory) dan nama domain Active Directory dalam huruf besar untuk nama pengguna tersebut. Nama pengguna Active Directory adalah pengguna yang diautentikasi secara eksternal, jadi gunakan tanda kutip pada nama ini seperti yang ditunjukkan berikut.

   ```
   postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
   CREATE ROLE
   ```

3. Beri peran `rds_ad` kepada pengguna basis data.

   ```
   postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
   GRANT ROLE
   ```

Setelah Anda selesai membuat semua pengguna PostgreSQL untuk identitas pengguna Active Directory Anda, pengguna dapat mengakses klaster DB Aurora PostgreSQL menggunakan kredensial Kerberos mereka.

Diperlukan bahwa pengguna database yang mengautentikasi menggunakan Kerberos melakukannya dari mesin klien yang merupakan anggota domain Active Directory.

Pengguna basis data yang telah diberi peran `rds_ad` tidak dapat memiliki peran `rds_iam`. Aturan ini juga berlaku untuk keanggotaan bertingkat. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

Mengonfigurasi klaster DB Aurora PostgreSQL Anda untuk nama pengguna yang tidak peka huruf besar/kecil

Aurora PostgreSQL versi 14.5, 13.8, 12.12, dan 11.17 mendukung parameter `krb_caseins_users` PostgreSQL. Parameter ini mendukung nama pengguna Active Directory yang tidak peka huruf besar/kecil. Secara default, parameter ini diatur ke salah, sehingga nama pengguna ditafsirkan sebagai tidak peka huruf besar/kecil oleh Aurora PostgreSQL. Hal tersebut merupakan perilaku default di semua versi lama Aurora PostgreSQL. Namun, Anda dapat mengatur parameter ini ke `true` dalam grup parameter klaster DB kustom Anda dan mengizinkan klaster DB Aurora PostgreSQL Anda untuk menafsirkan nama pengguna sebagai tidak peka huruf besar/kecil. Sebaiknya lakukan hal ini agar memudahkan pengguna basis data Anda, yang terkadang salah mengetik kapitalisasi nama pengguna saat melakukan autentikasi menggunakan Active Directory.

Untuk mengubah parameter `krb_caseins_users`, klaster DB Aurora PostgreSQL Anda harus menggunakan grup parameter klaster DB kustom. Untuk informasi tentang penggunaan grup parameter klaster DB kustom, lihat .

Anda dapat menggunakan AWS CLI atau AWS Management Console untuk mengubah pengaturan. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.

## Langkah 8: Konfigurasikan klien PostgreSQL

Untuk mengonfigurasi klien PostgreSQL, lakukan langkah berikut:

- Buat file krb5.conf (atau yang setara) untuk menunjuk ke domain.
- Verifikasi bahwa lalu lintas dapat mengalir antara host klien dan AWS Directory Service. Gunakan utilitas jaringan seperti Netcat untuk hal berikut:
  - Memeriksa lalu lintas melalui DNS untuk port 53.
  - Verifikasi lalu lintas TCP/UDP untuk port 53 dan untuk Kerberos, yang mencakup port 88 dan 464 untuk. AWS Directory Service
- Memastikan lalu lintas dapat mengalir di antara host klien dan instans DB melalui port basis data. Misalnya, gunakan psql untuk menghubungkan dan mengakses basis data.

Berikut ini adalah contoh konten krb5.conf untuk. AWS Managed Microsoft AD

```
[libdefaults]
 default_realm = EXAMPLE.COM
```

```
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

Berikut adalah contoh konten krb5.conf untuk Microsoft Active Directory on-premise.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

## Mengelola Aurora PostgreSQL DB cluster instance PostgreSQL DB dalam domain Active Directory

Anda dapat menggunakan konsol, CLI, atau RDS API untuk mengelola DB cluster DB Anda dan hubungannya dengan Microsoft Active Directory Anda. Misalnya, Anda dapat mengaitkan Active Directory untuk mengaktifkan autentikasi Kerberos. Anda juga dapat menghapus pengaitan Active Directory untuk menonaktifkan autentikasi Kerberos. Anda juga dapat memindahkan klaster DB untuk diautentikasi secara eksternal oleh satu Microsoft Active Directory ke yang lain.

Misalnya, dengan CLI, Anda dapat melakukan hal berikut:

- Untuk mencoba kembali mengaktifkan otentikasi Kerberos untuk keanggotaan yang gagal, gunakan perintah CLI. Tentukan ID direktori keanggotaan saat ini untuk opsi `--domain`.
- Untuk menonaktifkan otentikasi Kerberos pada instance DB, gunakan perintah CLI . Tentukan `none` untuk opsi `--domain`.
- Untuk memindahkan instance DB dari satu domain ke domain lainnya, gunakan perintah CLI. Tentukan pengidentifikasi domain dari domain baru untuk opsi `--domain`.

## Memahami keanggotaan Domain

Setelah Anda membuat atau memodifikasi klaster DB, instans DB menjadi anggota domain. Anda dapat melihat status keanggotaan domain di konsol atau dengan menjalankan perintah describe-db-instancesCLI. Status instans DB dapat berupa salah satu dari daftar berikut:

- `kerberos-enabled` – Instans basis data telah mengaktifkan autentikasi Kerberos.
- `enabling-kerberos`— AWS sedang dalam proses mengaktifkan otentikasi Kerberos pada instance DB ini.
- `pending-enable-kerberos` – Aktivasi autentikasi Kerberos pada instans DB ini tertunda.
- `pending-maintenance-enable-kerberos`— AWS akan mencoba mengaktifkan otentikasi Kerberos pada instans DB selama jendela pemeliharaan terjadwal berikutnya.
- `pending-disable-kerberos` – Penonaktifan autentikasi Kerberos pada instans DB ini tertunda.
- `pending-maintenance-disable-kerberos`— AWS akan mencoba menonaktifkan otentikasi Kerberos pada instans DB selama jendela pemeliharaan terjadwal berikutnya.
- `enable-kerberos-failed`— Masalah konfigurasi dicegah AWS dari mengaktifkan otentikasi Kerberos pada instans DB. Perbaiki masalah konfigurasi tersebut sebelum menerbitkan ulang perintah untuk memodifikasi instans DB.
- `disabling-kerberos`— AWS sedang dalam proses menonaktifkan otentikasi Kerberos pada instance DB ini.

Permintaan untuk mengaktifkan autentikasi Kerberos dapat gagal karena masalah koneksi jaringan atau kesalahan peran IAM. Dalam beberapa kasus, upaya untuk mengaktifkan autentikasi Kerberos mungkin gagal saat Anda membuat atau memodifikasi klaster DB. Jika demikian, pastikan Anda menggunakan peran IAM yang benar, kemudian ubah klaster DB untuk bergabung ke domain.

# Menghubungkan ke PostgreSQL dengan autentikasi Kerberos

Anda dapat terhubung ke PostgreSQL dengan autentikasi Kerberos, antarmuka pgAdmin, atau antarmuka baris perintah seperti psql. Untuk informasi selengkapnya tentang cara menghubungkan, lihat Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.. Untuk informasi tentang mendapatkan titik akhir, nomor port, dan detail lain yang diperlukan untuk koneksi, lihat Melihat titik akhir untuk klaster Aurora.

> **Note**
>
> Otentikasi dan enkripsi GSSAPI di PostgreSQL diimplementasikan oleh perpustakaan Kerberos. `libkrb5.so` Fitur seperti `postgres_fdw` dan `dblink` juga mengandalkan perpustakaan yang sama ini untuk koneksi keluar dengan otentikasi atau enkripsi Kerberos.

pgAdmin

Untuk menggunakan pgAdmin untuk terhubung ke PostgreSQL dengan autentikasi Kerberos, lakukan langkah berikut:

1. Luncurkan aplikasi pgAdmin di komputer klien Anda.
2. Pada tab Dasbor, pilih Tambahkan Server Baru.
3. Di kotak dialog Buat - Server, masukkan nama pada tab Umum untuk mengidentifikasi server di pgAdmin.
4. Pada tab Koneksi, masukkan informasi berikut dari basis data Aurora PostgreSQL Anda.
   - Untuk Host, masukkan titik akhir untuk instans Penulis klaster DB Aurora PostgreSQL Anda. Titik akhir terlihat seperti berikut ini:

     ```
     AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     Untuk menyambung ke Microsoft Active Directory lokal dari klien Windows, Anda menggunakan nama domain Direktori Aktif AWS Terkelola, bukan `rds.amazonaws.com` di titik akhir host. Misalnya, misalkan nama domain untuk AWS Managed Active Directory adalah`corp.example.com`. Kemudian untuk Host, titik akhir akan ditentukan sebagai berikut:

     ```
     AUR-cluster-instance.111122223333.aws-region.corp.example.com
     ```

- Untuk Port, masukkan port yang ditetapkan.
- Untuk Basis data pemeliharaan, masukkan nama basis data awal yang akan dihubungkan ke klien.
- Untuk Nama pengguna, masukkan nama pengguna yang Anda masukkan untuk autentikasi Kerberos di Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda .

5. Pilih Simpan.

Psql

Untuk menggunakan psql untuk terhubung ke PostgreSQL dengan autentikasi Kerberos, lakukan langkah berikut:

1. Pada jendela perintah, jalankan perintah berikut.

```
kinit username
```

   Ganti *username* dengan nama pengguna. Saat diminta, masukkan kata sandi yang disimpan dalam Microsoft Active Directory untuk pengguna.

2. Jika instance PostgreSQL DB menggunakan VPC yang dapat diakses publik, masukkan alamat IP untuk titik akhir instans DB Anda di file Anda di klien. `/etc/hosts` EC2 Misalnya, perintah berikut mendapatkan alamat IP lalu memasukkannya ke dalam file `/etc/hosts`.

```
% dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
;; Truncated, retrying in TCP mode.
ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
34.210.197.118

% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
hosts
```

   Jika Anda menggunakan Microsoft Active Directory on-premise dari klien Windows, Anda perlu terhubung menggunakan titik akhir khusus. Alih-alih menggunakan domain Amazon `rds.amazonaws.com` di titik akhir host, gunakan nama domain Direktori Aktif AWS Terkelola.

   Misalnya, misalkan nama domain untuk Direktori Aktif AWS Terkelola Anda adalah`corp.example.com`. Kemudian gunakan format *PostgreSQL-endpoint.AWS-Region.corp.example.com* untuk titik akhir dan masukkan ke dalam file `/etc/hosts`.

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
hosts
```

3. Gunakan perintah psql berikut untuk masuk ke klaster DB PostgreSQL yang terintegrasi dengan Active Directory. Gunakan titik akhir klaster atau instans.

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.rds.amazonaws.com postgres
```

Untuk masuk ke klaster DB PostgreSQL dari klien Windows menggunakan Active Directory on-premise, gunakan perintah psql berikut dengan nama domain dari langkah sebelumnya (`corp.example.com`):

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

# Menggunakan grup keamanan AD untuk kontrol akses Aurora Postgre SQL

Dari Aurora Postgre SQL 14.10 dan 15.5 versi, kontrol akses Aurora Postgre dapat dikelola menggunakan SQL AWS Directory Service untuk grup keamanan Microsoft Active Directory (AD). Versi Aurora Postgre sebelumnya SQL mendukung otentikasi berbasis Kerberos dengan AD hanya untuk pengguna individu. Setiap pengguna AD harus secara eksplisit disediakan ke cluster DB untuk mendapatkan akses.

Alih-alih secara eksplisit menyediakan setiap pengguna AD ke klaster DB berdasarkan kebutuhan bisnis, Anda dapat memanfaatkan grup keamanan AD seperti yang dijelaskan di bawah ini:

- Pengguna AD adalah anggota dari berbagai grup keamanan AD di Active Directory. Ini tidak ditentukan oleh administrator cluster DB, tetapi didasarkan pada persyaratan bisnis, dan ditangani oleh administrator AD.
- Administrator cluster DB membuat peran DB dalam instans DB berdasarkan persyaratan bisnis. Peran DB ini mungkin memiliki izin atau hak istimewa yang berbeda.
- Administrator cluster DB mengonfigurasi pemetaan dari grup keamanan AD ke peran DB berdasarkan per cluster DB.
- Pengguna DB dapat mengakses kluster DB menggunakan kredensyal AD mereka. Akses didasarkan pada keanggotaan grup keamanan AD. Pengguna iklan mendapatkan atau kehilangan akses secara otomatis berdasarkan keanggotaan grup iklan mereka.

## Prasyarat

Pastikan Anda memiliki hal berikut sebelum menyiapkan ekstensi untuk grup Keamanan AD:

- Siapkan otentikasi Kerberos untuk cluster SQL Postgre DB. Untuk informasi selengkapnya, lihat Menyiapkan otentikasi Kerberos untuk klaster SQL Postgre DB.

  > **Note**
  >
  > Untuk grup keamanan AD, lewati Langkah 7: Buat SQL pengguna Postgre untuk kepala sekolah Kerberos Anda dalam prosedur penyiapan ini.

- Mengelola cluster DB dalam Domain. Untuk informasi selengkapnya, lihat Mengelola cluster DB di Domain.

## Menyiapkan ekstensi pg_ad_mapping

Aurora Postgre SQL sekarang menyediakan `pg_ad_mapping` ekstensi untuk mengelola pemetaan antara grup keamanan AD dan peran DB di cluster Aurora Postgre. SQL Untuk informasi selengkapnya tentang fungsi yang disediakan oleh`pg_ad_mapping`, lihat[Menggunakan fungsi dari `pg_ad_mapping` ekstensi](#).

Untuk menyiapkan `pg_ad_mapping` ekstensi pada cluster SQL DB Aurora Postgre Anda, pertama-tama Anda menambahkan `pg_ad_mapping` ke pustaka bersama pada grup parameter cluster DB kustom untuk cluster DB Aurora Postgre Anda. SQL Untuk informasi tentang membuat grup parameter cluster DB kustom, lihat. Selanjutnya, Anda menginstal `pg_ad_mapping` ekstensi. Prosedur di bagian ini menunjukkan caranya kepada Anda. Anda dapat menggunakan AWS Management Console atau AWS CLI.

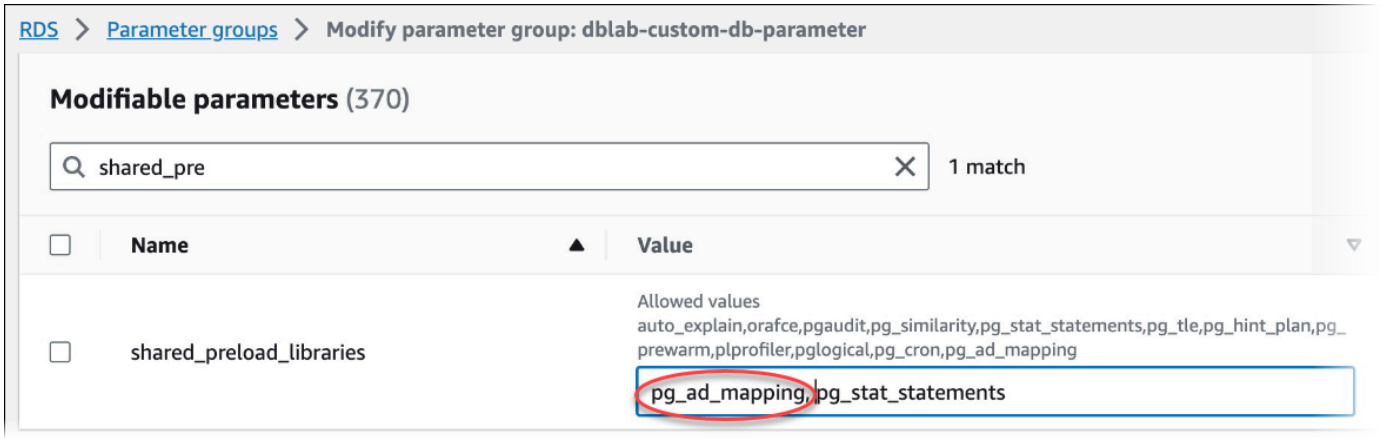
Anda harus memiliki izin sebagai peran `rds_superuser` untuk melakukan semua tugas ini.

Langkah-langkah berikut mengasumsikan bahwa cluster SQL DB Aurora Postgre Anda dikaitkan dengan grup parameter cluster DB kustom.

Konsol

Untuk mengatur **`pg_ad_mapping`** ekstensi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih instance Writer cluster Aurora SQL Postgre DB Anda.
3. Buka tab Konfigurasi untuk instance penulis cluster Aurora Postgre SQL DB Anda. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter khusus yang terkait dengan cluster Aurora SQL Postgre DB Anda.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pg_ad_mapping` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.




8. Reboot instance penulis cluster Aurora Postgre SQL DB Anda sehingga perubahan Anda pada `shared_preload_libraries` parameter berlaku.
9. Ketika instans tersedia, verifikasi bahwa `pg_ad_mapping` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda, dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. Dengan `pg_ad_mapping` diinisialisasi, Anda sekarang dapat membuat ekstensi. Anda perlu membuat ekstensi setelah menginisialisasi perpustakaan untuk mulai menggunakan fungsi yang disediakan oleh ekstensi ini.

```
CREATE EXTENSION pg_ad_mapping;
```

11. Tutup sesi `psql`.

```
labdb=> \q
```

AWS CLI

Untuk mengatur pg_ad_mapping

Untuk mengatur pg_ad_mapping menggunakan AWS CLI, Anda memanggil modify-db-parameter-groupoperasi untuk menambahkan parameter ini di grup parameter kustom Anda, seperti yang ditunjukkan dalam prosedur berikut.

1. Gunakan yang berikut ini AWS CLI perintah untuk `pg_ad_mapping` menambah `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan yang berikut ini AWS CLI perintah untuk me-reboot instance penulis cluster Aurora Postgre SQL DB Anda sehingga pg_ad_mapping diinisialisasi.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Saat instans tersedia, verifikasikan bahwa `pg_ad_mapping` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda, dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

   Dengan pg_ad_mapping diinisialisasi, Anda sekarang dapat membuat ekstensi.

```
CREATE EXTENSION pg_ad_mapping;
```

4. Tutup `psql` sesi sehingga Anda dapat menggunakan AWS CLI.

```
labdb=> \q
```

## Mengambil Grup SID Direktori Aktif di PowerShell

Security identifier (SID) digunakan untuk mengidentifikasi prinsip keamanan atau kelompok keamanan secara unik. Setiap kali grup keamanan atau akun dibuat di Active Directory a SID ditugaskan untuk itu. Untuk mengambil grup keamanan AD SID dari direktori aktif, Anda dapat menggunakan Get- ADGroup cmdlet dari mesin klien windows yang digabungkan dengan domain Active Directory tersebut. Parameter Identity menentukan nama grup Active Directory untuk mendapatkan yang sesuaiSID.

Contoh berikut mengembalikan SID grup AD *adgroup1*.

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

## Memetakan peran DB dengan grup keamanan AD

Anda perlu secara eksplisit menyediakan grup keamanan AD dalam database sebagai peran SQL Postgre DB. Pengguna AD, yang merupakan bagian dari setidaknya satu grup keamanan AD yang disediakan akan mendapatkan akses ke database. Anda tidak boleh memberikan `rds_ad role` peran DB berbasis keamanan grup AD. Otentikasi Kerberos untuk grup keamanan akan dipicu dengan menggunakan akhiran nama domain seperti *user1@example.com*. Peran DB ini tidak dapat menggunakan Kata Sandi atau IAM otentikasi untuk mendapatkan akses ke database.

> **Note**
>
> Pengguna AD yang memiliki peran DB terkait dalam database dengan `rds_ad` peran yang diberikan kepada mereka, tidak dapat masuk sebagai bagian dari grup keamanan AD. Mereka akan mendapatkan akses melalui peran DB sebagai pengguna individu.

Misalnya, grup akun adalah grup keamanan di AD tempat Anda ingin menyediakan grup keamanan ini di Aurora Postgre sebagai peran akun. SQL

| Grup Keamanan AD | Peran Posgre DB SQL |
| --- | --- |
| akun-grup | akun-peran |

Saat memetakan peran DB dengan grup keamanan AD, Anda harus memastikan bahwa peran DB memiliki LOGIN atribut yang ditetapkan dan memiliki CONNECT hak istimewa untuk database login yang diperlukan.

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

Admin sekarang dapat melanjutkan untuk membuat pemetaan antara grup keamanan AD dan peran Postgre DBSQL.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

Untuk informasi tentang pengambilan SID grup keamanan AD, lihatMengambil Grup SID Direktori Aktif di PowerShell.

Mungkin ada kasus di mana pengguna AD termasuk dalam beberapa grup, dalam hal ini, pengguna AD akan mewarisi hak istimewa peran DB, yang disediakan dengan bobot tertinggi. Jika kedua peran memiliki bobot yang sama, pengguna AD akan mewarisi hak istimewa peran DB yang sesuai dengan pemetaan yang ditambahkan baru-baru ini. Rekomendasinya adalah untuk menentukan bobot yang mencerminkan izin/hak istimewa relatif dari peran DB individu. Lebih tinggi izin atau hak istimewa peran DB, lebih tinggi bobot yang harus dikaitkan dengan entri pemetaan. Ini akan menghindari ambiguitas dua pemetaan yang memiliki bobot yang sama.

Tabel berikut menunjukkan contoh pemetaan dari grup keamanan AD ke peran Aurora SQL Postgre DB.

| Grup Keamanan AD | Peran Posgre DB SQL | Berat Badan |
|---|---|---|
| akun-grup | akun-peran | 7 |
| kelompok penjualan | peran penjualan | 10 |
| kelompok pengembang | peran pengembang | 7 |

Dalam contoh berikut, `user1` akan mewarisi hak istimewa peran penjualan karena memiliki bobot yang lebih tinggi sementara `user2` akan mewarisi hak istimewa sebagai pemetaan untuk peran ini dibuat `dev-role` setelahnya`accounts-role`, yang memiliki bobot yang sama dengan. `accounts-role`

| nama pengguna | Keanggotaan Grup Keamanan |
|---|---|
| user1 | akun-kelompok penjualan-kelompok |
| user2 | akun-grup dev-group |

Perintah psql untuk membuat, membuat daftar, dan menghapus pemetaan ditunjukkan di bawah ini. Saat ini, tidak mungkin untuk memodifikasi entri pemetaan tunggal. Entri yang ada perlu dihapus dan pemetaan dibuat ulang.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid       | pg_role        | weight | ad_grp
-------------+----------------+--------+--------------
S-1-5-67-890 | accounts-role | 7      | accounts-group
S-1-2-34-560 | sales-role    | 10     | sales-group
```

```
S-1-8-43-612 | dev-role      | 7      | dev-group
(3 rows)
```

## Pencatatan/audit identitas pengguna AD

Gunakan perintah berikut untuk menentukan peran database yang diwarisi oleh pengguna saat ini atau sesi:

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

Untuk menentukan identitas utama keamanan AD, gunakan perintah berikut:

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

Saat ini, identitas pengguna AD tidak terlihat di log audit. `log_connections`Parameter dapat diaktifkan untuk mencatat pembentukan sesi DB. Untuk informasi selengkapnya, lihat log_connections. Output untuk ini termasuk identitas pengguna AD, seperti yang ditunjukkan di bawah ini. Backend PID yang terkait dengan output ini kemudian dapat membantu tindakan atribut kembali ke pengguna AD yang sebenarnya.

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

## Batasan

- ID Microsoft Entra yang dikenal sebagai Azure Active Directory tidak didukung.

## Menggunakan fungsi dari **`pg_ad_mapping`** ekstensi

`pg_ad_mapping`ekstensi memberikan dukungan untuk fungsi-fungsi berikut:

pgadmap_set_mapping

Fungsi ini menetapkan pemetaan antara grup keamanan AD dan peran Database dengan bobot terkait.

Sintaks

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

Argumen

| Parameter | Deskripsi |
| --- | --- |
| ad_group | Nama Grup AD. Nilai tidak bisa null atau string kosong. |
| db_role | Peran database yang akan dipetakan ke Grup AD yang ditentukan. Nilai tidak bisa null atau string kosong. |
| ad_group_id | Pengidentifikasi keamanan yang digunakan untuk mengident ifikasi grup AD secara unik. Nilai dimulai dengan 'S-1-' dan tidak dapat berupa string nol atau kosong. Untuk informasi selengkap nya, lihat Mengambil Grup SID Direktori Aktif di PowerShell. |
| berat | Berat yang terkait dengan peran database. Peran dengan bobot tertinggi diutamakan ketika pengguna adalah anggota dari beberapa grup. Nilai default berat adalah 1. |

Jenis pengembalian

None

Catatan penggunaan

Fungsi ini menambahkan pemetaan baru dari grup keamanan AD ke peran database. Itu hanya dapat dieksekusi pada instance DB utama dari cluster DB oleh pengguna yang memiliki hak istimewa rds_superuser.

Contoh

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

Fungsi ini mencantumkan pemetaan antara grup keamanan AD dan peran DB yang disetel menggunakan `pgadmap_set_mapping` fungsi.

Sintaks

```
pgadmap_read_mapping()
```

Argumen

None

Jenis pengembalian

| Parameter | Deskripsi |
| --- | --- |
| ad_group_id | Pengidentifikasi keamanan yang digunakan untuk mengident ifikasi grup AD secara unik. Nilai dimulai dengan 'S-1-' dan tidak dapat berupa string nol atau kosong. Untuk informasi lebih lanjut, lihat Mengambil Grup SID Direktori Aktif di PowerShell .accounts-role@example.com |

| Parameter | Deskripsi |
| --- | --- |
| db_role | Peran database yang akan dipetakan ke Grup AD yang ditentukan. Nilai tidak bisa null atau string kosong. |
| berat | Berat yang terkait dengan peran database. Peran dengan bobot tertinggi diutamakan ketika pengguna adalah anggota dari beberapa grup. Nilai default berat adalah 1. |
| ad_group | Nama Grup AD. Nilai tidak bisa null atau string kosong. |

Catatan penggunaan

Panggil fungsi ini untuk mencantumkan semua pemetaan yang tersedia antara grup keamanan AD dan peran DB.

Contoh

```
postgres=> select * from pgadmap_read_mapping();

ad_sid                              | pg_role       | weight | ad_grp
------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678      | accounts-role | 10     | accounts-group
(1 row)

(1 row)
```

pgadmap_reset_mapping

Fungsi ini me-reset satu atau semua pemetaan yang diatur menggunakan fungsi. `pgadmap_set_mapping`

Sintaks

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
weight)
```

Argumen

| Parameter | Deskripsi |
| --- | --- |
| ad_group_id | Pengidentifikasi keamanan yang digunakan untuk mengident ifikasi grup AD secara unik. |
| db_role | Peran database yang akan dipetakan ke Grup AD yang ditentukan. |
| berat | Berat yang terkait dengan peran database. |

Jika tidak ada argumen yang diberikan, semua pemetaan peran grup AD ke DB disetel ulang. Entah semua argumen perlu disediakan atau tidak sama sekali.

Jenis pengembalian

None

Catatan penggunaan

Panggil fungsi ini untuk menghapus grup AD tertentu ke pemetaan peran DB atau untuk mengatur ulang semua pemetaan. Fungsi ini hanya dapat dijalankan pada instance DB utama dari cluster DB oleh pengguna yang memiliki `rds_superuser` hak istimewa.

Contoh

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+-------------------
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10          | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);

pgadmap_reset_mapping
(1 row)
```

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+--------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+-------------
 (0 rows)
```

# Migrasi data ke Amazon Aurora dengan kompatibilitas Postgre SQL

Anda memiliki beberapa opsi untuk memigrasikan data dari database yang ada ke cluster DB Amazon Aurora SQL Postgre -Compatible Edition. Opsi migrasi Anda juga bergantung pada basis data asal migrasi Anda dan ukuran data yang Anda migrasikan. Berikut adalah pilihan Anda:

Memigrasi instance RDS untuk Postgre SQL DB menggunakan snapshot

Anda dapat memigrasikan data langsung dari snapshot RDS for Postgre SQL DB ke cluster Aurora Postgre DB. SQL

Memigrasikan instans DB RDS for PostgreSQL menggunakan replika baca Aurora

Anda juga dapat bermigrasi dari instance RDS for Postgre SQL DB dengan membuat replika SQL baca Aurora Postgre dari instance for Postgre DB. RDS SQL Ketika jeda replika antara instance RDS for Postgre SQL DB dan replika SQL baca Aurora Postgre adalah nol, Anda dapat menghentikan replikasi. Pada titik ini, Anda dapat membuat replika baca Aurora sebagai cluster Aurora SQL Postgre DB mandiri untuk membaca dan menulis.

Mengimpor data dari Amazon S3 ke Aurora PostgreSQL

Anda dapat memigrasikan data dengan mengimpornya dari Amazon S3 ke tabel milik klaster Aurora Postgre DB. SQL

Migrasi dari database yang tidak kompatibel dengan SQL Postgre

Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan data dari database yang tidak kompatibel dengan PostgreSQL. Untuk informasi lebih lanjut tentang AWS DMS, lihat Apa itu AWS Database Migration Service? di AWS Database Migration Service Panduan Pengguna.

> **Note**
>
> Mengaktifkan otentikasi Kerberos saat ini tidak didukung di klaster DB Aurora Postgre selama migrasi dari untuk PostgreSQL. RDS SQL Anda dapat mengaktifkan otentikasi Kerberos hanya pada cluster Aurora Postgre DB mandiri. SQL

Untuk daftar Wilayah AWS di mana Aurora tersedia, lihat Amazon Aurora di Referensi Umum AWS.

> **Important**
>
> Jika Anda berencana untuk memigrasikan instans Postgre SQL DB RDS untuk Postgre ke SQL klaster DB Aurora Postgre dalam waktu dekat, kami sangat menyarankan Anda untuk mematikan upgrade versi minor otomatis untuk instans DB di awal fase perencanaan migrasi. Migrasi ke Aurora Postgre SQL mungkin tertunda jika SQL versi RDS for Postgre belum didukung oleh Aurora Postgre. SQL
> Untuk informasi tentang versi Aurora Postgre, lihat SQL Versi mesin untuk Amazon Aurora Postgre. SQL

## Memigrasi snapshot dari instance RDS for Postgre SQL DB ke cluster Aurora Postgre DB SQL

Untuk membuat cluster SQL DB Aurora Postgre, Anda dapat memigrasikan snapshot DB dari instance Postgre DB untuk. RDS SQL Cluster Aurora Postgre SQL DB baru diisi dengan data dari aslinya RDS untuk instance Postgre DB. SQL Untuk informasi tentang membuat snapshot DB, lihat Membuat snapshot DB.

Dalam beberapa kasus, snapshot DB mungkin tidak ada di Wilayah AWS di mana Anda ingin menemukan data Anda. Jika demikian, gunakan RDS konsol Amazon untuk menyalin snapshot DB ke sana Wilayah AWS. Untuk informasi tentang menyalin snapshot DB, lihat Menyalin snapshot DB.

Anda dapat bermigrasi RDS untuk SQL snapshot Postgre yang kompatibel dengan versi Aurora Postgre yang tersedia di versi yang diberikan SQL Wilayah AWS. Misalnya, Anda dapat memigrasikan snapshot dari instans RDS untuk Postgre SQL 11.1 DB ke Aurora Postgre SQL versi 11.4, 11.7, 11.8, atau 11.9 di Wilayah AS Barat (California Utara). Anda dapat bermigrasi RDS untuk SQL snapshot Postgre 10.11 ke Aurora Postgre 10.11, 10.12, 10.13, dan 10.14SQL. Dengan kata lain, SQL snapshot RDS untuk Postgre harus menggunakan versi minor yang sama atau lebih rendah seperti Aurora Postgre. SQL

Anda juga dapat memilih cluster Aurora Postgre SQL DB baru Anda untuk dienkripsi saat istirahat dengan menggunakan AWS KMS key. Opsi ini hanya tersedia untuk snapshot DB yang tidak terenkripsi.

Untuk memigrasikan snapshot Postgre SQL DB ke cluster DB Aurora SQL Postgre, Anda dapat menggunakan RDS AWS Management Console, AWS CLI, atau RDSAPI. Saat Anda menggunakan AWS Management Console, konsol mengambil tindakan yang diperlukan untuk membuat cluster DB dan instance utama.

Konsol

Untuk memigrasikan snapshot Postgre SQL DB dengan menggunakan konsol RDS

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Snapshot.
3. Pada halaman Snapshots, pilih snapshot RDS untuk Postgre SQL yang ingin Anda migrasikan ke cluster Aurora Postgre DB. SQL
4. Pilih Tindakan, lalu pilih Migrasikan snapshot.
5. Tetapkan nilai-nilai berikut pada halaman Migrasikan basis data:

   - Versi mesin DB: Pilih versi mesin DB yang ingin Anda gunakan untuk instans baru yang dimigrasikan.
   - Pengidentifikasi instans DB: Masukkan nama untuk cluster DB yang unik untuk akun Anda di Wilayah AWS yang Anda pilih. Pengidentifikasi ini digunakan di alamat titik akhir untuk instans di klaster DB Anda. Anda mungkin memilih untuk menambahkan beberapa kecerdasan ke nama, seperti memasukkan Wilayah AWS dan mesin DB yang Anda pilih, misalnya**`aurora-cluster1`**.

     Pengenal instans DB memiliki batasan berikut:

  - Pengidentifikasi ini harus berisi 1-63 karakter alfanumerik atau tanda hubung.
  - Karakter pertamanya harus berupa huruf.
  - Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.
  - Itu harus unik untuk semua instans DB per AWS akun, per Wilayah AWS.
- Kelas instans DB: Pilih kelas instans DB yang memiliki penyimpanan dan kapasitas yang diperlukan untuk basis data Anda, misalnya `db.r6g.large`. Volume klaster Aurora secara otomatis bertambah seiring jumlah data dalam basis data Anda meningkat. Jadi, Anda hanya perlu memilih kelas instans DB yang memenuhi persyaratan penyimpanan Anda saat ini. Untuk informasi selengkapnya, lihat Gambaran umum penyimpanan Amazon Aurora.
- Virtual private cloud (VPC): Jika Anda memiliki yang sudah adaVPC, maka Anda dapat menggunakannya VPC dengan cluster Aurora Postgre SQL DB Anda dengan memilih VPC pengenal Anda, misalnya. `vpc-a464d1c1` Untuk informasi tentang membuatVPC, lihatTutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .

  Jika tidak, Anda dapat memilih untuk RDS membuat Amazon VPC untuk Anda dengan memilih Buat baru VPC.
- Grup subnet DB: Jika Anda memiliki grup subnet yang ada, maka Anda dapat menggunakan grup subnet itu dengan cluster Aurora Postgre SQL DB Anda dengan memilih pengidentifikasi grup subnet Anda, misalnya. `gs-subnet-group1`
- Akses publik: Pilih Tidak untuk menentukan bahwa instance di cluster DB Anda hanya dapat diakses oleh sumber daya di dalam AndaVPC. Pilih Ya untuk menentukan bahwa instans di klaster DB Anda dapat diakses oleh sumber daya di jaringan publik.

  > **Note**
  >
  > Klaster DB produksi Anda mungkin tidak perlu berada di subnet publik karena hanya server aplikasi Anda yang memerlukan akses ke klaster DB Anda. Jika klaster DB Anda tidak perlu berada di subnet publik, tetapkan Akses publik menjadi Tidak.
- VPCgrup keamanan: Pilih grup VPC keamanan untuk mengizinkan akses ke database Anda.
- Availability Zone: Pilih Availability Zone untuk meng-host instans utama untuk cluster Aurora SQL Postgre DB Anda. Agar Amazon RDS memilih Availability Zone untuk Anda, pilih Tidak ada preferensi.

- Port database: Masukkan port default yang akan digunakan saat menghubungkan ke instance di cluster Aurora SQL Postgre DB. Default-nya adalah `5432`.

  > **Note**
  >
  > Anda mungkin berada di belakang firewall perusahaan yang tidak mengizinkan akses ke port default seperti port default Postgre, SQL 5432. Dalam hal ini, berikan nilai port yang diizinkan oleh firewall perusahaan Anda. Ingat nilai port itu nanti ketika Anda terhubung ke cluster Aurora SQL Postgre DB.

- Aktifkan Enkripsi: Pilih Aktifkan Enkripsi untuk cluster SQL DB Aurora Postgre baru Anda untuk dienkripsi saat istirahat. Juga pilih KMS kunci sebagai AWS KMS keynilai.
- Peningkatan versi minor otomatis: Pilih Aktifkan peningkatan versi minor otomatis untuk mengaktifkan cluster Aurora Postgre SQL DB Anda menerima peningkatan versi mesin Postgre SQL DB minor secara otomatis saat tersedia.

  Opsi peningkatan versi minor Otomatis hanya berlaku untuk peningkatan ke versi mesin SQL minor Postgre untuk cluster DB Aurora Postgre Anda. SQL Opsi ini tidak berlaku untuk patch biasa yang diterapkan untuk menjaga stabilitas sistem.

6. Pilih Migrasikan untuk memigrasikan snapshot DB Anda.
7. Pilih Basis data untuk melihat klaster DB baru. Pilih klaster DB baru untuk memantau progres migrasi. Ketika migrasi selesai, Status untuk klaster adalah Tersedia. Pada tab Konektivitas & keamanan, Anda dapat menemukan titik akhir klaster yang digunakan untuk menghubungkan ke instans penulis primer klaster DB. Untuk informasi lebih lanjut tentang menghubungkan ke cluster SQL DB Aurora Postgre, lihat. Menghubungkan ke klaster DB Amazon Aurora

AWS CLI

Menggunakan AWS CLI untuk memigrasikan snapshot Postgre SQL DB RDS untuk Postgre ke Aurora Postgre melibatkan dua terpisah SQL AWS CLI perintah. Pertama, Anda menggunakan `restore-db-cluster-from-snapshot` AWS CLI perintah membuat cluster Aurora SQL Postgre DB baru. Anda kemudian menggunakan perintah `create-db-instance` untuk membuat instans DB primer di klaster baru untuk menyelesaikan migrasi. Prosedur berikut membuat cluster Aurora Postgre SQL DB dengan instance DB primer yang memiliki konfigurasi yang sama dengan instans DB yang digunakan untuk membuat snapshot.

Untuk memigrasikan snapshot Postgre SQL DB ke cluster Aurora Postgre DB RDS SQL

1. Gunakan describe-db-snapshotsperintah untuk mendapatkan informasi tentang snapshot DB yang ingin Anda migrasikan. Anda dapat menentukan parameter `--db-instance-identifier` atau `--db-snapshot-identifier` dalam perintah. Jika Anda tidak menentukan salah satu parameter ini, Anda akan mendapatkan semua snapshot.

   ```
   aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
   ```

2. Perintah ini akan menghasilkan semua detail konfigurasi untuk setiap snapshot yang dibuat dari instans DB yang ditentukan. Dalam output, temukan snapshot yang ingin Anda migrasikan dan temukan Amazon Resource Name ()ARN. Untuk mempelajari lebih lanjut tentang Amazon RDSARNs, lihat Amazon Relational Database Service (RDSAmazon). Sebuah ARN terlihat mirip dengan output berikut.

   ```
   “DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
   ```

   Juga dalam output Anda dapat menemukan detail konfigurasi untuk instance RDS for Postgre SQL DB, seperti versi mesin, penyimpanan yang dialokasikan, apakah instans DB dienkripsi atau tidak, dan sebagainya.

3. Gunakan perintah restore-db-cluster-from-snapshot untuk memulai migrasi. Tentukan parameter berikut:

   - `--db-cluster-identifier`— Nama yang ingin Anda berikan ke cluster Aurora SQL Postgre DB. Klaster Aurora DB ini adalah target untuk migrasi snapshot DB Anda.
   - `--snapshot-identifier`— Nama Sumber Daya Amazon (ARN) dari snapshot DB untuk bermigrasi.
   - `--engine` – Tentukan `aurora-postgresql` untuk mesin klaster Aurora DB.
   - `--kms-key-id`— Parameter opsional ini memungkinkan Anda membuat cluster Aurora Postgre SQL DB terenkripsi dari snapshot DB yang tidak terenkripsi. Hal ini juga memungkinkan Anda memilih kunci enkripsi untuk klaster DB yang berbeda dari kunci yang digunakan untuk snapshot DB.

> **Note**
>
> Anda tidak dapat membuat cluster Aurora SQL Postgre DB yang tidak terenkripsi dari snapshot DB terenkripsi.

Tanpa `--kms-key-id` parameter yang ditentukan seperti yang ditunjukkan berikut, restore-db-cluster-from-snapshot AWS CLI perintah membuat cluster SQL DB Aurora Postgre kosong yang dienkripsi menggunakan kunci yang sama dengan snapshot DB atau tidak dienkripsi jika snapshot DB sumber tidak dienkripsi.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name \
    --engine aurora-postgresql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name ^
    --engine aurora-postgresql
```

4. Perintah mengembalikan detail tentang cluster Aurora Postgre SQL DB yang sedang dibuat untuk migrasi. Anda dapat memeriksa status cluster Aurora Postgre SQL DB dengan menggunakan describe-db-clusters AWS CLI perintah.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. Ketika cluster DB menjadi “tersedia”, Anda menggunakan create-db-instanceperintah untuk mengisi cluster Aurora SQL Postgre DB dengan instans DB berdasarkan snapshot Amazon DB Anda. RDS Tentukan parameter berikut:

   - `--db-cluster-identifier`— Nama cluster Aurora Postgre SQL DB baru yang Anda buat pada langkah sebelumnya.

- `--db-instance-identifier` – Nama yang ingin Anda berikan ke instans DB. Instance ini menjadi node utama di cluster Aurora SQL Postgre DB Anda.
- `----db-instance-class` – Tentukan kelas instans DB yang akan digunakan. Pilih dari antara kelas instans DB yang didukung oleh SQL versi Aurora Postgre tempat Anda bermigrasi. Untuk informasi selengkapnya, silakan lihat Jenis kelas instans DB dan Mesin DB yang didukung untuk kelas instans DB.
- `--engine` – Tentukan `aurora-postgresql` untuk instans DB.

Anda juga dapat membuat instans DB dengan konfigurasi yang berbeda dari snapshot DB sumber, dengan meneruskan opsi yang sesuai di `create-db-instance` AWS CLI perintah. Untuk informasi lebih lanjut, lihat create-db-instanceperintah.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier cluster-name \
    --db-instance-identifier --db-instance-class db.instance.class \
    --engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier cluster-name ^
    --db-instance-identifier --db-instance-class db.instance.class ^
    --engine aurora-postgresql
```

Ketika proses migrasi selesai, cluster Aurora SQL Postgre memiliki instans DB primer yang terisi.

# Memigrasikan data dari instans DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL menggunakan replika baca Aurora

Anda dapat menggunakan instans DB RDS for PostgreSQL sebagai dasar untuk klaster DB Aurora PostgreSQL baru dengan menggunakan replika baca Aurora untuk proses migrasi. Opsi replika baca Aurora hanya tersedia untuk migrasi dalam akun Wilayah AWS dan akun yang sama, dan hanya tersedia jika Wilayah menawarkan versi Aurora PostgreSQL yang kompatibel untuk RDS Anda untuk instans PostgreSQL DB. Dengan kompatibel, yang kami maksud adalah bahwa versi Aurora PostgreSQL sama dengan versi RDS for PostgreSQL, atau versi minor yang lebih tinggi dalam kelompok versi mayor yang sama.

Misalnya, untuk menggunakan teknik ini dalam memigrasikan instans DB RDS for PostgreSQL 11.14, Wilayahnya harus menawarkan Aurora PostgreSQL versi 11.14 atau versi minor yang lebih tinggi dalam kelompok PostgreSQL versi 11.

Topik


- Gambaran umum tentang memigrasikan data menggunakan replika baca Aurora
- Bersiap untuk memigrasikan data dengan menggunakan replika baca Aurora
- Membuat replika baca Aurora
- Mempromosikan replika baca Aurora


## Gambaran umum tentang memigrasikan data menggunakan replika baca Aurora

Migrasi dari instans DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL adalah prosedur multilangkah. Pertama, buat replika baca Aurora dari instans DB RDS for PostgreSQL sumber Anda. Tindakan ini adalah awal proses replikasi dari instans DB RDS for PostgreSQL Anda ke klaster DB tujuan khusus yang dikenal sebagai klaster Replika. Klaster Replika hanya terdiri dari replika baca Aurora (instans pembaca).

> **Note**
>
> Diperlukan beberapa jam per terabyte data agar migrasi selesai.

### Mempromosikan Replika PostgreSQL Aurora

Setelah Anda membuat klaster DB PostgreSQL Aurora, ikuti langkah-langkah berikut untuk mempromosikan Replika Aurora:

1. Hentikan semua beban kerja penulisan database pada sumber RDS untuk instance PostgreSQL DB.
2. Dapatkan arus `WAL LSN` dari sumber RDS untuk instance PostgreSQL DB:

```
SELECT pg_current_wal_lsn();
pg_current_wal_lsn
--------------------
0/F0000318
(1 row)
```

3. Pada cluster Aurora PostgreSQL Replica, periksa apakah LSN yang diputar ulang lebih besar dari LSN dari langkah 2:

```
SELECT pg_last_wal_replay_lsn();
pg_last_wal_replay_lsn
------------------------
0/F0000400
(1 row)
```

   Atau, Anda dapat menggunakan kueri berikut pada sumber RDS untuk PostgreSQL DB instance:

```
SELECT restart_lsn FROM pg_replication_slots;
```

4. Promosikan cluster Aurora PostgreSQL Replica.

   Ketika replikasi berhenti, cluster replika dipromosikan ke cluster Aurora PostgreSQL DB mandiri, dan pembaca dipromosikan ke instance penulis untuk cluster. Pada titik ini, Anda dapat menambahkan instance ke cluster Aurora PostgreSQL DB untuk mengukurnya sesuai dengan kasus penggunaan Anda. Jika Anda tidak lagi memerlukan RDS asli untuk PostgreSQL DB instance, Anda dapat menghapusnya.

Anda tidak dapat membuat replika baca Aurora jika instans DB RDS for PostgreSQL Anda sudah memiliki replika baca Aurora atau jika memiliki replika baca lintas Wilayah.

## Bersiap untuk memigrasikan data dengan menggunakan replika baca Aurora

Selama proses migrasi menggunakan replika baca Aurora, pembaruan yang dilakukan pada instans DB RDS for PostgreSQL sumber akan direplikasi secara asinkron ke replika baca Aurora dari klaster Replika. Proses ini menggunakan fungsionalitas replikasi aliran native PostgreSQL yang menyimpan segmen write-ahead log (WAL) pada instans sumber. Sebelum memulai proses migrasi ini, pastikan bahwa instans Anda memiliki kapasitas penyimpanan yang memadai dengan memeriksa nilai metrik yang tercantum dalam tabel.

| Metrik | Deskripsi |
| --- | --- |
| `FreeStorageSpace` | Ruang penyimpanan yang tersedia.<br><br>Unit: Byte |
| `OldestReplicationSlotLag` | Ukuran lag untuk data WAL di replika yang mengalami lag paling lama.<br><br>Unit: Megabyte |
| `RDSToAuroraPostgreSQLReplicaLag` | Jumlah waktu dalam hitungan detik ketika klaster DB Aurora PostgreSQL mengalami lag di belakang instans DB RDS sumber. |
| `TransactionLogsDiskUsage` | Ruang disk yang digunakan oleh log transaksi.<br><br>Unit: Megabyte |

Untuk informasi selengkapnya tentang pemantauan instans RDS Anda, lihat Pemantauan dalam Panduan Pengguna Amazon RDS.

## Membuat replika baca Aurora

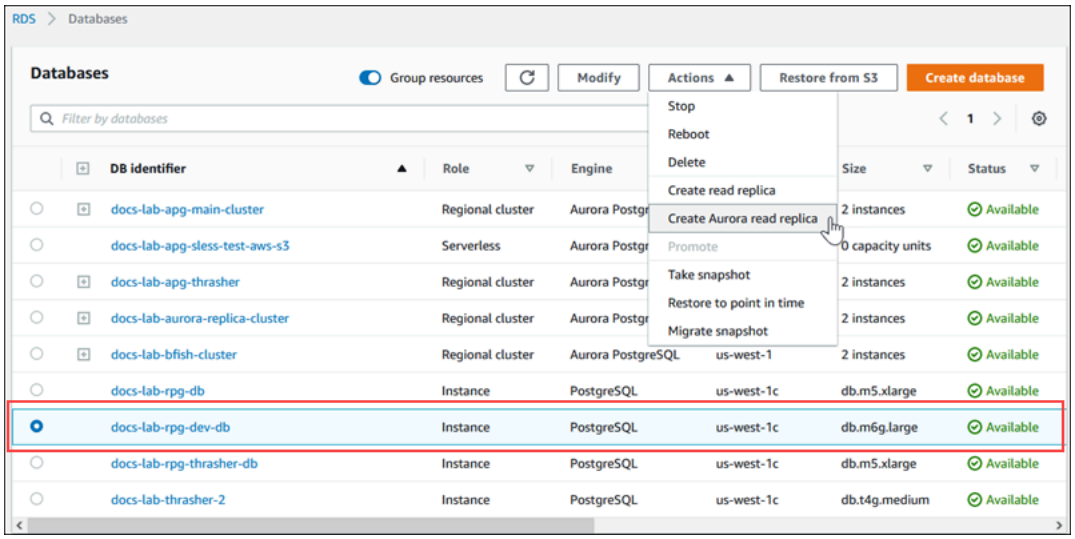
Anda dapat membuat replika baca Aurora untuk RDS untuk instance PostgreSQL DB dengan menggunakan atau. AWS Management Console AWS CLI Opsi untuk membuat replika baca Aurora menggunakan hanya tersedia jika Wilayah AWS menawarkan versi AWS Management

Console Aurora PostgreSQL yang kompatibel. Artinya, opsi ini hanya tersedia jika ada versi Aurora PostgreSQL yang sama dengan versi RDS for PostgreSQL atau versi minor yang lebih tinggi dalam kelompok versi mayor yang sama.

Konsol

Untuk membuat replika baca Aurora dari instans DB PostgreSQL sumber

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih instans DB RDS for PostgreSQL yang ingin Anda gunakan sebagai sumber untuk replika baca Aurora Anda. Untuk Tindakan, pilih Buat replika baca Aurora. Jika pilihan ini tidak ditampilkan, itu berarti versi Aurora PostgreSQL yang kompatibel tidak tersedia di Wilayah tersebut.




4. Pada halaman pengaturan "Buat replika baca Aurora", Anda perlu mengonfigurasi properti untuk klaster DB Aurora PostgreSQL seperti yang ditunjukkan pada tabel berikut. Klaster DB Replika dibuat dari snapshot instans DB sumber menggunakan nama pengguna dan kata sandi 'master' yang sama dengan sumbernya, sehingga Anda tidak dapat mengubahnya saat ini.

| Opsi | Deskripsi |
| --- | --- |
| Kelas instans DB | Pilih kelas instans DB yang memenuhi persyaratan pemrosesan dan memori untuk instans primer dalam |

| Opsi | Deskripsi |
| --- | --- |
| | klaster DB. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB. |
| Deployment Multi-AZ | Tidak tersedia selama migrasi |
| Pengidentifikasi instans DB | Masukkan nama yang ingin Anda berikan ke instans DB. Pengidentifikasi ini digunakan dalam alamat titik akhir untuk instans primer klaster DB baru.<br><br>Pengidentifikasi instans DB memiliki batasan berikut:<br><br>• Pengidentifikasi ini harus berisi 1-63 karakter alfanumerik atau tanda hubung.<br>• Karakter pertamanya harus berupa huruf.<br>• Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.<br>• Ini harus unik untuk semua instans DB untuk setiap AWS akun, untuk masing-masing Wilayah AWS. |
| Cloud Privat Virtual (VPC) | Pilih VPC untuk meng-host klaster DB. Pilih Buat VPC baru agar Amazon RDS membuatkan VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. |
| Grup subnet DB | Pilih grup subnet DB yang akan digunakan untuk klaster DB. Pilih Buat Grup Subnet DB baru agar Amazon RDS membuatkan grup subnet DB untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. |

| Opsi | Deskripsi |
| --- | --- |
| Aksesibilitas publik | Pilih Ya untuk memberi klaster DB alamat IP publik. Atau, pilih Tidak. Instans di klaster DB Anda dapat menjadi perpaduan dari instans DB publik dan privat. Untuk informasi selengkapnya tentang menyembun yikan instans dari akses publik, lihat Menyembunyikan klaster DB dalam VPC dari internet. |
| Zona ketersediaan | Tentukan apakah Anda ingin menentukan Zona Ketersediaan tertentu. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan. |
| Grup keamanan VPC | Pilih satu atau beberapa grup keamanan VPC untuk mengamankan akses jaringan ke klaster DB. Pilih Buat grup keamanan VPC baru agar Amazon RDS membuatkan grup keamanan VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB. |
| Port basis data | Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Klaster DB Aurora PostgreSQL akan ditetapkan secara default ke port default PostgreSQL, 5432. Firewall di beberapa perusahaan memblokir koneksi ke port ini. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru. |
| Grup parameter DB | Pilih grup parameter DB untuk klaster DB Aurora PostgreSQL. Aurora memiliki grup parameter DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter DB, lihat . |

| Opsi | Deskripsi |
| --- | --- |
| Grup parameter klaster DB | Pilih grup parameter klaster DB untuk klaster DB Aurora PostgreSQL. Aurora memiliki grup parameter klaster DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter klaster DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat . |
| Enkripsi | Pilih Aktifkan Enkripsi untuk klaster DB Aurora baru Anda agar dienkripsi saat diam. Jika Anda memilih Aktifkan enkripsi, pilih juga kunci KMS sebagai nilai AWS KMS key. |
| Prioritas | Pilih prioritas failover untuk klaster DB. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora. |
| Periode retensi cadangan | Pilih durasi waktu, dalam kisaran 1–35 hari, saat Aurora mempertahankan salinan cadangan basis data. Salinan cadangan dapat digunakan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua. |
| Pemantauan yang ditingkatkan | Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan. |

| Opsi | Deskripsi |
| --- | --- |
| Peran Pemantauan | Hanya tersedia jika Anda memilih Aktifkan pemantauan yang ditingkatkan. Peran AWS Identity and Access Management (IAM) yang digunakan untuk Pemantauan yang Ditingkatkan. Untuk informasi selengkapnya, lihat Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan. |
| Granularitas | Hanya tersedia jika Anda memilih Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda. |
| Peningkatan versi minor otomatis | Pilih Ya agar klaster DB Aurora PostgreSQL Anda dapat menerima peningkatan versi minor engine DB PostgreSQL secara otomatis saat tersedia.<br><br>Opsi Peningkatan versi minor otomatis hanya berlaku untuk peningkatan versi minor engine PostgreSQL untuk klaster DB Aurora PostgreSQL Anda. Opsi ini tidak berlaku untuk patch biasa yang diterapkan untuk menjaga stabilitas sistem. |
| Jendela pemeliharaan | Pilih rentang waktu mingguan untuk melakukan pemeliharaan sistem. |

5. Pilih Buat replika baca.

AWS CLI

Untuk membuat replika baca Aurora dari sumber RDS untuk instance PostgreSQL DB dengan menggunakan AWS CLI, pertama-tama Anda menggunakan perintah CLI create-db-clusteruntuk membuat cluster DB Aurora kosong. Setelah klaster DB ada, Anda kemudian menggunakan perintah create-db-instance untuk membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat di klaster DB Aurora. Dalam hal ini, instans tersebut awalnya dibuat sebagai replika baca Aurora pada instans DB RDS for PostgreSQL Anda. Saat prosesnya selesai, instans DB RDS for PostgreSQL Anda telah dimigrasikan secara efektif ke klaster DB Aurora PostgreSQL.

Anda tidak perlu menentukan akun pengguna primer (biasanya, `postgres`), kata sandinya, atau nama basis data. Replika baca Aurora memperolehnya secara otomatis dari sumber RDS untuk instance PostgreSQL DB yang Anda identifikasi saat Anda menjalankan perintah. AWS CLI

Anda perlu menentukan versi mesin yang akan digunakan untuk klaster DB dan instans DB Aurora PostgreSQL. Versi yang Anda tentukan harus cocok dengan instans DB RDS for PostgreSQL sumber. Jika instans DB RDS for PostgreSQL sumber dienkripsi, Anda juga perlu menentukan enkripsi untuk instans primer klaster DB Aurora PostgreSQL. Migrasi instans terenkripsi ke klaster DB Aurora yang tidak terenkripsi tidak didukung.

Contoh berikut membuat klaster DB Aurora PostgreSQL bernama `my-new-aurora-cluster` yang akan menggunakan instans DB RDS sumber yang tidak terenkripsi. Pertama-tama, Anda membuat klaster DB Aurora PostgreSQL dengan memanggil perintah CLI `create-db-cluster`. Contoh ini menunjukkan cara menggunakan parameter `--storage-encrypted` opsional untuk menentukan bahwa klaster DB harus dienkripsi. Karena sumber DB tidak dienkripsi, `--kms-key-id` akan digunakan untuk menentukan kunci yang akan digunakan. Untuk informasi selengkapnya tentang parameter wajib dan opsional, lihat contoh berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster  \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Untuk Windows:

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
```

```
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Dalam daftar berikut, Anda dapat menemukan informasi selengkapnya tentang beberapa opsi yang ditampilkan dalam contoh. Kecuali jika ditentukan lain, parameter ini wajib ada.

- `--db-cluster-identifier` – Anda harus memberikan nama untuk klaster DB Aurora PostgreSQL baru Anda.
- `--db-subnet-group-name` – Buat klaster DB Aurora PostgreSQL Anda di subnet DB yang sama dengan instans DB sumber.
- `--vpc-security-group-ids` – Tentukan grup keamanan untuk klaster DB Aurora PostgreSQL Anda.
- `--engine-version` – Tentukan versi yang akan digunakan untuk klaster DB Aurora PostgreSQL. Ini harus sama dengan atau versi minor yang lebih tinggi dari versi yang digunakan oleh RDS sumber Anda untuk instance PostgreSQL DB.
- `--replication-source-identifier` – Identifikasi instans DB RDS for PostgreSQL menggunakan Amazon Resource Name (ARN). Untuk informasi selengkapnya tentang Amazon RDS ARNs, lihat Amazon Relational Database Service (Amazon RDS) di. cluster Referensi Umum AWSDB Anda.
- `--storage-encrypted` – Opsional. Gunakan hanya jika diperlukan untuk menentukan enkripsi sebagai berikut:
  - Gunakan parameter ini ketika instans DB sumber memiliki penyimpanan terenkripsi. Panggilan ke `create-db-cluster` akan gagal jika Anda tidak menggunakan parameter ini dengan instans DB sumber yang memiliki penyimpanan terenkripsi. Jika Anda ingin menggunakan kunci untuk klaster DB Aurora PostgreSQL yang berbeda dari kunci yang digunakan oleh instans DB sumber, Anda juga perlu menentukan `--kms-key-id`.
  - Gunakan jika penyimpanan instans DB sumber tidak terenkripsi, tetapi Anda ingin klaster DB Aurora PostgreSQL menggunakan enkripsi. Jika demikian, Anda juga perlu mengidentifikasi kunci enkripsi yang akan digunakan dengan parameter `--kms-key-id`.
- `--kms-key-id` – Opsional. Saat digunakan, Anda dapat menentukan kunci yang akan digunakan untuk enkripsi penyimpanan (`--storage-encrypted`) dengan menggunakan ARN kunci, ID, ARN alias, atau nama aliasnya. Parameter ini diperlukan hanya untuk situasi-situasi berikut:

- Untuk memilih kunci untuk klaster DB Aurora PostgreSQL yang berbeda dari yang digunakan oleh instans DB sumber.
- Untuk membuat klaster terenkripsi dari sumber yang tidak terenkripsi. Dalam hal ini, Anda perlu menentukan kunci yang harus digunakan Aurora PostgreSQL untuk enkripsi.

Setelah membuat klaster DB Aurora PostgreSQL, Anda kemudian membuat instans primer dengan menggunakan perintah CLI `create-db-instance`, seperti yang ditunjukkan berikut ini:

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

Dalam daftar berikut, Anda dapat menemukan informasi selengkapnya tentang beberapa opsi yang ditampilkan dalam contoh.

- `--db-cluster-identifier` – Tentukan nama klaster DB Aurora PostgreSQL yang Anda buat dengan perintah `create-db-instance` di langkah sebelumnya.
- `--db-instance-class` – Nama kelas instans DB yang akan digunakan untuk instans primer Anda, seperti `db.r4.xlarge`, `db.t4g.medium`, `db.x2g.16xlarge`, dan sebagainya. Untuk daftar kelas instans DB yang tersedia, lihat Jenis kelas instans DB.
- `--db-instance-identifier` – Tentukan nama yang akan diberikan untuk instans primer Anda.
- `--engine aurora-postgresql` – Tentukan `aurora-postgresql` untuk mesin.

API RDS

Untuk membuat replika baca Aurora dari instans DB RDS for PostgreSQL sumber, pertama-tama gunakan operasi API RDS `CreateDBCluster` untuk membuat klaster DB Aurora baru untuk replika baca Aurora yang dibuat dari instans DB RDS for PostgreSQL sumber Anda. Ketika klaster DB Aurora PostgreSQL tersedia, Anda kemudian menggunakan `CreateDBInstance` untuk membuat instans primer untuk klaster DB Aurora.

Anda tidak perlu menentukan akun pengguna primer (biasanya, `postgres`), kata sandinya, atau nama basis data. Replika baca Aurora akan memperolehnya secara otomatis dari instans DB RDS for PostgreSQL sumber yang ditentukan dengan `ReplicationSourceIdentifier`.

Anda perlu menentukan versi mesin yang akan digunakan untuk klaster DB dan instans DB Aurora PostgreSQL. Versi yang Anda tentukan harus cocok dengan instans DB RDS for PostgreSQL sumber. Jika instans DB RDS for PostgreSQL sumber dienkripsi, Anda juga perlu menentukan enkripsi untuk instans primer klaster DB Aurora PostgreSQL. Migrasi instans terenkripsi ke klaster DB Aurora yang tidak terenkripsi tidak didukung.

Untuk membuat klaster DB Aurora untuk replika baca Aurora, gunakan operasi API RDS `CreateDBCluster` dengan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB yang akan dibuat.
- `DBSubnetGroupName` – Nama grup subnet DB yang akan dikaitkan dengan klaster DB ini.
- `Engine=aurora-postgresql` – Nama mesin yang akan digunakan.
- `ReplicationSourceIdentifier` – Amazon Resource Name (ARN) untuk instans DB PostgreSQL sumber. Untuk informasi selengkapnya tentang Amazon RDS ARNs, lihat Amazon Relational Database Service (Amazon RDS) di bagian.Referensi Umum Amazon Web Services Jika `ReplicationSourceIdentifier` mengidentifikasi sumber terenkripsi, Amazon RDS akan menggunakan kunci KMS default Anda kecuali jika Anda menentukan kunci yang berbeda menggunakan opsi `KmsKeyId`.
- `VpcSecurityGroupIds`— Daftar grup keamanan Amazon EC2 VPC untuk dikaitkan dengan cluster DB ini.
- `StorageEncrypted` – Menunjukkan bahwa klaster DB dienkripsi. Jika Anda menggunakan parameter ini tanpa menentukan `ReplicationSourceIdentifier`, Amazon RDS akan menggunakan kunci KMS default Anda.

- `KmsKeyId` – Kunci untuk klaster terenkripsi. Saat digunakan, Anda dapat menentukan kunci yang akan digunakan untuk enkripsi penyimpanan dengan menggunakan ARN kunci, ID, ARN alias, atau nama aliasnya.

Untuk informasi selengkapnya, lihat `CreateDBCluster` dalam Referensi API Amazon RDS.

Setelah klaster DB Aurora tersedia, Anda dapat membuat instans primer untuk klaster tersebut dengan menggunakan operasi API RDS `CreateDBInstance` dengan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB Anda.
- `DBInstanceClass` – Nama kelas instans DB yang akan digunakan untuk instans primer Anda.
- `DBInstanceIdentifier` – Nama instans primer Anda.
- `Engine=aurora-postgresql` – Nama mesin yang akan digunakan.

Untuk informasi selengkapnya, lihat `CreateDBInstance` dalam Referensi API Amazon RDS.

## Mempromosikan replika baca Aurora

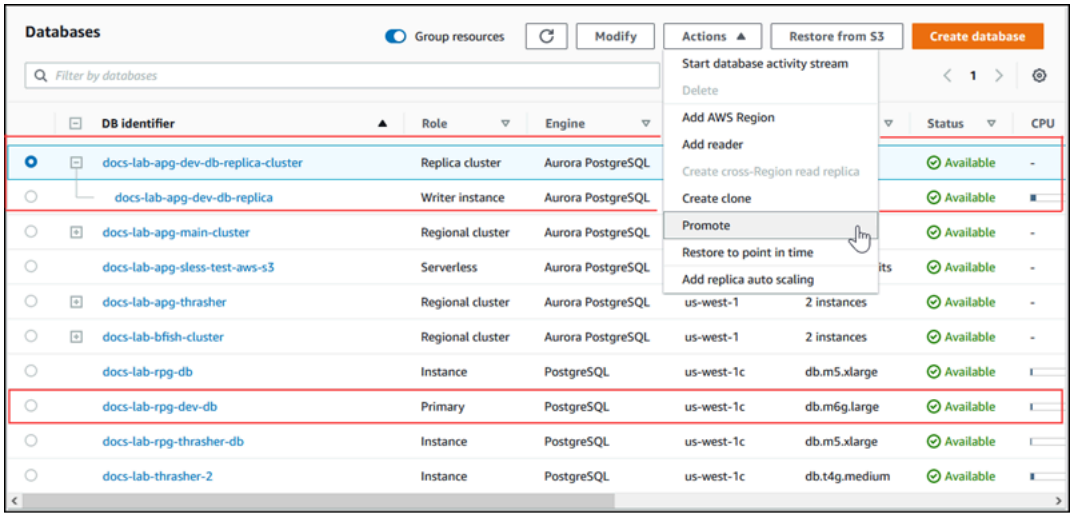
Migrasi ke Aurora PostgreSQL belum selesai sampai Anda mempromosikan klaster Replika, jadi jangan hapus instans DB RDS for PostgreSQL sumber.

Sebelum mempromosikan klaster Replika, pastikan bahwa instans DB RDS for PostgreSQL tidak memiliki transaksi dalam proses atau aktivitas lain yang menulis ke basis data. Saat lag replika pada replika baca Aurora mencapai nol (0), Anda dapat mempromosikan klaster Replika. Untuk informasi selengkapnya tentang pemantauan lag replika, lihat Memantau replikasi Aurora PostgreSQL dan Metrik tingkat instans untuk Amazon Aurora.

Konsol

Untuk mempromosikan replika baca Aurora ke klaster DB Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster Replika.




4. Untuk Tindakan, pilih Promosikan. Hal ini mungkin memerlukan waktu beberapa menit dan dapat menyebabkan waktu henti.

Ketika prosesnya selesai, klaster Aurora Replika adalah klaster DB Aurora PostgreSQL Regional, dengan instans Penulis yang berisi data dari instans DB RDS for PostgreSQL.

AWS CLI

Untuk mempromosikan replika baca Aurora ke cluster DB yang berdiri sendiri, gunakan perintah. promote-read-replica-db-cluster AWS CLI

Example

Untuk Linux, macOS, atau Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Untuk Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

API RDS

Untuk mempromosikan replika baca Aurora ke cluster DB yang berdiri sendiri, gunakan operasi API RDS. PromoteReadReplicaDBCluster

Setelah mempromosikan klaster Replika, Anda dapat mengonfirmasi bahwa promosinya telah selesai dengan memeriksa log peristiwa sebagai berikut.

Untuk mengonfirmasi bahwa klaster Replika Aurora telah dipromosikan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Peristiwa.
3. Pada halaman Peristiwa, temukan nama klaster Anda di daftar Sumber. Setiap peristiwa memiliki sumber, jenis, waktu, dan pesan. Anda dapat melihat semua peristiwa yang terjadi di Wilayah AWS untuk akun Anda. Promosi yang sukses menghasilkan pesan berikut.

```
Promoted Read Replica cluster to a stand-alone database cluster.
```

Setelah promosi selesai, instans DB RDS for PostgreSQL sumber dan klaster DB Aurora PostgreSQL akan menjadi tidak tertaut. Anda dapat mengarahkan aplikasi klien Anda ke titik akhir untuk replika baca Aurora. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Koneksi titik akhir Amazon Aurora. Pada tahap ini, Anda dapat menghapus instans DB dengan aman.

# Mengoptimalkan kinerja kueri di Aurora PostgreSQL

Mengoptimalkan kinerja kueri sangat penting karena membantu database berjalan lebih cepat dan lebih efisien saat menggunakan lebih sedikit sumber daya, yang menghasilkan pengalaman pengguna yang lebih baik dan biaya operasi yang lebih rendah. Amazon Aurora PostgreSQL menyediakan beberapa fitur untuk membantu mengoptimalkan kinerja kueri untuk beban kerja PostgreSQL.

Topik

- Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads
- Mengoptimalkan subkueri berkorelasi di Aurora PostgreSQL
- Meningkatkan kinerja kueri menggunakan gabungan adaptif

# Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads

Anda dapat mencapai pemrosesan kueri yang lebih cepat untuk Aurora PostgreSQL dengan Aurora Optimized Reads. Instans DB Aurora PostgreSQL yang menggunakan Aurora Optimized Reads memberikan latensi kueri hingga 8x lebih baik dan penghematan biaya hingga 30% untuk aplikasi dengan set data besar, yang melebihi kapasitas memori instans DB.

Topik


- Gambaran umum Aurora Optimized Reads di PostgreSQL
- Menggunakan Aurora Optimized Reads
- Kasus penggunaan untuk Aurora Optimized Reads
- Memantau instans DB yang menggunakan Aurora Optimized Reads
- Praktik terbaik untuk Aurora Optimized Reads


## Gambaran umum Aurora Optimized Reads di PostgreSQL

Bacaan yang Dioptimalkan Aurora tersedia secara default saat Anda membuat cluster DB yang menggunakan instans R6GD berbasis Graviton dan R6id berbasis Intel dengan penyimpanan memori express () non-volatile. NVMe Fitur ini tersedia dari versi PostgreSQL berikut:

- 16.1 dan semua versi yang lebih tinggi
- 15.4 dan versi yang lebih tinggi
- 14.9 dan versi yang lebih tinggi

Aurora Optimized Reads mendukung dua kemampuan: cache berjenjang dan objek sementara.

Cache berjenjang yang didukung Optimized Reads - Menggunakan cache berjenjang, Anda dapat memperluas kapasitas caching instans DB hingga 5x memori instans. Kemampuan ini secara otomatis mempertahankan cache untuk menyimpan data terbaru yang konsisten secara transaksional, sehingga membebaskan aplikasi dari overhead dalam mengelola keterkinian data dalam solusi caching berbasis hasil set eksternal. Kemampuan ini menawarkan latensi hingga 8x lebih baik untuk kueri yang sebelumnya mengambil data dari penyimpanan Aurora.

Di Aurora, nilai untuk `shared_buffers` dalam kelompok parameter default biasanya diatur ke sekitar 75% dari memori yang tersedia. Namun, untuk jenis instans r6gd dan r6id, Aurora akan

mengurangi `shared_buffers` ruang sebesar 4,5% untuk meng-host metadata untuk cache Bacaan yang Dioptimalkan.

Objek sementara yang diaktifkan Bacaan yang dioptimalkan - Menggunakan objek sementara, Anda dapat mencapai pemrosesan kueri yang lebih cepat dengan menempatkan file sementara yang dihasilkan oleh PostgreSQL di penyimpanan lokal. NVMe Kemampuan ini mengurangi lalu lintas ke Elastic Block Storage (EBS) melalui jaringan. Ini menawarkan latensi dan throughput hingga 2x lebih baik untuk kueri lanjutan yang mengurutkan, menggabungkan, atau menggabungkan volume besar data yang tidak sesuai dengan kapasitas memori yang tersedia pada instans DB.

Pada cluster Aurora I/O-Optimized, Optimized Reads menggunakan cache berjenjang dan objek sementara pada penyimpanan. NVMe Dengan kemampuan cache berjenjang yang didukung Optimized Reads, Aurora mengalokasikan 2x memori instans untuk objek sementara, sekitar 10% penyimpanan untuk operasi internal, dan penyimpanan yang tersisa sebagai cache berjenjang. Pada klaster Standar Aurora, Optimized Reads hanya menggunakan objek sementara.

Cluster Aurora I/O-Optimized memungkinkan Anda mengubah ukuran ruang yang dialokasikan untuk objek sementara yang diaktifkan Bacaan yang Dioptimalkan menggunakan parameter dinamis pada tingkat instans. `aurora_temp_space_size` Fitur pengubahan ukuran ini tersedia dari versi PostgreSQL berikut:

- 16.8 dan semua versi yang lebih tinggi
- 15.12 dan versi 15 yang lebih tinggi
- 14.17 dan versi 14 yang lebih tinggi

Dengan parameter ini, Anda dapat mengubah ukuran kapasitas dari 2x hingga 6x memori instans tanpa memerlukan restart mesin database. Saat Anda memperluas ruang objek sementara, perubahan akan segera berlaku, terlepas dari beban kerja bersamaan. Namun, ketika Anda mengurangi ruang, penyesuaian selesai hanya setelah ada cukup ruang yang tidak terpakai di objek sementara untuk mengakomodasi permintaan ukuran baru. Setelah Anda mengubah ukuran objek sementara yang diaktifkan Bacaan yang Dioptimalkan, cache berjenjang secara otomatis menyesuaikan untuk menggunakan ruang yang tersedia.

| Engine | Konfigurasi penyimpanan klaster | Objek sementara yang didukung Optimized Reads | Cache berjenjang yang didukung Optimized Reads | Versi yang didukung |
|---|---|---|---|---|
| yang kompatibel dengan Aurora PostgreSQL | Standar | Ya | Tidak | Aurora PostgreSQL versi 16.1 dan semua versi yang lebih tinggi, 15.4 dan lebih tinggi, versi 14.9 dan lebih tinggi |
| | I/O Dioptimalkan | Ya | Ya | |

> **Note**
>
> Peralihan antara cluster yang dioptimalkan oleh iOS dan standar pada kelas instans DB NVMe berbasis menyebabkan restart mesin database langsung.

Di Aurora PostgreSQL, gunakan `temp_tablespaces` parameter untuk mengkonfigurasi ruang tabel tempat objek sementara disimpan.

Untuk memeriksa apakah objek sementara dikonfigurasi, gunakan perintah berikut:

```
postgres=> show temp_tablespaces;
temp_tablespaces
---------------------
aurora_temp_tablespace
(1 row)
```

`aurora_temp_tablespace`Ini adalah tablespace yang dikonfigurasi oleh Aurora yang menunjuk ke penyimpanan lokal NVMe . Anda tidak dapat mengubah parameter ini atau beralih kembali ke penyimpanan Amazon EBS.

Untuk memeriksa apakah cache baca yang dioptimalkan diaktifkan, gunakan perintah berikut:

```
postgres=> show shared_preload_libraries;
```

```
                shared_preload_libraries
-----------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

## Menggunakan Aurora Optimized Reads

Saat Anda menyediakan instans Aurora PostgreSQL DB dengan salah satu instans DB NVMe berbasis, instans DB secara otomatis menggunakan Aurora Optimized Reads.

Untuk mengaktifkan Aurora Optimized Reads, lakukan salah satu hal berikut ini:

- Buat cluster DB PostgreSQL Aurora menggunakan salah satu kelas instans DB berbasis. NVMe Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.
- Ubah cluster Aurora PostgreSQL DB yang ada untuk menggunakan salah satu kelas instance DB berbasis. NVMe Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

Bacaan yang Dioptimalkan Aurora tersedia di semua Wilayah AWS tempat di mana satu atau lebih kelas instans DB dengan penyimpanan NVMe SSD lokal didukung. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB.

Untuk beralih kembali ke instance Aurora pembacaan yang tidak dioptimalkan, ubah kelas instans DB dari instance Aurora Anda ke kelas instance serupa NVMe tanpa penyimpanan sementara untuk beban kerja database Anda. Misalnya, jika kelas instans DB saat ini adalah db.r6gd.4xlarge, pilih db.r6g.4xlarge untuk beralih kembali. Untuk informasi selengkapnya, lihat Memodifikasi instans Aurora DB.

## Kasus penggunaan untuk Aurora Optimized Reads

Cache berjenjang yang didukung Optimized Reads

Berikut ini adalah beberapa kasus penggunaan yang dapat memperoleh manfaat dari Optimized Reads dengan cache berjenjang:

- Aplikasi skala internet seperti pemrosesan pembayaran, penagihan, e-commerce dengan kinerja SLAs yang ketat.
- Dasbor pelaporan waktu nyata yang menjalankan ratusan kueri titik untuk pengumpulan metrik/data.
- Aplikasi AI generatif dengan ekstensi pgvector untuk mencari neighbor yang tepat atau terdekat dalam jutaan penyematan vektor.

Objek sementara yang didukung Optimized Reads

Berikut ini adalah beberapa kasus penggunaan yang dapat memperoleh manfaat dari Optimized Reads dengan objek sementara:

- Kueri analitis yang mencakup Ekspresi Tabel Umum (CTEs), tabel turunan, dan operasi pengelompokan.
- Replika baca yang menangani kueri yang tidak dioptimalkan untuk aplikasi.
- Kueri pelaporan sesuai permintaan atau dinamis dengan operasi kompleks seperti GROUP BY dan ORDER BY yang tidak selalu dapat menggunakan indeks yang sesuai.
- `CREATE INDEX`atau `REINDEX` operasi untuk menyortir.
- Beban kerja lain yang menggunakan tabel sementara internal.

## Memantau instans DB yang menggunakan Aurora Optimized Reads

Anda dapat memantau kueri yang menggunakan cache berjenjang yang didukung Optimized Reads dengan perintah EXPLAIN seperti yang ditunjukkan pada contoh berikut:

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
------------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> `aurora_orcache_hit`dan `aurora_storage_read` bidang di `Buffers` bagian rencana penjelasan hanya ditampilkan ketika Pembacaan yang Dioptimalkan diaktifkan dan nilainya lebih besar dari nol. Bidang baca adalah total `aurora_storage_read` bidang `aurora_orcache_hit` dan.

Anda dapat memantau instans DB yang menggunakan Bacaan yang Dioptimalkan Aurora menggunakan metrik berikut CloudWatch:

- `AuroraOptimizedReadsCacheHitRatio`
- `FreeEphemeralStorage`
- `ReadIOPSEphemeralStorage`
- `ReadLatencyEphemeralStorage`
- `ReadThroughputEphemeralStorage`
- `WriteIOPSEphemeralStorage`
- `WriteLatencyEphemeralStorage`
- `WriteThroughputEphemeralStorage`

Metrik ini menyediakan data tentang penyimpanan instans, IOPS, dan throughput yang tersedia. Untuk informasi selengkapnya tentang metrik ini, lihat Metrik tingkat instans untuk Amazon Aurora.

Anda juga dapat menggunakan `pg_proctab` ekstensi untuk memantau NVMe penyimpanan.

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |    819350 |          0 |
 3847580 |      831020
```

```
      |       | rdsephemeralstorage |           23271 |            0 |       193098 |
   2620 |           114961 |             0 |        13845120 |    130770 |          0 |
  215010 |      133410
(2 rows)
```

### Praktik terbaik untuk Aurora Optimized Reads

Gunakan praktik terbaik berikut untuk Aurora Optimized Reads:

- Pantau ruang penyimpanan yang tersedia di penyimpanan instans dengan CloudWatch metrik`FreeEphemeralStorage`. Jika penyimpanan instance mencapai batasnya karena beban kerja pada instans DB, atur konkurensi dan kueri yang banyak menggunakan objek sementara atau memodifikasinya untuk menggunakan kelas instans DB yang lebih besar.
- Pantau CloudWatch metrik untuk hit rate cache Optimized Reads. Operasi seperti VACUUM akan memodifikasi sejumlah besar blok dengan sangat cepat. Hal ini dapat menyebabkan penurunan sementara dalam rasio hit. Ekstensi `pg_prewarm` dapat digunakan untuk memuat data ke dalam cache buffer yang memungkinkan Aurora secara proaktif menulis beberapa blok tersebut ke cache Optimized Reads.
- Anda dapat mengaktifkan manajemen cache klaster (CCM) untuk melakukan "warm up" cache buffer dan cache berjenjang pada pembaca tingkat 0, yang akan digunakan sebagai target failover. Ketika CCM diaktifkan, cache buffer dipindai secara berkala untuk menulis halaman yang memenuhi syarat untuk pengosongan dalam cache berjenjang. Untuk informasi selengkapnya tentang CCM, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL.

## Mengoptimalkan subkueri berkorelasi di Aurora PostgreSQL

Sebuah kolom tabel referensi subquery berkorelasi dari kueri luar. Ini dievaluasi sekali untuk setiap baris yang dikembalikan oleh kueri luar. Dalam contoh berikut, subquery referensi kolom dari tabel ot. Tabel ini tidak termasuk dalam klausa FROM subquery, tetapi direferensikan dalam klausa FROM kueri luar. Jika tabel ot memiliki 1 juta baris, subquery perlu dievaluasi 1 juta kali.

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

> **Note**
>
> - Transformasi subquery dan cache subquery tersedia di Aurora PostgreSQL dimulai dengan versi 16.8, sedangkan Babelfish untuk Aurora PostgreSQL mendukung fitur ini dari 4.2.0.
> - Dimulai dengan Babelfish untuk Aurora PostgreSQL versi 4.6.0 dan 5.2.0, parameter berikut mengontrol fitur-fitur ini:
>   - babelfishpg_tsql.apg_enable_correlated_scalar_transform
>   - babelfishpg_tsql.apg_enable_subquery_cache
>
>   Secara default, kedua parameter dihidupkan.

## Meningkatkan kinerja kueri PostgreSQL Aurora menggunakan transformasi subquery

Aurora PostgreSQL dapat mempercepat subquery yang berkorelasi dengan mengubahnya menjadi gabungan luar yang setara. Optimalisasi ini berlaku untuk dua jenis subkueri berkorelasi berikut:

- Subquery yang mengembalikan nilai agregat tunggal dan muncul dalam daftar SELECT.

```
SELECT ot.a, ot.b, (SELECT AVG(it.b) FROM it WHERE it.a = ot.a) FROM ot;
```

- Subquery yang mengembalikan nilai agregat tunggal dan muncul dalam klausa WHERE.

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

### Mengaktifkan transformasi dalam subquery

Untuk mengaktifkan transformasi subkueri yang berkorelasi menjadi gabungan luar yang setara, atur parameternya ke. `apg_enable_correlated_scalar_transform` `ON` Nilai default dari parameter ini adalah`OFF`.

Anda dapat memodifikasi kelompok parameter cluster atau instance untuk mengatur parameter. Untuk mempelajari selengkapnya, lihat .

Atau, Anda dapat mengonfigurasi pengaturan hanya untuk sesi saat ini dengan perintah berikut:

```
SET apg_enable_correlated_scalar_transform TO ON;
```

### Memverifikasi transformasi

Gunakan perintah EXPLORE untuk memverifikasi apakah subquery yang berkorelasi telah diubah menjadi gabungan luar dalam rencana kueri.

Ketika transformasi diaktifkan, bagian subquery berkorelasi yang berlaku akan diubah menjadi gabungan luar. Misalnya:

```
postgres=> CREATE TABLE ot (a INT, b INT);
CREATE TABLE
postgres=> CREATE TABLE it (a INT, b INT);
CREATE TABLE

postgres=> SET apg_enable_correlated_scalar_transform TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);

                        QUERY PLAN
-----------------------------------------------------------------
 Hash Join
   Hash Cond: (ot.a = apg_scalar_subquery.scalar_output)
   Join Filter: ((ot.b)::numeric < apg_scalar_subquery.avg)
   ->  Seq Scan on ot
   ->  Hash
         ->  Subquery Scan on apg_scalar_subquery
               ->  HashAggregate
                     Group Key: it.a
                     ->  Seq Scan on it
```

Kueri yang sama tidak diubah ketika parameter GUC diputar`OFF`. Rencana tidak akan memiliki gabungan luar tetapi subplan sebagai gantinya.

```
postgres=> SET apg_enable_correlated_scalar_transform TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);
               QUERY PLAN
----------------------------------------
 Seq Scan on ot
   Filter: ((b)::numeric < (SubPlan 1))
   SubPlan 1
     ->  Aggregate
```

```
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

Batasan

- Subquery harus dalam daftar SELECT atau dalam salah satu kondisi dalam klausa where. Jika tidak, itu tidak akan diubah.
- Subquery harus mengembalikan fungsi agregat. Fungsi agregat yang ditentukan pengguna tidak didukung untuk transformasi.
- Subquery yang ekspresi pengembaliannya bukan fungsi agregat sederhana tidak akan diubah.
- Kondisi berkorelasi dalam klausa WHERE subquery harus menjadi referensi kolom sederhana. Jika tidak, itu tidak akan diubah.
- Kondisi berkorelasi dalam subquery di mana klausa harus berupa predikat kesetaraan biasa.
- Subquery tidak dapat berisi klausa HAVING atau GROUP BY.
- Klausa where dalam subquery mungkin berisi satu atau lebih predikat yang dikombinasikan dengan AND.

> **Note**
>
> Dampak kinerja transformasi bervariasi tergantung pada skema, data, dan beban kerja Anda. Eksekusi subquery yang berkorelasi dengan transformasi dapat secara signifikan meningkatkan kinerja karena jumlah baris yang dihasilkan oleh kueri luar meningkat. Kami sangat menyarankan Anda menguji fitur ini di lingkungan non-produksi dengan skema, data, dan beban kerja Anda yang sebenarnya sebelum mengaktifkannya di lingkungan produksi.

## Menggunakan cache subquery untuk meningkatkan kinerja kueri PostgreSQL Aurora

Aurora PostgreSQL mendukung cache subquery untuk menyimpan hasil subquery berkorelasi. Fitur ini melewatkan eksekusi subquery berkorelasi berulang saat hasil subquery sudah ada di cache.

Memahami cache subquery

Node Memoize PostgreSQL adalah bagian penting dari cache subquery. Node Memoize mempertahankan tabel hash di cache lokal untuk memetakan dari nilai parameter input ke baris hasil kueri. Batas memori untuk tabel hash adalah produk dari work_mem dan hash_mem_multiplier. Untuk mempelajari lebih lanjut, lihat Konsumsi Sumber Daya.

Selama eksekusi kueri, cache subquery menggunakan Cache Hit Rate (CHR) untuk memperkirakan apakah cache meningkatkan kinerja kueri dan untuk memutuskan pada runtime kueri apakah akan terus menggunakan cache. CHR adalah rasio jumlah klik cache dengan jumlah total permintaan. Misalnya, jika subquery yang berkorelasi perlu dieksekusi 100 kali, dan 70 dari hasil eksekusi tersebut dapat diambil dari cache, CHR adalah 0,7.

Untuk setiap jumlah cache apg_subquery_cache_check_interval yang hilang, manfaat cache subquery dievaluasi dengan memeriksa apakah CHR lebih besar dari apg_subquery_cache_hit_rate_threshold. Jika tidak, cache akan dihapus dari memori, dan eksekusi kueri akan kembali ke eksekusi ulang subquery asli yang tidak di-cache.

Parameter yang mengontrol perilaku cache subquery

Tabel berikut mencantumkan parameter yang mengontrol perilaku cache subquery.

| Parameter | Deskripsi | Default | Diizinkan |
|---|---|---|---|
| apg_enable_subquery_cache | Mengaktifkan penggunaan cache untuk subquery skalar berkorelasi. | MATI | ON, OFF |
| apg_subquery_cache_check_interval | Menetapkan frekuensi, dalam jumlah kesalahan cache, untuk mengevaluasi tingkat hit cache subquery. | 500 | 0–2147483647 |
| apg_subquery_cache_hit_rate_threshold | Menetapkan ambang batas untuk hit rate cache subquery. | 0,3 | 0,0—1,0 |

**Note**

- Nilai yang lebih besar `apg_subquery_cache_check_interval` dapat meningkatkan akurasi estimasi manfaat cache berbasis ChR, tetapi akan meningkatkan overhead

cache, karena CHR tidak akan dievaluasi hingga tabel cache memiliki baris. `apg_subquery_cache_check_interval`

- Nilai `apg_subquery_cache_hit_rate_threshold` bias yang lebih besar untuk meninggalkan cache subquery dan kembali ke eksekusi ulang subquery asli yang tidak di-cache.

Anda dapat memodifikasi kelompok parameter cluster atau instance untuk mengatur parameter. Untuk mempelajari selengkapnya, lihat .

Atau, Anda dapat mengonfigurasi pengaturan hanya untuk sesi saat ini dengan perintah berikut:

```
SET apg_enable_subquery_cache TO ON;
```

Mengaktifkan cache subquery di Aurora PostgreSQL

Ketika cache subquery diaktifkan, Aurora PostgreSQL menerapkan cache untuk menyimpan hasil subquery. Rencana kueri kemudian akan memiliki node Memoize di bawah SubPlan.

Misalnya, urutan perintah berikut menunjukkan perkiraan rencana eksekusi kueri dari subquery berkorelasi sederhana tanpa cache subquery.

```
postgres=> SET apg_enable_subquery_cache TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Seq Scan on it
           Filter: (a = ot.a)
```

Setelah dihidupkan`apg_enable_subquery_cache`, rencana kueri akan berisi node Memoize di bawah SubPlan node, yang menunjukkan bahwa subquery berencana untuk menggunakan cache.

```
postgres=> SET apg_enable_subquery_cache TO ON;
```

```
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Memoize
           Cache Key: ot.a
           Cache Mode: binary
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

Rencana eksekusi kueri yang sebenarnya berisi rincian lebih lanjut dari cache subquery, termasuk klik cache dan kesalahan cache. Output berikut menunjukkan rencana eksekusi query aktual dari contoh query di atas setelah memasukkan beberapa nilai ke tabel.

```
postgres=> EXPLAIN (COSTS FALSE, TIMING FALSE, ANALYZE TRUE) SELECT ot.a, ot.b FROM ot
 WHERE ot.b < (SELECT it.b FROM it WHERE it.a = ot.a);
            QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on ot (actual rows=2 loops=1)
   Filter: (b < (SubPlan 1))
   Rows Removed by Filter: 8
   SubPlan 1
     ->  Memoize (actual rows=0 loops=10)
           Cache Key: ot.a
           Cache Mode: binary
           Hits: 4  Misses: 6  Evictions: 0  Overflows: 0  Memory Usage: 1kB
           ->  Seq Scan on it (actual rows=0 loops=6)
                 Filter: (a = ot.a)
                 Rows Removed by Filter: 4
```

Total nomor hit cache adalah 4, dan total nomor cache miss adalah 6. Jika jumlah total hit dan misses kurang dari jumlah loop di node Memoize, itu berarti evaluasi CHR tidak lulus dan cache dibersihkan dan ditinggalkan di beberapa titik. Eksekusi subquery kemudian kembali ke eksekusi ulang asli yang tidak di-cache.

Batasan

Cache subquery tidak mendukung pola tertentu dari subquery yang berkorelasi. Jenis kueri tersebut akan dijalankan tanpa cache, bahkan jika cache subquery diaktifkan:

- IN/EXISTS/ANY/ALLsubquery berkorelasi
- Subquery berkorelasi yang berisi fungsi nondeterministik.
- Subkueri berkorelasi yang mereferensikan kolom tabel luar dengan tipe data yang tidak mendukung operasi hashing atau kesetaraan.

# Meningkatkan kinerja kueri menggunakan gabungan adaptif

## Gambaran Umum

Adaptive join adalah fitur pratinjau di Aurora PostgreSQL 17.4 yang membantu meningkatkan kinerja kueri. Fitur ini dinonaktifkan secara default, tetapi Anda dapat mengaktifkannya menggunakan parameter Global User Configuration (GUC). Karena ini adalah fitur pratinjau, nilai parameter default mungkin berubah. Saat diaktifkan, gabungan adaptif membantu mengoptimalkan kinerja kueri dengan beralih secara dinamis dari gabungan loop bersarang ke gabungan hash saat runtime. Peralihan ini terjadi ketika pengoptimal PostgreSQL salah memilih sambungan loop bersarang karena perkiraan kardinalitas yang tidak akurat.

## Mengkonfigurasi gabungan adaptif

Anda dapat mengontrol gabungan adaptif menggunakan tiga parameter GUC ini:

Parameter konfigurasi gabungan adaptif

| Parameter GUC | Deskripsi | Opsi default dan konfigurasi |
|---|---|---|
| apg_adaptive_join_crossover_multiplier | Pengganda ini bekerja dengan titik crossover baris untuk menentukan kapan harus beralih dari loop bersarang ke gabungan hash. Titik crossover baris adalah di mana pengoptimal SQL memperkirakan bahwa loop bersarang dan operasi gabungan hash memiliki biaya yang | Mengontrol apakah Adaptive Join diaktifkan<br>• Nilai default: -1 (dinonaktifkan)<br>• Rentang yang valid: -1 hingga DBL_MAX |

| Parameter GUC | Deskripsi | Opsi default dan konfigurasi |
| --- | --- | --- |
| | sama. Nilai multiplier yang lebih tinggi mengurangi kemungkinan pengalihan gabungan adaptif ke gabungan hash. | • Untuk mengaktifkan: Setel ke >= 1 |
| apg_adaptive_join_cost_threshold | Parameter ini menetapkan ambang biaya kueri minimum. Gabungan adaptif secara otomatis menonaktifkan dirinya sendiri untuk kueri di bawah ambang batas ini. Ini mencegah overhead kinerja dalam kueri sederhana di mana biaya perencanaan gabungan adaptif dapat melebihi manfaat beralih dari loop bersarang ke gabungan hash. | Menetapkan ambang biaya minimum untuk kueri<br>• Nilai default: 100<br>• Rentang yang valid: 0 hingga DBL_MAX |

| Parameter GUC | Deskripsi | Opsi default dan konfigurasi |
| --- | --- | --- |
| apg_enable_parameterized_adaptive_join | Parameter ini memperluas fungsionalitas gabungan adaptif ke gabungan loop bersarang berparameter saat diaktifkan. Secara default, gabungan adaptif hanya berfungsi dengan gabungan loop bersarang tanpa parameter, karena ini lebih mungkin mendapat manfaat dari beralih ke gabungan hash. Gabungan loop bersarang berparameter biasanya berkinerja lebih baik, membuat peralihan ke bergabung dengan hash menjadi kurang penting. | Mengontrol perilaku gabungan adaptif untuk gabungan loop bersarang<br>• Nilai default: salah<br>• Nilai yang benar: benar/salah<br>  • Ketika false: Hanya berfungsi dengan gabungan loop bersarang tanpa parameter<br>  • Ketika true: Bekerja dengan gabungan loop bersarang berparameter dan tidak berparameter<br>**Note**<br>`apg_adaptive_join_crossover_multiplier` Harus diaktifkan terlebih dahulu |

# Bekerja dengan tabel yang tidak tercatat di Aurora PostgreSQL

Amazon Aurora PostgreSQL mendukung tabel yang tidak tercatat yang aman untuk crash dan menjaga integritas data bahkan setelah kegagalan atau kegagalan instance penulis. Dalam

PostgreSQL standar, tabel yang tidak tercatat melewati Write-Ahead Log (WAL) selama operasi penulisan, menghasilkan kecepatan tulis yang lebih cepat. Namun, ini datang dengan biaya mengurangi daya tahan, karena tabel yang tidak tercatat tidak aman untuk crash dan dapat kehilangan data setelah kegagalan sistem atau shutdown yang tidak bersih. Tabel yang tidak tercatat ini secara otomatis terpotong setelah crash atau shutdown yang tidak bersih. Konten dan indeksnya juga tidak direplikasi ke server siaga.

Sebaliknya, Aurora PostgreSQL menangani tabel yang tidak tercatat secara berbeda karena arsitektur penyimpanan terdistribusi. Ini karena sistem penyimpanan Aurora tidak bergantung pada PostgreSQL WAL tradisional untuk daya tahan. Namun, manfaat kinerja yang biasanya terkait dengan tabel yang tidak tercatat di PostgreSQL standar mungkin tidak begitu signifikan di Aurora. Hal ini disebabkan arsitektur penyimpanan terdistribusi Aurora, yang dapat memperkenalkan overhead tambahan dibandingkan dengan penyimpanan lokal yang digunakan dalam PostgreSQL standar.

Saat menggunakan tabel yang tidak tercatat di Aurora PostgreSQL, pertimbangkan hal berikut:

- Anda dapat mengakses tabel yang tidak tercatat hanya dari node penulis di cluster Aurora DB.
- Node pembaca dapat mengakses tabel yang tidak tercatat hanya ketika dipromosikan ke status penulis.
- Ketika Anda mencoba mengakses tabel yang tidak tercatat dari node pembaca, itu akan menghasilkan kesalahan berikut:

  ```
  cannot access temporary or unlogged relations during recovery.
  ```

## Membuat tabel yang tidak tercatat

Untuk membuat tabel yang tidak tercatat di Aurora PostgreSQL, tambahkan kata kunci UNLOGGED dalam pernyataan CREATE TABLE Anda:

```
CREATE UNLOGGED TABLE staging_sales_data (
    transaction_id bigint,
    customer_id bigint,
    product_id bigint,
    transaction_date date,
    amount NUMERIC
);
```

## Mengonversi tabel yang tidak tercatat ke tabel yang dicatat

Saat Anda perlu mengonversi tabel yang tidak tercatat kembali ke tabel yang dicatat, Anda dapat menggunakan perintah berikut:

```
ALTER TABLE table_name SET LOGGED;
```

Operasi ini menulis ulang seluruh tabel dan menempatkan kunci eksklusif di atasnya sampai operasi selesai. Untuk tabel besar, ini dapat mengakibatkan downtime yang signifikan.

## Tabel yang tidak tersumbat dan replikasi logis

Tabel yang tidak tercatat umumnya tidak termasuk dalam replikasi logis karena replikasi logis bergantung pada WAL untuk menangkap dan mentransfer perubahan. Secara default, perubahan pada tabel yang tidak tercatat tidak dicatat oleh WAL dan dikecualikan dari aliran replikasi, membuatnya tidak cocok untuk kasus penggunaan di mana replikasi logis diperlukan. Namun, Aurora PostgreSQL menyediakan parameter yang disebut `rds.logically_replicate_unlogged_tables` yang memungkinkan Anda mengontrol perilaku ini:

- Ketika `rds.logically_replicate_unlogged_tables` diatur ke 0 (off) tabel yang tidak tercatat dikecualikan dari replikasi logis.
- Ketika `rds.logically_replicate_unlogged_tables` diatur ke 1 (on) tabel yang tidak tercatat disertakan dalam replikasi logis.

> **Note**
>
> Di Aurora PostgreSQL, `rds.logically_replicate_unlogged_tables` parameter diatur secara default ke 1 (on) di versi 14 dan sebelumnya, dan ke 0 (off) di versi 15 dan yang lebih baru.

# Bekerja dengan PostgreSQL autovacuum di Amazon RDS Aurora PostgreSQL

Kami sangat menyarankan Anda menggunakan fitur autovacuum untuk menjaga kondisi instans DB PostgreSQL Anda. Autovacuum akan mengotomatiskan awal perintah VACUUM dan ANALYZE. Kemudian memeriksa tabel yang memuat banyak tuple yang dimasukkan, diperbarui, atau dihapus. Setelah pemeriksaan ini, autovacuum akan mengambil kembali penyimpanan dengan menghapus data usang atau tuple dari basis data PostgreSQL.

Secara default, autovacuum diaktifkan untuk DB instance yang Anda buat menggunakan salah satu grup parameter PostgreSQL DB default. Parameter konfigurasi lain yang terkait dengan fitur autovacuum juga diatur secara default. Karena parameter default ini cukup umum, Anda dapat memanfaatkannya dengan menyetel beberapa parameter yang terkait dengan fitur autovacuum untuk beban kerja spesifik Anda.

Berikut ini, Anda dapat menemukan informasi lebih lanjut tentang autovacuum dan cara menyetel beberapa parameternya pada DB instance.

Topik


- Mengalokasikan memori untuk autovacuum
- Mengurangi kemungkinan penyelesaian ID transaksi
- Menentukan apakah tabel di basis data Anda perlu divakum atau tidak
- Menentukan tabel mana yang saat ini memenuhi syarat untuk autovacuum
- Menentukan apakah autovacuum saat ini sedang berjalan atau tidak, dan berapa lama durasinya
- Melakukan pembekuan vakum manual
- Mengindeks ulang tabel saat autovacuum berjalan
- Mengelola autovacuum dengan indeks berukuran besar
- Parameter lain yang memengaruhi autovacuum
- Mengatur parameter autovacuum tingkat tabel
- Melakukan log aktivitas autovacuum dan vakum
- Memahami perilaku autovacuum dengan database yang tidak valid
- Identifikasi dan selesaikan penghambat vakum agresif di

# Mengalokasikan memori untuk autovacuum

Salah satu parameter paling penting yang memengaruhi kinerja autovacuum adalah parameter autovacuum_work_mem. Dalam Aurora PostgreSQL versi 14 dan autovacuum_work_mem sebelumnya, parameter diatur ke -1, menunjukkan bahwa pengaturan digunakan sebagai gantinya. maintenance_work_mem Untuk semua versi lainnya, autovacuum_work_mem ditentukan oleh GREATEST ({DBInstanceClassMemory/32768}, 65536).

Operasi vakum manual selalu menggunakan maintenance_work_mem pengaturan, dengan pengaturan default GREATEST ({DBInstanceClassMemory/63963136 *1024}, 65536), dan juga dapat disesuaikan pada tingkat sesi menggunakan perintah untuk operasi manual yang lebih bertarget. SET VACUUM

Memori autovacuum_work_mem menentukan untuk autovacuum untuk menampung pengidentifikasi tupel mati () untuk indeks penyedot debu. pg_stat_all_tables.n_dead_tup

Saat melakukan perhitungan untuk menentukan nilai autovacuum_work_mem parameter, perhatikan hal-hal berikut:

- Jika Anda mengatur parameter terlalu rendah, proses vakum mungkin harus memindai tabel beberapa kali untuk menyelesaikan pekerjaannya. Pemindaian berulang seperti ini dapat berdampak negatif pada performa. Untuk contoh yang lebih besar, pengaturan maintenance_work_mem atau autovacuum_work_mem setidaknya 1 GB dapat meningkatkan kinerja tabel penyedot debu dengan jumlah tupel mati yang tinggi. Namun, dalam PostgreSQL versi 16 dan sebelumnya, penggunaan memori vakum dibatasi pada 1 GB, yang cukup untuk memproses sekitar 179 juta tupel mati dalam satu lintasan. Jika tabel memiliki lebih banyak tupel mati daripada ini, vakum perlu membuat beberapa lintasan melalui indeks tabel, secara signifikan meningkatkan waktu yang dibutuhkan. Dimulai dengan PostgreSQL versi 17, tidak ada batasan 1 GB, dan autovacuum dapat memproses lebih dari 179 juta tupel dengan menggunakan pohon radix.

  Pengenal Tuple berukuran 6 byte. Untuk memperkirakan memori yang diperlukan untuk menyedot indeks tabel, kueri pg_stat_all_tables.n_dead_tup untuk menemukan jumlah tupel mati, lalu kalikan angka ini dengan 6 untuk menentukan memori yang diperlukan untuk menyedot debu indeks dalam satu lintasan. Anda dapat menggunakan kueri berikut:

```
SELECT
    relname AS table_name,
    n_dead_tup,
```

```
    pg_size_pretty(n_dead_tup * 6) AS estimated_memory
FROM
    pg_stat_all_tables
WHERE
    relname = 'name_of_the_table';
```

- Parameter `autovacuum_work_mem` bekerja secara bersama-sama dengan parameter `autovacuum_max_workers`. Setiap pekerja di antara `autovacuum_max_workers` dapat menggunakan memori yang Anda alokasikan. Jika Anda memiliki banyak tabel kecil, alokasikan lebih banyak parameter `autovacuum_max_workers` dan lebih sedikit parameter `autovacuum_work_mem`. Jika Anda memiliki tabel besar (lebih besar dari 100 GB), alokasikan lebih banyak memori dan lebih sedikit proses pekerja. Anda harus memiliki cukup memori yang dialokasikan agar berhasil di tabel terbesar Anda. Jadi, pastikan bahwa kombinasi proses pekerja dan memori sama dengan total memori yang ingin Anda alokasikan.

## Mengurangi kemungkinan penyelesaian ID transaksi

Dalam beberapa kasus, pengaturan kelompok parameter yang terkait dengan autovacuum mungkin tidak cukup agresif untuk mencegah penyelesaian ID transaksi. Untuk mengatasi hal ini, Aurora PostgreSQL menyediakan mekanisme yang menyesuaikan nilai parameter autovacuum secara otomatis. Autovacuum adaptif adalah fitur untuk RDS untuk PostgreSQL Aurora . Penjelasan terperinci tentang TransactionID wraparound dapat ditemukan dalam dokumentasi PostgreSQL.

Autovacuum adaptif diaktifkan secara default untuk RDS untuk instance PostgreSQL dengan parameter dinamis disetel ke ON. `rds.adaptive_autovacuum` Kami sangat menyarankan Anda untuk tetap mengaktifkan parameter dinamis ini. Namun, untuk menonaktifkan penyesuaian parameter autovacuum adaptif, atur parameter `rds.adaptive_autovacuum` ke 0 atau nonaktifkan.

Sampul ID transaksi masih dimungkinkan bahkan ketika menyetel parameter autovacuum. Kami mendorong Anda untuk menerapkan CloudWatch alarm Amazon untuk sampul ID transaksi. Untuk informasi selengkapnya, lihat posting Menerapkan sistem peringatan dini untuk sampul ID transaksi di RDS untuk PostgreSQL di Blog Database. AWS

Dengan penyetelan parameter autovacuum adaptif diaktifkan, Amazon RDS mulai menyesuaikan parameter autovacuum ketika CloudWatch metrik `MaximumUsedTransactionIDs` mencapai nilai parameter atau 500.000.000, mana yang lebih besar. `autovacuum_freeze_max_age`

Amazon RDS terus menyesuaikan parameter untuk autovacuum jika tabel terus menuju ke penyelesaian ID transaksi. Setiap penyesuaian ini secara khusus mengalokasikan lebih banyak

sumber daya ke autovacuum untuk menghindari penyelesaian. Amazon RDS memperbarui parameter terkait autovacuum berikut ini:

- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit
- `autovacuum_work_mem`
- autovacuum_naptime

RDS akan memodifikasi parameter ini hanya jika nilai yang baru membuat autovacuum lebih agresif. Parameter dimodifikasi dalam memori di instans DB. Nilai di grup parameter tidak diubah. Untuk melihat pengaturan dalam memori saat ini, gunakan perintah PostgreSQL SHOW SQL.

Jika Amazon RDS memodifikasi salah satu parameter autovacuum ini, Amazon RDS akan menghasilkan peristiwa untuk instans DB yang terdampak. Acara ini terlihat di AWS Management Console dan melalui Amazon RDS API. Setelah `MaximumUsedTransactionIDs` CloudWatch metrik kembali di bawah ambang batas, Amazon RDS mengatur ulang parameter terkait autovacuum dalam memori kembali ke nilai yang ditentukan dalam grup parameter. Kemudian, Amazon RDS akan menghasilkan peristiwa lain yang sesuai dengan perubahan ini.

## Menentukan apakah tabel di basis data Anda perlu divakum atau tidak

Anda dapat menggunakan kueri berikut untuk menunjukkan jumlah transaksi yang tidak dibekukan dalam database. `datfrozenxid`Kolom `pg_database` baris database adalah batas bawah pada transaksi normal yang IDs muncul di database itu. Kolom ini adalah minimum dari nilai per tabel `relfrozenxid` di dalam basis data.

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age(datfrozenxid) desc
 limit 20;
```

Sebagai contoh, hasil dari menjalankan kueri sebelumnya adalah sebagai berikut.

```
datname    | age
mydb       | 1771757888
template0  | 1721757888
template1  | 1721757888
rdsadmin   | 1694008527
postgres   | 1693881061
(5 rows)
```

Ketika usia database mencapai 2 miliar transaksi IDs, sampul ID transaksi (XID) terjadi dan database menjadi read-only. Anda dapat menggunakan kueri ini untuk menghasilkan metrik dan menjalankannya beberapa kali dalam sehari. Secara default, autovacuum diatur untuk menjaga usia transaksi tidak lebih dari 200.000.000 (`autovacuum_freeze_max_age`).

Contoh strategi pemantauan mungkin terlihat seperti ini:

- Tetapkan nilai `autovacuum_freeze_max_age` hingga 200 juta transaksi.
- Jika sebuah tabel mencapai 500 juta transaksi yang tidak dibekukan, itu memicu alarm tingkat keparahan rendah. Tidak ada yang dengan nilai ini, tetapi dapat mengindikasikan bahwa autovacuum tidak dapat diteruskan.
- Jika tabel berumur 1 miliar, indikasi ini harus diperlakukan sebagai peringatan untuk diambil tindakan. Umumnya, Anda ingin mempertahankan usia tabel mendekati `autovacuum_freeze_max_age` karena alasan performa. Sebaiknya Anda menyelidiki hal ini menggunakan rekomendasi berikut.
- Jika tabel mencapai 1,5 miliar transaksi yang tidak divakum, ini akan memicu alarm dengan keparahan tingkat tinggi. Bergantung pada seberapa cepat database Anda menggunakan transaksi IDs, alarm ini dapat menunjukkan bahwa sistem kehabisan waktu untuk menjalankan autovacuum. Dalam hal ini, sebaiknya Anda segera menyelesaikan proses ini.

Jika tabel terus-menerus melanggar ambang batas ini, ubah parameter autovacuum Anda. Secara default, menggunakan VACUUM secara manual (yang menonaktifkan penundaan berbasis biaya) bersifat lebih agresif dibandingkan menggunakan autovacuum default, tetapi juga lebih mengganggu sistem secara keseluruhan.

Sebaiknya lakukan hal berikut:

- Waspada dan aktifkan mekanisme pemantauan agar Anda dapat mengetahui usia transaksi Anda yang paling lama.

  Untuk informasi tentang membuat proses yang memperingatkan Anda tentang sampul ID transaksi, lihat posting Blog AWS Database Menerapkan sistem peringatan dini untuk sampul ID transaksi di Amazon RDS for PostgreSQL.
- Untuk tabel yang lebih sibuk, lakukan pembekuan vakum manual secara teratur selama pemeliharaan, selain mengandalkan autovacuum. Untuk informasi cara melakukan pembekuan vakum manual, lihat Melakukan pembekuan vakum manual.

# Menentukan tabel mana yang saat ini memenuhi syarat untuk autovacuum

Sering kali, ada satu atau dua tabel yang harus divakum. Tabel dengan nilai `relfrozenxid` lebih besar dari jumlah transaksi dalam `autovacuum_freeze_max_age` selalu menjadi target autovacuum. Sebaliknya, jika jumlah tuple yang dibuat usang sejak VACUUM terakhir melebihi "ambang batas vakum", tabel akan divakum.

Ambang batas autovacuum didefinisikan sebagai:

```
Vacuum-threshold = vacuum-base-threshold + vacuum-scale-factor * number-of-tuples
```

di mana `vacuum base threshold` adaa`utovacuum_vacuum_threshold`, `vacuum scale factor` adalaha`utovacuum_vacuum_scale_factor`, dan `number of tuples` adalah`pg_class.reltuples`.

Saat Anda terkoneksi ke basis data, jalankan kueri berikut untuk melihat daftar tabel yang dianggap oleh autovacuum telah memenuhi syarat untuk divakum.

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM
pg_settings WHERE name = 'autovacuum_vacuum_threshold'),
vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM
pg_settings WHERE name = 'autovacuum_vacuum_scale_factor'),
fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name =
 'autovacuum_freeze_max_age'),
sto AS (select opt_oid, split_part(setting, '=', 1) as param,
split_part(setting, '=', 2) as value from (select oid opt_oid, unnest(reloptions)
 setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,
pg_size_pretty(pg_table_size(c.oid)) as table_size,
age(relfrozenxid) as xid_age,
coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1)
 join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid =
 cvbt.opt_oid
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid =
 cvsf.opt_oid
```

```
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)
ORDER BY age(relfrozenxid) DESC LIMIT 50;
```

## Menentukan apakah autovacuum saat ini sedang berjalan atau tidak, dan berapa lama durasinya

Jika harus memvakum tabel secara manual, pastikan Anda bisa menentukan apakah autovacuum saat ini sedang berjalan atau tidak. Jika sudah berjalan, Anda mungkin perlu menyesuaikan parameter agar berjalan lebih efisien, atau mematikan autovacuum sementara sehingga Anda dapat menjalankan VACUUM secara manual.

Gunakan kueri berikut untuk menentukan apakah autovacuum berjalan, berapa lama sudah berjalan, dan apakah sedang menunggu sesi lainnya atau tidak.

```
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query
FROM pg_stat_activity
WHERE upper(query) LIKE '%VACUUM%'
ORDER BY xact_start;
```

Setelah menjalankan kueri, Anda akan melihat hasil yang serupa dengan yang berikut ini.

```
 datname | usename  |  pid  | state  | wait_event |      xact_runtime       | query
 --------+----------+-------+--------+------------+-------------------------
+-------------------------------------------------------------------------------------------
 mydb    | rdsadmin | 16473 | active |            | 33 days 16:32:11.600656 |
 autovacuum: VACUUM ANALYZE public.mytable1 (to prevent wraparound)
 mydb    | rdsadmin | 22553 | active |            | 14 days 09:15:34.073141 |
 autovacuum: VACUUM ANALYZE public.mytable2 (to prevent wraparound)
 mydb    | rdsadmin | 41909 | active |            | 3 days 02:43:54.203349  |
 autovacuum: VACUUM ANALYZE public.mytable3
 mydb    | rdsadmin |   618 | active |            | 00:00:00                |
 SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query+
```

```
        |          |       |        |            |                          | FROM
pg_stat_activity
          +
        |          |       |        |            |                          | WHERE
query like '%VACUUM%'
         +
        |          |       |        |            |                          | ORDER BY
xact_start;
      +
```

Beberapa masalah dapat menyebabkan sesi autovacuum yang lama (yaitu beberapa hari). Masalah yang paling umum adalah nilai parameter `maintenance_work_mem` Anda diatur terlalu rendah untuk ukuran tabel atau laju pembaruan.

Sebaiknya Anda menggunakan rumus berikut untuk menetapkan nilai parameter `maintenance_work_mem`.

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

Sesi autovacuum yang singkat juga dapat mengindikasikan adanya masalah:

- Sesi ini dapat mengindikasikan bahwa `autovacuum_max_workers` tidak cukup untuk beban kerja Anda. Dalam hal ini, Anda perlu menetapkan jumlah pekerja.
- Itu dapat mengindikasikan adanya kerusakan indeks (kesalahan pada autovacuum dan mulai ulang di hal yang sama, tetapi tidak ada kemajuan). Dalam hal ini, jalankan `vacuum freeze verbose` *`table`* manual untuk mengetahui penyebab pastinya.

## Melakukan pembekuan vakum manual

Anda mungkin dapat melakukan vakum manual di tabel yang memiliki proses vakum yang sudah berjalan. Cara ini berguna jika Anda telah mengidentifikasi tabel dengan usia yang mendekati 2 miliar transaksi (atau di atas ambang batas yang Anda pantau).

Langkah-langkah berikut ini merupakan panduan, yang disertai dengan beberapa variasi proses. Misalnya, selama pengujian, anggaplah Anda menemukan bahwa nilai parameter `maintenance_work_mem` diatur terlalu kecil dan Anda harus segera mengambil tindakan di tabel. Namun, mungkin pada saat ini Anda tidak ingin mengembalikan instans. Dengan menggunakan kueri di bagian sebelumnya, Anda dapat menentukan tabel mana yang menjadi masalah dan melihat sesi

autovacuum yang berjalan lama. Anda tahu bahwa Anda perlu mengubah pengaturan parameter `maintenance_work_mem`, tetapi Anda juga harus mengambil tindakan cepat dan memvakum tabel yang dimaksud. Prosedur berikut menunjukkan tindakan apa yang harus dilakukan dalam situasi ini.

Untuk melakukan pembekuan vakum secara manual

1. Buka dua sesi ke basis data yang berisi tabel yang ingin divakum. Untuk sesi kedua, gunakan "layar" atau peralatan lain yang dapat mempertahankan sesi jika koneksi terputus.
2. Dalam sesi pertama, dapatkan ID proses (PID) dari sesi autovacuum yang berjalan di tabel.

   Jalankan kueri berikut untuk mendapatkan PID dari sesi autovacuum.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) LIKE '%VACUUM%' ORDER BY
   xact_start;
   ```

3. Dalam sesi kedua, hitung jumlah memori yang Anda butuhkan untuk menjalankan operasi ini. Dalam contoh ini, kita akan menentukan bahwa kita mampu menggunakan memori hingga 2 GB untuk operasi ini sehingga kita menetapkan `maintenance_work_mem` untuk sesi saat ini hingga 2 GB.

   ```
   SET maintenance_work_mem='2 GB';
   SET
   ```

4. Dalam sesi kedua, munculkan perintah `vacuum freeze verbose` untuk tabel. Pengaturan verbose berguna karena Anda tetap dapat melihat aktivitas pembekuan tersebut meskipun tidak ada laporan kemajuannya di PostgreSQL saat ini.

   ```
   \timing on
   Timing is on.
   vacuum freeze verbose pgbench_branches;
   ```

   ```
   INFO:  vacuuming "public.pgbench_branches"
   INFO:  index "pgbench_branches_pkey" now contains 50 row versions in 2 pages
   DETAIL:  0 index row versions were removed.
   0 index pages have been deleted, 0 are currently reusable.
   CPU 0.00s/0.00u sec elapsed 0.00 sec.
   INFO:  index "pgbench_branches_test_index" now contains 50 row versions in 2 pages
   DETAIL:  0 index row versions were removed.
   ```

```
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  "pgbench_branches": found 0 removable, 50 nonremovable row versions
     in 43 out of 43 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 9347 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
Time: 2.765 ms
```

5. Dalam sesi satu, jika autovacuum memblokir sesi vakum, `pg_stat_activity` menunjukkan bahwa T menunggu adalah sesi vakum Anda. Dalam hal ini, akhiri proses autovacuum sebagai berikut.

```
SELECT pg_terminate_backend('the_pid');
```

> **Note**
>
> Beberapa versi Aurora yang lebih rendah tidak dapat menghentikan proses autovacuum menggunakan perintah sebelumnya dan gagal dengan kesalahan berikut:. `ERROR: 42501: must be a superuser to terminate superuser process LOCATION: pg_terminate_backend, signalfuncs.c:227` Untuk menemukan versi PostgreSQL yang telah ditambal, cari bullet berikut di pembaruan Amazon Aurora PostgreSQL:
>
> ```
> Allow rds_superuser to terminate backends which are not explicitly
>  associated with a role
> ```

Di titik ini, sesi Anda dimulai. Autovacuum restart segera karena tabel ini mungkin yang tertinggi dalam daftar pekerjaannya.

6. Mulai perintah `vacuum freeze verbose` di sesi kedua, lalu akhiri proses autovacuum di sesi pertama.

# Mengindeks ulang tabel saat autovacuum berjalan

Jika indeks rusak, autovacuum tetap akan terus memproses tabel lalu gagal. Jika mencoba vakum manual dalam situasi ini, Anda akan menerima pesan kesalahan seperti berikut ini.

```
postgres=>  vacuum freeze pgbench_branches;
ERROR: index "pgbench_branches_test_index" contains unexpected
   zero page at block 30521
HINT: Please REINDEX it.
```

Saat indeks rusak dan autovacuum mencoba untuk tetap berjalan di tabel, maka Anda bersaing dengan sesi autovacuum yang sudah berjalan. Jika Anda mengeluarkan perintah “REINDEX“, kunci eksklusif akan diambil di tabel. Operasi penulisan, dan juga operasi membaca yang menggunakan indeks spesifik tersebut, akan diblokir.

Untuk mengindeks ulang tabel saat autovacuum berjalan di tabel

1. Buka dua sesi ke basis data yang berisi tabel yang ingin divakum. Untuk sesi kedua, gunakan "layar" atau peralatan lain yang dapat mempertahankan sesi jika koneksi terputus.
2. Dalam sesi pertama, dapatkan PID dari sesi autovacuum yang berjalan di tabel.

   Jalankan kueri berikut untuk mendapatkan PID dari sesi autovacuum.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) like '%VACUUM%' ORDER BY
   xact_start;
   ```

3. Dalam sesi kedua, munculkan perintah pengindeksan ulang.

   ```
   \timing on
   Timing is on.
   reindex index pgbench_branches_test_index;
   REINDEX
     Time: 9.966 ms
   ```

4. Dalam sesi pertama, jika autovacuum memblokir proses, Anda akan melihat di `pg_stat_activity` bahwa proses menunggu untuk sesi vakum diberi tanda "T". Dalam hal ini, Anda mengakhiri proses autovacuum.

```
SELECT pg_terminate_backend('the_pid');
```

Di titik ini, sesi Anda dimulai. Perlu diperhatikan bahwa proses autovacuum akan segera dimulai ulang karena tabel ini mungkin merupakan urutan tertinggi di daftar pekerjaan.

5. Mulai perintah Anda di sesi kedua, lalu akhiri proses autovacuum di sesi 1.

## Mengelola autovacuum dengan indeks berukuran besar

Sebagai bagian dari operasinya, autovacuum akan melakukan beberapa fase vakum saat berjalan di tabel. Sebelum tabel dibersihkan, semua indeksnya akan divakum terlebih dahulu. Fase menghapus beberapa indeks berukuran besar akan menghabiskan banyak waktu dan sumber daya. Oleh karena itu, sebagai praktik terbaik, pastikan Anda mengontrol jumlah indeks di tabel dan menyingkirkan indeks yang tidak digunakan.

Untuk proses ini, pertama-tama periksa ukuran indeks keseluruhan. Kemudian, tentukan apakah ada indeks yang berpotensi tidak digunakan dan dapat dihapus seperti yang ditunjukkan dalam contoh berikut.

Untuk memeriksa ukuran tabel beserta indeksnya

```
postgres=> select pg_size_pretty(pg_relation_size('pgbench_accounts'));
pg_size_pretty
6404 MB
(1 row)
```

```
postgres=> select pg_size_pretty(pg_indexes_size('pgbench_accounts'));
pg_size_pretty
11 GB
(1 row)
```

Dalam contoh ini, ukuran indeks lebih besar dari tabel. Perbedaan ini dapat menyebabkan masalah performa karena indeks membengkak atau tidak digunakan sehingga berdampak pada operasi autovacuum serta operasi penyisipan.

Untuk memeriksa indeks yang tidak digunakan

Dengan menggunakan tampilan pg_stat_user_indexes, Anda dapat memeriksa seberapa sering indeks digunakan dengan kolom idx_scan. Dalam contoh berikut, indeks yang tidak digunakan memiliki nilai idx_scan dari 0.

```
postgres=> select * from pg_stat_user_indexes where relname = 'pgbench_accounts' order
 by idx_scan desc;

relid  | indexrelid | schemaname | relname          | indexrelname          | idx_scan
 | idx_tup_read | idx_tup_fetch
-------+------------+------------+------------------+-----------------------+----------
+--------------+---------------
16433  | 16454      | public     | pgbench_accounts | index_f               | 6
 | 6            | 0
16433  | 16450      | public     | pgbench_accounts | index_b               | 3
 | 199999       | 0
16433  | 16447      | public     | pgbench_accounts | pgbench_accounts_pkey | 0
 | 0            | 0
16433  | 16452      | public     | pgbench_accounts | index_d               | 0
 | 0            | 0
16433  | 16453      | public     | pgbench_accounts | index_e               | 0
 | 0            | 0
16433  | 16451      | public     | pgbench_accounts | index_c               | 0
 | 0            | 0
16433  | 16449      | public     | pgbench_accounts | index_a               | 0
 | 0            | 0
(7 rows)
```

```
postgres=> select schemaname, relname, indexrelname, idx_scan from pg_stat_user_indexes
 where relname = 'pgbench_accounts' order by idx_scan desc;

schemaname  | relname          | indexrelname          | idx_scan
------------+------------------+-----------------------+----------
public      | pgbench_accounts | index_f               | 6
public      | pgbench_accounts | index_b               | 3
public      | pgbench_accounts | pgbench_accounts_pkey | 0
public      | pgbench_accounts | index_d               | 0
public      | pgbench_accounts | index_e               | 0
public      | pgbench_accounts | index_c               | 0
public      | pgbench_accounts | index_a               | 0
(7 rows)
```

> **Note**
>
> Statistik ini bersifat inkremental sejak statistik diatur ulang. Misalkan Anda memiliki indeks yang hanya digunakan pada akhir kuartal bisnis atau hanya untuk laporan tertentu. Ada kemungkinan bahwa indeks ini belum digunakan sejak statistik diatur ulang. Untuk informasi selengkapnya, lihat Fungsi Statistik. Indeks yang digunakan untuk menerapkan keunikan tidak akan memiliki pemindaian yang dilakukan dan tidak boleh diidentifikasi sebagai indeks yang tidak digunakan. Untuk mengidentifikasi indeks yang tidak digunakan, Anda harus memiliki pengetahuan mendalam tentang aplikasi dan pertanyaannya.

Untuk memeriksa kapan statistik terakhir basis data disetel ulang, gunakan `pg_stat_database`

```
postgres=> select datname, stats_reset from pg_stat_database where datname =
 'postgres';

datname   | stats_reset
----------+-------------------------------
postgres  | 2022-11-17 08:58:11.427224+00
(1 row)
```

## Melakukan vakum di tabel secepat mungkin

RDS untuk PostgreSQL 12 dan versi lebih tinggi

Jika Anda memiliki terlalu banyak indeks dalam tabel besar, instans DB Anda mungkin mendekati penyelesaian ID transaksi (XID), yaitu saat penghitung XID mendekati nol. Jika tidak terkendali, situasi ini dapat mengakibatkan kehilangan data. Namun, Anda dapat dengan cepat melakuan vakum di tabel tanpa harus membersihkan indeks. Dalam RDS untuk PostgreSQL 12 dan versi lebih tinggi, Anda dapat menggunakan VACUUM dengan klausa `INDEX_CLEANUP`.

```
postgres=> VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) pgbench_accounts;

INFO: vacuuming "public.pgbench_accounts"
INFO: table "pgbench_accounts": found 0 removable, 8 nonremovable row versions in 1 out
 of 819673 pages
DETAIL: 0 dead row versions cannot be removed yet, oldest xmin: 7517
Skipped 0 pages due to buffer pins, 0 frozen pages.
CPU: user: 0.01 s, system: 0.00 s, elapsed: 0.01 s.
```

Jika sesi autovacuum sudah berjalan, Anda harus menghentikannya agar dapat memulai VACUUM manual. Untuk informasi cara melakukan pembekuan vakum manual, lihat Melakukan pembekuan vakum manual.

> **Note**
>
> Melewatkan pembersihan indeks secara teratur menyebabkan indeks kembung, yang menurunkan kinerja pemindaian. Indeks mempertahankan baris mati, dan tabel mempertahankan pointer garis mati. Akibatnya, `pg_stat_all_tables.n_dead_tup` meningkat hingga autovacuum atau VACUUM manual dengan pembersihan indeks berjalan. Sebagai praktik terbaik, gunakan prosedur ini hanya untuk mencegah pembungkus ID transaksi.

RDS untuk PostgreSQL 11 dan versi lama

Namun, dalam RDS untuk PostgreSQL 11 dan versi lama, satu-satunya cara agar vakum dapat selesai lebih cepat adalah dengan mengurangi jumlah indeks di tabel. Menghapus sementara indeks dapat memengaruhi rencana kueri. Sebaiknya Anda menghapus sementara indeks yang tidak digunakan terlebih dahulu, lalu menghapus sementara indeks saat penyelesaian XID sangat dekat. Setelah proses vakum selesai, Anda dapat membuat ulang indeks ini.

## Parameter lain yang memengaruhi autovacuum

Kueri berikut menunjukkan nilai dari beberapa parameter yang secara langsung memengaruhi autovacuum dan perilakunya. Parameter autovacuum dijelaskan dengan lengkap dalam dokumentasi PostgreSQL.

```
SELECT name, setting, unit, short_desc
FROM pg_settings
WHERE name IN (
'autovacuum_max_workers',
'autovacuum_analyze_scale_factor',
'autovacuum_naptime',
'autovacuum_analyze_threshold',
'autovacuum_analyze_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_scale_factor',
```

```
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_cost_delay',
'autovacuum_vacuum_cost_limit',
'vacuum_cost_limit',
'autovacuum_freeze_max_age',
'maintenance_work_mem',
'vacuum_freeze_min_age');
```

Meskipun semua ini memengaruhi autovacuum, beberapa hal yang paling penting adalah:

- maintenance_work_mem
- autovacuum_freeze_max_age
- autovacuum_max_workers
- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit

## Mengatur parameter autovacuum tingkat tabel

Anda dapat mengatur parameter penyimpanan terkait autovacuum di tingkat tabel, dan biasanya cara ini bisa lebih baik daripada harus mengubah perilaku seluruh basis data. Untuk tabel besar, Anda mungkin perlu mengatur pengaturan yang agresif dan tidak ingin membuat autovacuum berperilaku seperti itu untuk semua tabel.

Kueri berikut menunjukkan tabel mana yang saat ini memiliki opsi tingkat tabel.

```
SELECT relname, reloptions
FROM pg_class
WHERE reloptions IS NOT null;
```

Pengaturan ini mungkin berguna pada tabel yang jauh lebih besar daripada tabel lainnya, misalnya. Misalkan Anda memiliki satu tabel berukuran 300 GB dan 30 table lainnya berukuran kurang dari 1 GB. Dalam hal ini, Anda dapat mengatur beberapa parameter khusus untuk tabel besar sehingga tidak perlu mengubah perilaku dari seluruh sistem.

```
ALTER TABLE mytable set (autovacuum_vacuum_cost_delay=0);
```

Melakukan tindakan ini akan menonaktifkan penundaan autovacuum berbasis biaya untuk tabel ini dengan mengorbankan lebih banyak penggunaan sumber daya di sistem Anda. Biasanya, jeda

autovacuum untuk `autovacuum_vacuum_cost_delay` setiap kali `autovacuum_cost_limit` tercapai. Untuk mengetahui detail selengkapnya, lihat dokumentasi PostgreSQL tentang pemvakuman berbasis biaya.

## Melakukan log aktivitas autovacuum dan vakum

Informasi mengenai aktivitas autovacuum dikirim ke `postgresql.log` berdasarkan level yang ditentukan dalam parameter `rds.force_autovacuum_logging_level`. Berikut ini adalah nilai yang diizinkan untuk parameter ini dan versi PostgreSQL dengan nilai berupa pengaturan default:

- `disabled` (PostgreSQL 10, PostgreSQL 9.6)
- `debug5`, `debug4`, `debug3`, `debug2`, `debug1`
- `info` (PostgreSQL 12, PostgreSQL 11)
- `notice`
- `warning` (PostgreSQL 13 dan versi lebih tinggi)
- `error`, log `fatal`, `panic`

`rds.force_autovacuum_logging_level` berfungsi dengan parameter `log_autovacuum_min_duration`. Nilai parameter `log_autovacuum_min_duration` adalah ambang batas (dalam milidetik) tempat tindakan autovacuum dicatat. Pengaturan `-1` tidak mencatat apa pun, sedangkan pengaturan 0 mencatat semua tindakan. Seperti halnya`rds.force_autovacuum_logging_level`, nilai default untuk `log_autovacuum_min_duration` bergantung pada versi, sebagai berikut:

- `10000 ms` – PostgreSQL 14, PostgreSQL 13, PostgreSQL 12, dan PostgreSQL 11
- `(empty)` – Tidak ada nilai default untuk PostgreSQL 10 dan PostgreSQL 9.6

Sebaiknya Anda mengatur `rds.force_autovacuum_logging_level` ke `WARNING`. Sebaiknya Anda juga mengatur `log_autovacuum_min_duration` ke nilai dari 1000 hingga 5000. Pengaturan aktivitas 5000 log yang membutuhkan waktu lebih dari 5.000 milidetik. Pengaturan apa pun selain -1 juga akan mencatat pesan jika tindakan autovacuum dilewati karena kunci yang bertentangan atau hubungan yang dihapus sementara secara bersamaan. Untuk informasi selengkapnya, lihat Automatic Vacuuming dalam dokumentasi PostgreSQL.

Untuk memecahkan masalah, Anda dapat mengubah parameter `rds.force_autovacuum_logging_level` ke salah satu level debug, dari `debug1` hingga

debug5 untuk informasi panjang. Sebaiknya Anda menggunakan pengaturan debug untuk jangka waktu singkat dan hanya untuk tujuan pemecahan masalah. Untuk mempelajari selengkapnya, lihat When to log dalam dokumentasi PostgreSQL.

> **Note**
>
> PostgreSQL memungkinkan akun `rds_superuser` untuk melihat sesi autovacuum di `pg_stat_activity`. Misalnya, Anda dapat mengidentifikasi dan mengakhiri sesi autovacuum yang memblokintah agar tidak berjalan, atau berjalan lebih lambat daripada perintah vakum yang dikeluarkan secara manual.

## Memahami perilaku autovacuum dengan database yang tidak valid

Nilai `-2` baru diperkenalkan ke `datconnlimit` kolom dalam `pg_database` katalog untuk menunjukkan database yang telah terputus di tengah operasi DROP DATABASE sebagai tidak valid.

- 15.4 dan semua versi yang lebih tinggi
- 14.9 dan versi yang lebih tinggi
- 13.12 dan versi yang lebih tinggi
- 12.16 dan versi yang lebih tinggi
- 11.21 dan versi yang lebih tinggi

Database yang tidak valid tidak memengaruhi kemampuan autovacuum untuk membekukan fungsionalitas untuk database yang valid. Autovacuum mengabaikan database yang tidak valid. Akibatnya, operasi vakum reguler akan terus berfungsi dengan baik dan efisien untuk semua database yang valid di lingkungan PostgreSQL Anda.

Topik

- Memantau ID transaksi
- Menyesuaikan kueri pemantauan
- Menyelesaikan masalah database yang tidak valid

## Memantau ID transaksi

`age(datfrozenxid)`Fungsi ini biasa digunakan untuk memantau usia transaksi ID (XID) database untuk mencegah pembungkus ID transaksi.

Karena database yang tidak valid dikecualikan dari autovacuum, penghitung ID transaksi (XID) mereka dapat mencapai nilai maksimum`2 billion`, membungkus, dan melanjutkan siklus ini tanpa batas waktu. `- 2 billion` Kueri khas untuk memantau sampul ID Transaksi mungkin terlihat seperti:

```
SELECT max(age(datfrozenxid)) FROM pg_database;
```

Namun, dengan diperkenalkannya nilai -2 untuk`datconnlimit`, database yang tidak valid dapat memiringkan hasil kueri ini. Karena database ini tidak valid dan tidak boleh menjadi bagian dari pemeriksaan pemeliharaan rutin, mereka dapat menyebabkan positif palsu, membuat Anda percaya bahwa `age(datfrozenxid)` itu lebih tinggi daripada yang sebenarnya.

## Menyesuaikan kueri pemantauan

Untuk memastikan pemantauan yang akurat, Anda harus menyesuaikan kueri pemantauan untuk mengecualikan database yang tidak valid. Ikuti kueri yang disarankan ini:

```
SELECT
    max(age(datfrozenxid))
FROM
    pg_database
WHERE
    datconnlimit <> -2;
```

Kueri ini memastikan bahwa hanya database yang valid yang dipertimbangkan dalam `age(datfrozenxid)` perhitungan, memberikan refleksi sebenarnya dari usia ID transaksi di seluruh lingkungan PostgreSQL Anda.

## Menyelesaikan masalah database yang tidak valid

Saat mencoba menyambung ke database yang tidak valid, Anda mungkin menemukan pesan galat yang mirip dengan berikut ini:

```
postgres=> \c db1
connection to server at "mydb.xxxxxxxxxx.us-west-2.rds.amazonaws.com" (xx.xx.xx.xxx),
 port xxxx failed: FATAL:  cannot connect to invalid database "db1"
HINT:  Use DROP DATABASE to drop invalid databases.
Previous connection kept
```

Selain itu, jika `log_min_messages` parameter disetel ke `DEBUG2` atau lebih tinggi, Anda mungkin melihat entri log berikut yang menunjukkan bahwa proses autovacuum melewatkan database yang tidak valid:

```
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db6"
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db1"
```

Untuk mengatasi masalah ini, ikuti yang `HINT` disediakan selama upaya koneksi. Connect ke database yang valid menggunakan akun master RDS Anda atau akun database dengan `rds_superuser` peran tersebut, dan jatuhkan database yang tidak valid.

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

## Identifikasi dan selesaikan penghambat vakum agresif di

Di PostgreSQL, menyedot debu sangat penting untuk memastikan kesehatan database karena merebut kembali penyimpanan dan mencegah masalah pembungkus ID transaksi. Namun, ada kalanya penyedot debu dapat dicegah agar tidak beroperasi sesuai keinginan, yang dapat mengakibatkan penurunan kinerja, pembengkakan penyimpanan, dan bahkan memengaruhi ketersediaan instans DB Anda dengan sampul ID transaksi. Oleh karena itu, mengidentifikasi dan menyelesaikan masalah ini sangat penting untuk kinerja dan ketersediaan database yang optimal.

Baca Memahami autovacuum di Amazon RDS untuk lingkungan PostgreSQL untuk mempelajari lebih lanjut tentang autovacuum.

`postgres_get_av_diag()`Fungsi ini membantu mengidentifikasi masalah yang mencegah atau menunda kemajuan vakum yang agresif. Saran disediakan, yang mungkin termasuk perintah untuk menyelesaikan masalah yang dapat diidentifikasi atau panduan untuk diagnostik lebih lanjut di mana masalah tidak dapat diidentifikasi. Penghambat vakum agresif dilaporkan ketika usia melebihi ambang batas autovacuum adaptif RDS sebesar 500 juta transaksi. IDs

Berapa usia ID transaksi?

`age()`Fungsi untuk transaksi IDs menghitung jumlah transaksi yang telah terjadi sejak ID transaksi yang tidak dibekukan tertua untuk database (`pg_database.datfrozenxid`) atau tabel (`pg_class.relfrozenxid`). Nilai ini menunjukkan aktivitas database sejak operasi vakum agresif terakhir dan menyoroti kemungkinan beban kerja untuk proses VACUUM yang akan datang.

Apa itu vakum agresif?

Operasi VACUUM yang agresif melakukan pemindaian komprehensif semua halaman dalam tabel, termasuk yang biasanya dilewati selama reguler. VACUUMs Pemindaian menyeluruh ini bertujuan untuk “membekukan” transaksi IDs mendekati usia maksimum mereka, secara efektif mencegah situasi yang dikenal sebagai sampul ID transaksi.

`postgres_get_av_diag()`Untuk melaporkan pemblokir, pemblokir harus berusia setidaknya 500 juta transaksi.

Topik

- Menginstal alat pemantauan dan diagnostik autovacuum di RDS untuk PostgreSQL Aurora
- 
- 
- 
- Menyelesaikan masalah kinerja vakum di
- Penjelasan pesan PEMBERITAHUAN di

## Menginstal alat pemantauan dan diagnostik autovacuum di RDS untuk PostgreSQL Aurora

`postgres_get_av_diag()`Fungsi ini saat ini tersedia dalam :

- 17.4 dan versi 17 yang lebih tinggi
- 16.7 dan versi 16 yang lebih tinggi
- 15.11 dan versi 15 yang lebih tinggi
- 14.16 dan versi 14 yang lebih tinggi
- 13.19 dan 13 versi yang lebih tinggi

Untuk menggunakan`postgres_get_av_diag()`, buat `rds_tools` ekstensi.

```
postgres=> CREATE EXTENSION rds_tools ;
CREATE EXTENSION
```

Verifikasi bahwa ekstensi diinstal.

```
postgres=> \dx rds_tools
             List of installed extensions
   Name    | Version |  Schema   |                    Description
 ----------+---------+-----------
+------------------------------------------------------------
 rds_tools |   1.9   | rds_tools | miscellaneous administrative functions for RDS
 PostgreSQL
 1 row
```

Verifikasi bahwa fungsi tersebut dibuat.

```
postgres=> SELECT
    proname function_name,
    pronamespace::regnamespace function_schema,
    proowner::regrole function_owner
FROM
    pg_proc
WHERE
    proname = 'postgres_get_av_diag';
    function_name     | function_schema | function_owner
----------------------+-----------------+----------------
 postgres_get_av_diag | rds_tools       | rds_superuser
(1 row)
```

`postgres_get_av_diag()`Fungsi ini mengambil informasi diagnostik tentang proses autovacuum yang memblokir atau tertinggal dalam RDS untuk PostgreSQL Aurora PostgreSQL database . Kueri

harus dieksekusi dalam database dengan ID transaksi tertua untuk hasil yang akurat. Untuk informasi selengkapnya tentang menggunakan database dengan ID transaksi tertua, lihat Tidak terhubung ke database dengan usia ID transaksi terlama

```
SELECT
    blocker,
    DATABASE,
    blocker_identifier,
    wait_event,
    TO_CHAR(autovacuum_lagging_by, 'FM9,999,999,999') AS autovacuum_lagging_by,
    suggestion,
    suggested_action
FROM (
    SELECT
        *
    FROM
        rds_tools.postgres_get_av_diag ()
    ORDER BY
        autovacuum_lagging_by DESC) q;
```

`postgres_get_av_diag()`Fungsi mengembalikan tabel dengan informasi berikut:

pemblokir

Menentukan kategori aktivitas database yang memblokir vakum.

- Pernyataan aktif
- Idle pada transaksi
- Transaksi yang disiapkan
- Slot replikasi logis
- Contoh pembaca
- Tabel sementara

basis data

Menentukan nama database jika berlaku dan didukung. Ini adalah database di mana aktivitas sedang berlangsung dan memblokir atau akan memblokir autovacuum. Ini adalah database yang harus Anda sambungkan dan ambil tindakan.

blocker_identifier

Menentukan pengenal aktivitas yang memblokir atau akan memblokir autovacuum. Pengidentifikasi dapat berupa ID proses bersama dengan pernyataan SQL, transaksi yang disiapkan, alamat IP replika baca, dan nama slot replikasi, baik logis atau fisik.

wait_event

Menentukan menunggu sesi pemblokiran dan berlaku untuk blocker berikut:

- Pernyataan aktif
- Idle pada transaksi

autovacum_lagging_by

Menentukan jumlah transaksi yang autovacuum tertinggal dalam pekerjaan backlog-nya per kategori.

saran

Menentukan saran untuk menyelesaikan blocker. Instruksi ini mencakup nama database tempat aktivitas ada jika berlaku, ID Proses (PID) sesi jika berlaku, dan tindakan yang akan diambil.

suggested_action

Menyarankan tindakan yang perlu diambil untuk menyelesaikan pemblokir.

Autovacuum melakukan penyedot debu agresif dan menurunkan usia transaksi IDs hingga di bawah ambang batas yang ditentukan oleh parameter instans RDS Anda. `autovacuum_freeze_max_age` Anda dapat melacak usia ini menggunakan CloudWatch metrik Amazon`MaximumUsedTransactionIDs`.

Untuk menemukan pengaturan `autovacuum_freeze_max_age` (yang memiliki default 200 juta transaksi IDs) untuk instans Amazon RDS Anda, Anda dapat menggunakan kueri berikut:

```
SELECT
    TO_CHAR(setting::bigint, 'FM9,999,999,999') autovacuum_freeze_max_age
FROM
    pg_settings
WHERE
    name = 'autovacuum_freeze_max_age';
```

Perhatikan bahwa `postgres_get_av_diag()` hanya memeriksa penghambat vakum agresif ketika usia melebihi ambang batas autovacuum adaptif Amazon RDS sebesar 500 juta transaksi. IDs `postgres_get_av_diag()`Untuk mendeteksi pemblokir, pemblokir harus berusia setidaknya 500 juta transaksi.

`postgres_get_av_diag()`Fungsi ini mengidentifikasi jenis blocker berikut:

Topik

- Pernyataan aktif
- Idle pada transaksi
- Transaksi yang disiapkan
- Slot replikasi logis
- Contoh pembaca
- Tabel sementara

### Pernyataan aktif

Dalam PostgreSQL, pernyataan aktif adalah pernyataan SQL yang saat ini sedang dijalankan oleh database. Ini termasuk kueri, transaksi, atau operasi apa pun yang sedang berlangsung. Saat memantau via`pg_stat_activity`, kolom status menunjukkan bahwa proses dengan PID yang sesuai aktif.

`postgres_get_av_diag()`Fungsi ini menampilkan output yang mirip dengan berikut ketika mengidentifikasi pernyataan yang merupakan pernyataan aktif.

```
blocker               | Active statement
database              | my_database
blocker_identifier    | SELECT pg_sleep(20000);
wait_event            | Timeout:PgSleep
autovacuum_lagging_by | 568,600,871
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (29621);"}
```

### Tindakan yang disarankan

Mengikuti panduan di `suggestion` kolom, pengguna dapat terhubung ke database di mana pernyataan aktif hadir dan, seperti yang ditentukan dalam `suggested_action` kolom,

disarankan untuk hati-hati meninjau opsi untuk mengakhiri sesi. Jika penghentian aman, Anda dapat menggunakan `pg_terminate_backend()` fungsi untuk mengakhiri sesi. Tindakan ini dapat dilakukan oleh administrator (seperti akun master RDS) atau pengguna dengan `pg_terminate_backend()` hak istimewa yang diperlukan.

> **Warning**
>
> Sesi yang dihentikan akan membatalkan (`ROLLBACK`) perubahan yang dibuatnya. Tergantung pada kebutuhan Anda, Anda mungkin ingin menjalankan kembali pernyataan tersebut. Namun, disarankan untuk melakukannya hanya setelah proses autovacuum menyelesaikan operasi vakum agresifnya.

Idle pada transaksi

Pernyataan transaksi yang menganggur mengacu pada setiap sesi yang telah membuka transaksi eksplisit (seperti dengan mengeluarkan `BEGIN` pernyataan), melakukan beberapa pekerjaan, dan sekarang menunggu klien untuk melewati lebih banyak pekerjaan atau memberi sinyal akhir transaksi dengan mengeluarkan`COMMIT`,`ROLLBACK`, atau `END` (yang akan menghasilkan implisit). `COMMIT`

`postgres_get_av_diag()`Fungsi ini menampilkan output yang mirip dengan berikut ketika mengidentifikasi `idle in transaction` pernyataan sebagai pemblokir.

```
blocker               | idle in transaction
database              | my_database
blocker_identifier    | INSERT INTO tt SELECT * FROM tt;
wait_event            | Client:ClientRead
autovacuum_lagging_by | 1,237,201,759
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (28438);"}
```

Tindakan yang disarankan

Seperti yang ditunjukkan dalam `suggestion` kolom, Anda dapat terhubung ke database tempat idle dalam sesi transaksi hadir dan mengakhiri sesi menggunakan fungsi. `pg_terminate_backend()` Pengguna dapat menjadi pengguna admin Anda (akun master RDS) atau pengguna dengan `pg_terminate_backend()` hak istimewa.

> **Warning**
>
> Sesi yang dihentikan akan membatalkan (`ROLLBACK`) perubahan yang dibuatnya. Tergantung pada kebutuhan Anda, Anda mungkin ingin menjalankan kembali pernyataan tersebut. Namun, disarankan untuk melakukannya hanya setelah proses autovacuum menyelesaikan operasi vakum agresifnya.

Transaksi yang disiapkan

PostgreSQL memungkinkan transaksi yang merupakan bagian dari strategi komit dua fase yang disebut transaksi siap. Ini diaktifkan dengan mengatur `max_prepared_transactions` parameter ke nilai bukan nol. Transaksi yang disiapkan dirancang untuk memastikan bahwa transaksi tahan lama dan tetap tersedia bahkan setelah database crash, restart, atau pemutusan klien. Seperti transaksi reguler, mereka diberi ID transaksi dan dapat memengaruhi autovacuum. Jika dibiarkan dalam keadaan siap, autovacuum tidak dapat melakukan freeezing dan dapat menyebabkan penutupan ID transaksi.

Ketika transaksi dibiarkan dipersiapkan tanpa batas waktu tanpa diselesaikan oleh manajer transaksi, mereka menjadi transaksi yang disiapkan yatim piatu. Satu-satunya cara untuk memperbaikinya adalah dengan melakukan atau mengembalikan transaksi menggunakan `ROLLBACK PREPARED` perintah `COMMIT PREPARED` atau, masing-masing.

> **Note**
>
> Ketahuilah bahwa cadangan yang diambil selama transaksi yang disiapkan akan tetap berisi transaksi tersebut setelah restorasi. Lihat informasi berikut tentang cara menemukan dan menutup transaksi tersebut.

`postgres_get_av_diag()`Fungsi ini menampilkan output berikut ketika mengidentifikasi pemblokir yang merupakan transaksi yang disiapkan.

```
blocker               | Prepared transaction
database              | my_database
blocker_identifier    | myptx
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
```

```
suggestion            | Connect to database "my_database" and consider either COMMIT
 or ROLLBACK the prepared transaction using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"COMMIT PREPARED 'myptx';",[OR],"ROLLBACK PREPARED 'myptx';"}
```

Tindakan yang disarankan

Seperti disebutkan di kolom saran, sambungkan ke database tempat transaksi yang disiapkan berada. Berdasarkan `suggested_action` kolom, hati-hati meninjau apakah akan melakukan salah satu `COMMIT` atau`ROLLBACK`, dan menyetujui tindakan.

Untuk memantau transaksi yang disiapkan secara umum, PostgreSQL menawarkan tampilan katalog yang disebut. `pg_prepared_xacts` Anda dapat menggunakan kueri berikut untuk menemukan transaksi yang disiapkan.

```
SELECT
    gid,
    prepared,
    owner,
    database,
    transaction AS oldest_xmin
FROM
    pg_prepared_xacts
ORDER BY
    age(transaction) DESC;
```

Slot replikasi logis

Tujuan dari slot replikasi adalah untuk menahan perubahan yang tidak dikonsumsi sampai mereka direplikasi ke server target. Untuk informasi lebih lanjut, lihat Replikasi Logis PostgreSQL.

Ada dua jenis slot replikasi logis.

Slot replikasi logis tidak aktif

Ketika replikasi dihentikan, log transaksi yang tidak dikonsumsi tidak dapat dihapus, dan slot replikasi menjadi tidak aktif. Meskipun slot replikasi logis yang tidak aktif saat ini tidak digunakan oleh pelanggan, itu tetap di server, yang mengarah ke retensi file WAL dan mencegah penghapusan log transaksi lama. Ini dapat meningkatkan penggunaan disk dan secara khusus memblokir autovacuum dari membersihkan tabel katalog internal, karena sistem harus menjaga informasi LSN agar tidak

ditimpa. Jika tidak ditangani, hal ini dapat mengakibatkan kembung katalog, penurunan kinerja, dan peningkatan risiko kekosongan sampul, yang berpotensi menyebabkan downtime transaksi.

Slot replikasi logis aktif tapi lambat

Terkadang penghapusan tupel mati katalog tertunda karena penurunan kinerja replikasi logis. Keterlambatan replikasi ini memperlambat pembaruan `catalog_xmin` dan dapat menyebabkan katalog kembung dan vakum sampul.

`postgres_get_av_diag()`Fungsi ini menampilkan output yang mirip dengan berikut ini ketika menemukan slot replikasi logis sebagai pemblokir.

```
blocker               | Logical replication slot
database              | my_database
blocker_identifier    | slot1
wait_event            | Not applicable
autovacuum_lagging_by | 1,940,103,068
suggestion            | Ensure replication is active and resolve any lag for the slot
 if active. If inactive, consider dropping it using the command in suggested_action.
 For more information, see Working with PostgreSQL autovacuum in the Amazon RDS User
 Guide.
suggested_action      | {"SELECT pg_drop_replication_slot('slot1') FROM
 pg_replication_slots WHERE active = 'f';"}
```

Tindakan yang disarankan

Untuk mengatasi masalah ini, periksa konfigurasi replikasi untuk masalah dengan skema target atau data yang mungkin menghentikan proses penerapan. Alasan paling umum adalah sebagai berikut:

- Kolom hilang
- Tipe data yang tidak kompatibel
- Ketidakcocokan data
- Tabel hilang

Jika masalah terkait dengan masalah infrastruktur:

- Masalah jaringan - Bagaimana cara mengatasi masalah dengan Amazon RDS DB dalam status jaringan yang tidak kompatibel? .
- Database atau instans DB tidak tersedia karena alasan berikut:

- Instans replika kehabisan penyimpanan - Tinjau instans Amazon RDS DB kehabisan penyimpanan untuk informasi tentang menambahkan penyimpanan.
- Parameter Tidak Kompatibel - Tinjau Bagaimana cara memperbaiki instans Amazon RDS DB yang macet dalam status parameter yang tidak kompatibel? untuk informasi lebih lanjut tentang bagaimana Anda dapat menyelesaikan masalah.

Jika instans Anda berada di luar AWS jaringan atau aktif AWS EC2, konsultasikan dengan administrator Anda tentang cara mengatasi masalah ketersediaan atau terkait infrastruktur.

Menjatuhkan slot yang tidak aktif

> **Warning**
>
> Perhatian: Sebelum menjatuhkan slot replikasi, pastikan dengan hati-hati bahwa ia tidak memiliki replikasi yang sedang berlangsung, tidak aktif, dan dalam keadaan tidak dapat dipulihkan. Menjatuhkan slot sebelum waktunya dapat mengganggu replikasi atau menyebabkan kehilangan data.

Setelah memastikan bahwa slot replikasi tidak lagi diperlukan, jatuhkan untuk memungkinkan autovacuum melanjutkan. Kondisi ini `active = 'f'` memastikan bahwa hanya slot yang tidak aktif yang dijatuhkan.

```
SELECT pg_drop_replication_slot('slot1') WHERE active ='f'
```

Contoh pembaca

Ketika pengaturan hot_standby_feedback diaktifkan, ini mencegah autovacuum pada instance penulis menghapus baris mati yang mungkin masih diperlukan oleh kueri yang berjalan pada instance pembaca. Perilaku ini diperlukan karena kueri yang berjalan pada instance pembaca (juga berlaku untuk instance pembaca di Aurora Global Database) mengharuskan baris tersebut tetap tersedia pada instance penulis yang mencegah konflik dan pembatalan kueri.

> **Note**
>
> `hot_standby_feedback`diaktifkan secara default dan tidak dapat dimodifikasi di Aurora PostgreSQL.

`postgres_get_av_diag()`Fungsi ini menampilkan output yang mirip dengan berikut ini ketika menemukan replika baca dengan slot replikasi fisik sebagai pemblokir.

```
blocker               | Oldest query running on aurora reader
database              | Not applicable
blocker_identifier    | my-aurora-reader-2
wait_event            | Not applicable
autovacuum_lagging_by | 540,122,859
suggestion            | Run the following query on the reader "my-aurora-reader-2" to
 find the long running query:

                      | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 523476310;

                      | Review carefully and you may consider terminating the query on
 reader using suggested_action.
suggested_action      | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 523476310;","

                      | [OR]


                      | ","Delete the reader if not needed"}
```

Seperti yang direkomendasikan di `suggested_action` kolom, tinjau opsi ini dengan cermat untuk membuka blokir autovacuum.

- Mengakhiri kueri — Mengikuti panduan di kolom saran, Anda dapat terhubung ke replika baca, seperti yang ditentukan dalam kolom suggested_action, disarankan untuk hati-hati meninjau opsi untuk mengakhiri sesi. Jika penghentian dianggap aman, Anda dapat menggunakan `pg_terminate_backend()` fungsi ini untuk mengakhiri sesi. Tindakan ini dapat dilakukan oleh administrator (seperti akun master RDS) atau pengguna dengan hak istimewa pg_terminate_backend () yang diperlukan.

  Anda dapat menjalankan perintah SQL berikut pada replika baca untuk mengakhiri kueri yang mencegah vakum pada primer membersihkan baris lama. Nilai dilaporkan dalam output fungsi: `backend_xmin`

  ```
  SELECT
      pg_terminate_backend(pid)
  FROM
  ```

```
    pg_catalog.pg_stat_activity
WHERE
    backend_xmin::text::bigint = backend_xmin;
```

- Hapus instance pembaca jika tidak diperlukan — Jika instance pembaca tidak lagi diperlukan, Anda dapat menghapusnya. Ini akan menghapus overhead replikasi terkait dan memungkinkan primer untuk mendaur ulang log transaksi tanpa ditahan oleh instance.

Tabel sementara

Tabel sementara, dibuat menggunakan TEMPORARY kata kunci, berada dalam skema temp, misalnya pg_temp_xxx, dan hanya dapat diakses oleh sesi yang membuatnya. Tabel sementara dijatuhkan saat sesi berakhir. Namun, tabel ini tidak terlihat oleh proses autovacuum PostgreSQL, dan harus disedot secara manual oleh sesi yang membuatnya. Mencoba menyedot tabel suhu dari sesi lain tidak berpengaruh.

Dalam keadaan yang tidak biasa, tabel sementara ada tanpa sesi aktif yang memilikinya. Jika sesi kepemilikan berakhir secara tak terduga karena kecelakaan fatal, masalah jaringan, atau peristiwa serupa, tabel sementara mungkin tidak dibersihkan, meninggalkannya sebagai tabel “yatim piatu”. Ketika proses autovacuum PostgreSQL mendeteksi tabel sementara yatim piatu, ia mencatat pesan berikut:

```
LOG: autovacuum: found orphan temp table \"%s\".\"%s\" in database \"%s\"
```

`postgres_get_av_diag()`Fungsi ini menampilkan output yang mirip dengan berikut ketika mengidentifikasi tabel sementara sebagai pemblokir. Agar fungsi menampilkan output yang terkait dengan tabel sementara dengan benar, itu perlu dieksekusi dalam database yang sama di mana tabel tersebut ada.

```
blocker               | Temporary table
database              | my_database
blocker_identifier    | pg_temp_14.ttemp
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database". Review carefully, you
 may consider dropping temporary table using command in suggested_action. For more
 information, see Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"DROP TABLE ttemp;"}
```

Tindakan yang disarankan

Ikuti instruksi yang diberikan di `suggestion` kolom output untuk mengidentifikasi dan menghapus tabel sementara yang mencegah autovacuum berjalan. Gunakan perintah berikut untuk menjatuhkan tabel sementara yang dilaporkan oleh`postgres_get_av_diag()`. Ganti nama tabel berdasarkan output yang disediakan oleh `postgres_get_av_diag()` fungsi.

```
DROP TABLE my_temp_schema.my_temp_table;
```

Kueri berikut dapat digunakan untuk mengidentifikasi tabel sementara:

```
SELECT
    oid,
    relname,
    relnamespace::regnamespace,
    age(relfrozenxid)
FROM
    pg_class
WHERE
relpersistence = 't'
ORDER BY
    age(relfrozenxid) DESC;
```

Bagian ini mengeksplorasi alasan tambahan yang dapat mencegah penyedot debu membuat kemajuan. Masalah-masalah ini saat ini tidak dapat diidentifikasi secara langsung oleh fungsi. `postgres_get_av_diag()`

Topik

- Inkonsistensi indeks
- Tingkat transaksi yang sangat tinggi

### Inkonsistensi indeks

Indeks yang tidak konsisten secara logis dapat mencegah autovacuum membuat kemajuan. Kesalahan berikut atau kesalahan serupa dicatat selama fase vakum indeks atau ketika indeks diakses oleh pernyataan SQL.

```
ERROR: right sibling's left-link doesn't match:block 5 links to 10 instead of expected
 2 in index ix_name
```

```
ERROR: failed to re-find parent key in index "XXXXXXXXXX" for deletion target page XXX
CONTEXT:  while vacuuming index index_name of relation schema.table
```

Bimbingan

Bangun kembali indeks atau lewati indeks menggunakan manual`INDEX_CLEANUP`. `VACUUM FREEZE`

- Menggunakan opsi CONCURRENT - Sebelum PostgreSQL versi 12, membangun kembali indeks memerlukan kunci tabel eksklusif, membatasi akses ke tabel. Dengan PostgreSQL versi 12, dan versi yang lebih baru, opsi CONCURRENT memungkinkan penguncian tingkat baris, secara signifikan meningkatkan ketersediaan tabel. Berikut ini adalah perintah:

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

  Sementara CONCURRENT kurang mengganggu, itu bisa lebih lambat pada tabel sibuk. Pertimbangkan untuk membangun indeks selama periode lalu lintas rendah jika memungkinkan. Untuk informasi selengkapnya, lihat REINDEX dalam dokumentasi PostgreSQL.
- Menggunakan opsi INDEX_CLEANUP FALSE — Jika indeksnya besar dan diperkirakan membutuhkan banyak waktu untuk menyelesaikannya, Anda dapat membuka blokir autovacuum dengan menjalankan VACUUM FREEZE manual sambil mengecualikan indeks. Fungsi ini tersedia di PostgreSQL versi 12 dan versi yang lebih baru.

  Melewati indeks akan memungkinkan Anda untuk melewati proses vakum dari indeks yang tidak konsisten dan mengurangi masalah sampul. Namun, ini tidak akan menyelesaikan masalah halaman tidak valid yang mendasarinya. Untuk sepenuhnya mengatasi dan menyelesaikan masalah halaman yang tidak valid, Anda masih perlu membangun kembali indeks.

Tingkat transaksi yang sangat tinggi

Di PostgreSQL, tingkat transaksi yang tinggi dapat secara signifikan memengaruhi kinerja autovacuum, yang mengarah pada pembersihan tupel mati yang lebih lambat dan peningkatan risiko sampul ID transaksi. Anda dapat memantau tingkat transaksi dengan mengukur perbedaan `max(age(datfrozenxid))` antara dua periode waktu, biasanya per detik. Selain itu, Anda dapat menggunakan metrik penghitung berikut dari RDS Performance Insights untuk mengukur tingkat transaksi (jumlah xact_commit dan xact_rollback) yang merupakan jumlah total transaksi.

| Penghitung | Jenis | Unit | Metrik |
|---|---|---|---|
| xact_commit | Transaksi | Commit per detik | db.Transactions.xact_commit |
| xact_rollback | Transaksi | Pemulihan per detik | db.Transactions.xact_rollback |

Peningkatan yang cepat menunjukkan beban transaksi yang tinggi, yang dapat membanjiri autovacuum, menyebabkan kembung, perselisihan kunci, dan potensi masalah kinerja. Hal ini dapat berdampak negatif pada proses autovacuum dalam beberapa cara:

- Aktivitas Tabel: Tabel spesifik yang disedot dapat mengalami volume transaksi yang tinggi, menyebabkan penundaan.
- Sistem Sumber Daya Sistem secara keseluruhan mungkin kelebihan beban, sehingga sulit bagi autovacuum untuk mengakses sumber daya yang diperlukan agar berfungsi secara efisien.

Pertimbangkan strategi berikut untuk memungkinkan autovacuum beroperasi lebih efektif dan mengikuti tugasnya:

1. Kurangi tingkat transaksi jika memungkinkan. Pertimbangkan untuk mengelompokkan atau mengelompokkan transaksi serupa jika memungkinkan.
2. Targetkan tabel yang sering diperbarui dengan `VACUUM FREEZE` operasi manual setiap malam, mingguan, atau dua mingguan selama jam sibuk.
3. Pertimbangkan untuk meningkatkan kelas instans Anda untuk mengalokasikan lebih banyak sumber daya sistem untuk menangani volume transaksi dan autovacuum yang tinggi.

## Menyelesaikan masalah kinerja vakum di

Bagian ini membahas faktor-faktor yang sering berkontribusi pada kinerja vakum yang lebih lambat dan bagaimana mengatasi masalah ini.

Topik

- Vakum indeks besar
- Terlalu banyak tabel atau database untuk vakum

- Vakum agresif (untuk mencegah pembungkus) sedang berjalan

Vakum indeks besar

VACUUM beroperasi melalui fase berurutan: inisialisasi, pemindaian tumpukan, penyedot debu indeks dan tumpukan, pembersihan indeks, pemotongan tumpukan, dan pembersihan akhir. Selama pemindaian tumpukan, proses memangkas halaman, mendefragmen, dan membekukannya. Setelah menyelesaikan pemindaian heap, VACUUM membersihkan indeks, mengembalikan halaman kosong ke sistem operasi, dan melakukan tugas pembersihan akhir seperti menyedot debu peta ruang kosong dan memperbarui statistik.

Penyedot debu indeks mungkin memerlukan beberapa lintasan ketika `maintenance_work_mem` (atau`autovacuum_work_mem`) tidak cukup untuk memproses indeks. Di PostgreSQL 16 dan sebelumnya, batas memori 1 GB untuk menyimpan IDs tupel mati sering memaksa beberapa pass pada indeks besar. PostgreSQL 17 `TidStore` memperkenalkan, yang secara dinamis mengalokasikan memori alih-alih menggunakan array alokasi tunggal. Ini menghilangkan batasan 1 GB, menggunakan memori lebih efisien, dan mengurangi kebutuhan untuk beberapa pemindaian indeks per setiap indeks.

Indeks besar mungkin masih memerlukan beberapa pass di PostgreSQL 17 jika memori yang tersedia tidak dapat mengakomodasi seluruh pemrosesan indeks sekaligus. Biasanya, indeks yang lebih besar mengandung lebih banyak tupel mati yang membutuhkan banyak lintasan.

Mendeteksi operasi vakum yang lambat

`postgres_get_av_diag()`Fungsi ini dapat mendeteksi ketika operasi vakum berjalan lambat karena memori yang tidak mencukupi. Untuk informasi lebih lanjut tentang fungsi ini, lihatMenginstal alat pemantauan dan diagnostik autovacuum di RDS untuk PostgreSQL Aurora .

`postgres_get_av_diag()`Fungsi mengeluarkan pemberitahuan berikut ketika memori yang tersedia tidak cukup untuk menyelesaikan penyedot debu indeks dalam satu lintasan.

**`rds_tools`**1.9

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is XX might not be sufficient.
 Consider increasing the setting to XXX, and if necessary, scaling up the RDS instance
 class for more
```

```
       memory. The suggested value is an estimate based on the current number of dead
tuples for the table being vacuumed, which might not fully reflect the latest state.
Additionally, review the possibility of manual
       vacuum with exclusion of indexes using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE
TRUE) table_name;). For more information, see

       Working with PostgreSQL autovacuum in the Amazon Aurora User Guide.
```

> **Note**
>
> `postgres_get_av_diag()`Fungsi ini bergantung pada `pg_stat_all_tables.n_dead_tup` untuk memperkirakan jumlah memori yang diperlukan untuk penyedot debu indeks.

Ketika `postgres_get_av_diag()` fungsi mengidentifikasi operasi vakum lambat yang memerlukan beberapa pemindaian indeks karena tidak mencukupi`autovacuum_work_mem`, itu akan menghasilkan pesan berikut:

```
NOTICE: Your vacuum is performing multiple index scans due to insufficient
 autovacuum_work_mem:XXX for index vacuuming.
       For more information, see Working with PostgreSQL autovacuum in the Amazon
 Amazon RDS User Guide.
```

Bimbingan

Anda dapat menerapkan solusi berikut menggunakan manual `VACUUM FREEZE` untuk mempercepat pembekuan tabel.

Tingkatkan memori untuk menyedot debu

Seperti yang disarankan oleh `postgres_get_av_diag()` fungsi, disarankan untuk meningkatkan `autovacuum_work_mem` parameter untuk mengatasi kendala memori potensial pada tingkat instance. Meskipun `autovacuum_work_mem` merupakan parameter dinamis, penting untuk dicatat bahwa agar pengaturan memori baru diterapkan, daemon autovacuum perlu me-restart pekerjanya. Untuk mencapai ini:

1. Konfirmasikan bahwa pengaturan baru sudah ada.
2. Hentikan proses yang saat ini menjalankan autovacuum.

Pendekatan ini memastikan bahwa alokasi memori yang disesuaikan diterapkan pada operasi autovacuum baru.

Untuk hasil yang lebih cepat, pertimbangkan untuk melakukan `VACUUM FREEZE` operasi secara manual dengan `maintenance_work_mem` pengaturan yang ditingkatkan dalam sesi Anda:

```
SET maintenance_work_mem TO '1GB';
VACUUM FREEZE VERBOSE table_name;
```

Jika Anda menggunakan Amazon RDS dan menemukan bahwa Anda memerlukan memori tambahan untuk mendukung nilai yang lebih tinggi untuk `maintenance_work_mem` atau`autovacuum_work_mem`, pertimbangkan untuk meningkatkan ke kelas instans dengan lebih banyak memori. Ini dapat menyediakan sumber daya yang diperlukan untuk meningkatkan operasi vakum manual dan otomatis, yang mengarah pada peningkatan kinerja vakum dan basis data secara keseluruhan.

Nonaktifkan INDEX_CLEANUP

Manual `VACUUM` di PostgreSQL versi 12 dan yang lebih baru memungkinkan melewatkan fase pembersihan indeks, sementara autovacuum darurat di PostgreSQL versi 14 dan yang lebih baru melakukan ini secara otomatis berdasarkan parameter. `vacuum_failsafe_age`

> **Warning**
>
> Melewatkan pembersihan indeks dapat menyebabkan indeks kembung dan berdampak negatif pada kinerja kueri. Untuk mengurangi ini, pertimbangkan untuk mengindeks ulang atau menyedot debu indeks yang terpengaruh selama jendela pemeliharaan.

Untuk panduan tambahan tentang penanganan indeks besar, lihat dokumentasi diMengelola autovacuum dengan indeks berukuran besar .

Penyedot debu indeks paralel

Dimulai dengan PostgreSQL 13, indeks dapat disedot dan dibersihkan secara paralel secara default menggunakan `VACUUM` manual, dengan satu proses vacuum worker ditetapkan untuk setiap indeks. Namun, untuk PostgreSQL untuk menentukan apakah operasi vakum memenuhi syarat untuk eksekusi paralel, kriteria khusus harus dipenuhi:

- Setidaknya harus ada dua indeks.

- `max_parallel_maintenance_workers`Parameter harus diatur ke setidaknya 2.
- Ukuran indeks harus melebihi `min_parallel_index_scan_size` batas, yang defaultnya 512KB.

Anda dapat menyesuaikan `max_parallel_maintenance_workers` pengaturan berdasarkan jumlah v yang CPUs tersedia di instans Amazon RDS Anda dan jumlah indeks di atas meja untuk mengoptimalkan waktu penyelesaian vakum.

Untuk informasi selengkapnya, lihat Penyedot debu paralel di Amazon RDS untuk PostgreSQL dan Amazon Aurora PostgreSQL.

Terlalu banyak tabel atau database untuk vakum

Seperti disebutkan dalam dokumentasi The Autovacuum Daemon PostgreSQL, daemon autovacuum beroperasi melalui beberapa proses. Ini termasuk peluncur autovacuum persisten yang bertanggung jawab untuk memulai proses pekerja autovacuum untuk setiap database dalam sistem. Peluncur menjadwalkan pekerja ini untuk memulai kira-kira setiap `autovacuum_naptime` detik per database.

Dengan database 'N', pekerja baru memulai kira-kira setiap [`autovacuum_naptime`/N detik]. Namun, jumlah total pekerja bersamaan dibatasi oleh `autovacuum_max_workers` pengaturan. Jika jumlah database atau tabel yang membutuhkan penyedot debu melebihi batas ini, database atau tabel berikutnya akan diproses segera setelah pekerja tersedia.

Ketika banyak tabel besar atau database memerlukan penyedot debu secara bersamaan, semua pekerja autovacuum yang tersedia dapat menjadi sibuk untuk durasi yang lama, menunda pemeliharaan pada tabel dan database lain. Di lingkungan dengan tingkat transaksi tinggi, kemacetan ini dapat dengan cepat meningkat dan berpotensi menyebabkan masalah vakum sampul dalam instans Amazon RDS Anda.

Ketika `postgres_get_av_diag()` mendeteksi sejumlah besar tabel atau database, ia memberikan rekomendasi berikut:

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_max_workers:3 might not be sufficient.
 Consider increasing the setting and, if necessary, consider scaling up the Amazon RDS
 instance class for more workers.
```

Bimbingan

Tingkatkan autovacuum_max_workers

Untuk mempercepat penyedotan debu, kami sarankan untuk menyesuaikan `autovacuum_max_workers` parameter untuk memungkinkan lebih banyak pekerja autovacuum bersamaan. Jika kemacetan kinerja tetap ada, pertimbangkan untuk meningkatkan instans Amazon RDS Anda ke kelas dengan lebih banyak v, yang selanjutnya dapat meningkatkan kemampuan pemrosesan CPUs paralel.

Vakum agresif (untuk mencegah pembungkus) sedang berjalan

Usia database (MaximumUsedTransactionIDs) di PostgreSQL hanya berkurang ketika vakum agresif (untuk mencegah sampul) berhasil diselesaikan. Sampai kekosongan ini selesai, usia akan terus meningkat tergantung pada tingkat transaksi.

`postgres_get_av_diag()`Fungsi ini menghasilkan yang berikut `NOTICE` ketika mendeteksi vakum yang agresif. Namun, itu hanya memicu output ini setelah vakum telah aktif setidaknya selama dua menit.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

Untuk informasi lebih lanjut tentang vakum agresif, lihat Saat vakum agresif sudah berjalan.

Anda dapat memverifikasi apakah vakum agresif sedang berlangsung dengan kueri berikut:

```
SELECT
    a.xact_start AS start_time,
    v.datname "database",
    a.query,
    a.wait_event,
    v.pid,
    v.phase,
    v.relid::regclass,
    pg_size_pretty(pg_relation_size(v.relid)) AS heap_size,
    (
        SELECT
            string_agg(pg_size_pretty(pg_relation_size(i.indexrelid)) || ':' ||
 i.indexrelid::regclass || chr(10), ', ')
        FROM
```

```
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS index_sizes,
    trunc(v.heap_blks_scanned * 100 / NULLIF(v.heap_blks_total, 0)) AS step1_scan_pct,
    v.index_vacuum_count || '/' || (
        SELECT
            count(*)
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS step2_vacuum_indexes,
    trunc(v.heap_blks_vacuumed * 100 / NULLIF(v.heap_blks_total, 0)) AS
 step3_vacuum_pct,
    age(CURRENT_TIMESTAMP, a.xact_start) AS total_time_spent_sofar
FROM
    pg_stat_activity a
    INNER JOIN pg_stat_progress_vacuum v ON v.pid = a.pid;
```

Anda dapat menentukan apakah itu vakum agresif (untuk mencegah sampul) dengan memeriksa kolom kueri di output. Ungkapan “untuk mencegah sampul” menunjukkan bahwa itu adalah kekosongan yang agresif.

```
query                 | autovacuum: VACUUM public.t3 (to prevent wraparound)
```

Misalnya, Anda memiliki pemblokir pada usia transaksi 1 miliar dan tabel yang membutuhkan vakum agresif untuk mencegah pemblokiran pada usia transaksi yang sama. Selain itu, ada pemblokir lain pada usia transaksi 750 juta. Setelah menyelesaikan pemblokir pada usia transaksi 1 miliar, usia transaksi tidak akan langsung turun menjadi 750 juta. Ini akan tetap tinggi sampai tabel yang membutuhkan vakum agresif atau transaksi apa pun dengan usia di atas 750 juta selesai. Selama periode ini, usia transaksi klaster PostgreSQL Anda akan terus meningkat. Setelah proses vakum selesai, usia transaksi akan turun menjadi 750 juta tetapi akan mulai meningkat lagi hingga penyedotan lebih lanjut selesai. Siklus ini akan berlanjut selama kondisi ini tetap ada, hingga usia transaksi akhirnya turun ke level yang dikonfigurasi untuk instans Amazon RDS Anda, yang ditentukan oleh. `autovacuum_freeze_max_age`

## Penjelasan pesan PEMBERITAHUAN di

`postgres_get_av_diag()`Fungsi ini menyediakan pesan PEMBERITAHUAN berikut:

Ketika usia belum mencapai ambang pemantauan

Ambang batas pemantauan `postgres_get_av_diag()` untuk mengidentifikasi pemblokir adalah 500 juta transaksi secara default. Jika `postgres_get_av_diag()` menghasilkan PEMBERITAHUAN berikut, ini menunjukkan bahwa usia transaksi belum mencapai ambang batas ini.

```
NOTICE: postgres_get_av_diag() checks for blockers that prevent aggressive vacuums
 only, it does so only after exceeding dvb_threshold which is 500,000,000 and age of
 this PostgreSQL cluster is currently at 2.
```

Tidak terhubung ke database dengan usia ID transaksi tertua

`postgres_get_av_diag()`Fungsi ini memberikan output paling akurat saat terhubung ke database dengan usia ID transaksi tertua. Database dengan usia ID transaksi tertua yang dilaporkan oleh `postgres_get_av_diag()` akan berbeda dari “my_database” dalam kasus Anda. Jika Anda tidak terhubung ke database yang benar, PEMBERITAHUAN berikut akan dibuat:

```
NOTICE: You are not connected to the database with the age of oldest transaction
 ID. Connect to my_database database and run postgres_get_av_diag() for accurate
 reporting.
```

Menghubungkan ke database dengan usia transaksi tertua penting karena alasan berikut:

- Mengidentifikasi pemblokir tabel sementara: Karena metadata untuk tabel sementara khusus untuk setiap database, mereka biasanya ditemukan dalam database tempat mereka dibuat. Namun, jika tabel sementara kebetulan menjadi pemblokir teratas dan berada di database dengan transaksi tertua, ini bisa menyesatkan. Menghubungkan ke database yang benar memastikan identifikasi akurat dari pemblokir tabel sementara.
- Mendiagnosis vakum lambat: Metadata indeks dan informasi jumlah tabel bersifat spesifik basis data dan diperlukan untuk mendiagnosis masalah vakum lambat.

Database dengan transaksi tertua berdasarkan usia ada pada database rdsadmin atau template0

Dalam kasus tertentu, `rdsadmin` atau `template0` database dapat diidentifikasi sebagai database dengan usia ID transaksi tertua. Jika ini terjadi, `postgres_get_av_diag()` akan mengeluarkan PEMBERITAHUAN berikut:

```
NOTICE: The database with the age of oldest transaction ID is rdsadmin or template0,
 reach out to support if the reported blocker is in rdsadmin or template0.
```

Verifikasi bahwa pemblokir yang terdaftar tidak berasal dari salah satu dari dua database ini. Jika pemblokir dilaporkan ada di salah satu `rdsadmin` atau`template0`, hubungi dukungan karena basis data ini tidak dapat diakses pengguna dan memerlukan intervensi.

Sangat tidak mungkin bagi `template0` database `rdsadmin` atau untuk berisi pemblokir teratas.

Saat vakum agresif sudah berjalan

`postgres_get_av_diag()`Fungsi ini dirancang untuk melaporkan ketika proses vakum agresif berjalan, tetapi hanya memicu keluaran ini setelah vakum aktif setidaknya selama 1 menit. Penundaan yang disengaja ini membantu mengurangi kemungkinan positif palsu. Dengan menunggu, fungsi memastikan bahwa hanya penyedot debu yang efektif dan signifikan yang dilaporkan, yang mengarah pada pemantauan aktivitas vakum yang lebih akurat dan andal.

`postgres_get_av_diag()`Fungsi ini menghasilkan PEMBERITAHUAN berikut ketika mendeteksi satu atau lebih vakum agresif yang sedang berlangsung.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

Sebagaimana ditunjukkan dalam PEMBERITAHUAN, terus memantau kinerja vakum. Untuk informasi lebih lanjut tentang vakum agresif lihat Vakum agresif (untuk mencegah pembungkus) sedang berjalan

Saat autovacuum dimatikan

`postgres_get_av_diag()`Fungsi ini menghasilkan PEMBERITAHUAN berikut jika autovacuum dinonaktifkan pada instance database Anda:

```
NOTICE: Autovacuum is OFF, we strongly recommend to enable it, no restart is
 necessary.
```

Autovacuum adalah fitur penting dari instance yang memastikan kelancaran operasi database. Ini secara otomatis menghapus versi baris lama, merebut kembali ruang penyimpanan, dan mencegah kembung tabel, membantu menjaga tabel dan indeks tetap efisien untuk kinerja optimal. Selain itu, ini melindungi terhadap sampul ID transaksi, yang dapat menghentikan transaksi di instans Amazon RDS Anda. Menonaktifkan autovacuum dapat menyebabkan

penurunan jangka panjang dalam kinerja dan stabilitas database. Kami menyarankan Anda untuk tetap menggunakannya setiap saat. Untuk informasi lebih lanjut, lihat Memahami autovacuum di .

> **Note**
>
> Mematikan autovacuum tidak menghentikan penyedot debu agresif. Ini masih akan terjadi setelah tabel Anda mencapai `autovacuum_freeze_max_age` ambang batas.

Jumlah transaksi yang tersisa sangat rendah

`postgres_get_av_diag()`Fungsi ini menghasilkan PEMBERITAHUAN berikut ketika vakum sampul sudah dekat. PEMBERITAHUAN ini dikeluarkan ketika instans Amazon RDS Anda berjarak 100 juta transaksi dari kemungkinan menolak transaksi baru.

```
WARNING: Number of transactions remaining is critically low, resolve issues with
 autovacuum or perform manual VACUUM FREEZE before your instance stops accepting
 transactions.
```

Tindakan segera Anda diperlukan untuk menghindari downtime database. Anda harus memonitor operasi penyedot debu Anda dan mempertimbangkan secara manual memulai database yang `VACUUM FREEZE` terpengaruh untuk mencegah kegagalan transaksi.

# Menggunakan Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL memperluas klaster DB Aurora PostgreSQL Anda dengan kemampuan untuk menerima koneksi basis data dari klien SQL Server. Dengan Babelfish, aplikasi yang awalnya dibuat untuk SQL Server dapat bekerja secara langsung dengan Aurora PostgreSQL dengan sedikit perubahan kode dibandingkan dengan migrasi tradisional dan tanpa mengubah driver basis data. Untuk informasi selengkapnya tentang bermigrasi, lihat Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL.

Babelfish menyediakan titik akhir tambahan untuk klaster basis data Aurora PostgreSQL yang memungkinkannya untuk memahami protokol tingkat kabel SQL Server dan pernyataan SQL Server yang umum digunakan. Aplikasi klien yang menggunakan protokol kabel Tabular Data Stream (TDS) dapat terhubung secara native ke port pendengar TDS di Aurora PostgreSQL. Untuk mempelajari lebih lanjut tentang TDS, lihat [MS-TDS]: Tabular Data Stream Protocol di situs web Microsoft.

> **Note**
>
> Babelfish for Aurora PostgreSQL mendukung TDS versi 7.1 hingga 7.4.

Babelfish juga menyediakan akses ke data menggunakan koneksi PostgreSQL. Secara default, kedua dialek SQL yang didukung oleh Babelfish tersedia melalui protokol kabel aslinya di port berikut:

- Dialek SQL Server (T-SQL), klien terhubung ke port 1433.
- Dialek PostgreSQL (PL/pgSQL), klien terhubung ke port 5432.

Babelfish menjalankan bahasa Transact-SQL (T-SQL) dengan beberapa perbedaan. Untuk informasi selengkapnya, lihat Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server.

Pada bagian berikut, Anda dapat menemukan informasi tentang pengaturan dan penggunaan klaster DB Babelfish for Aurora PostgreSQL.

Topik


- Batasan Babelfish
- Memahami konfigurasi dan arsitektur Babelfish
- Membuat klaster DB Babelfish for Aurora PostgreSQL
- Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL

- Autentikasi basis data dengan Babelfish for Aurora PostgreSQL
- Menghubungkan ke klaster DB Babelfish
- Menggunakan Babelfish
- Memecahkan Masalah Babelfish
- Menonaktifkan Babelfish
- Mengelola Babelfish untuk pembaruan versi Aurora Postgre SQL
- Referensi Babelfish for Aurora PostgreSQL

# Batasan Babelfish

Batasan berikut saat ini berlaku untuk Babelfish for Aurora PostgreSQL:

- Saat memutakhirkan, Anda mungkin melihat perbedaan kepemilikan objek dalam skema yang sama. Objek pra-upgrade dimiliki oleh pengguna saat ini sementara objek pasca-upgrade dimiliki oleh pemilik skema, yang mungkin merupakan pengguna yang berbeda. Untuk mengatasi hal ini, kami memperkenalkan `sys.generate_alter_ownership_statements()` fungsi.

  Untuk memperbaiki ketidakcocokan kepemilikan objek, sambungkan ke klaster Anda menggunakan titik akhir PostgreSQL, jalankan fungsi sys.generate_alter_ownership_statements (), dan jalankan pernyataan SQL yang dihasilkan.

  Pertimbangkan batasan penting ini terkait dengan perubahan kepemilikan objek:

  - Pengguna yang diberikan izin CREATE melalui titik akhir PostgreSQL tidak dapat membuat objek melalui titik akhir TDS. Kami tidak menyarankan mengubah izin pada objek T-SQL melalui titik akhir PostgreSQL karena dapat menyebabkan perilaku T-SQL yang salah.
  - Izin akses objek mungkin berubah. Misalnya, jika skema yang dimiliki oleh 'sch_own' berisi objek yang dimiliki oleh 'dbo' sebelum upgrade, kemampuan pengguna untuk melakukan operasi seperti SELECT atau INSERT mungkin berbeda antara objek pra-upgrade (dimiliki oleh 'dbo') dan objek pasca-upgrade (dimiliki oleh 'sch_own').
- Babelfish saat ini tidak mendukung fitur Aurora berikut ini:
  - AWS Identity and Access Management
  - Stream Aktivitas Basis Data (DAS)
  - RDS Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan
  - Proksi RDS dengan RDS for SQL Server
  - Salted Challenge Response Authentication Mechanism (SCRAM)
  - Editor kueri
- Babelfish tidak menyediakan dukungan API driver klien berikut ini:
  - Permintaan API dengan atribut koneksi yang terkait dengan Microsoft Distributed Transaction Coordinator (MSDTC) tidak didukung. Ini termasuk panggilan XA oleh SQLServer XAResource kelas di server SQL JDBC driver.
- Babelfish saat ini tidak mendukung ekstensi Aurora PostgreSQL berikut:
  - `bloom`
  - `btree_gin`

- `btree_gist`
- `citext`
- `cube`
- `hstore`
- `hypopg`
- Replikasi logis menggunakan `pglogical`
- `ltree`
- `pgcrypto`
- Manajemen rencana kueri menggunakan `apg_plan_mgmt`

Untuk mempelajari lebih lanjut tentang ekstensi PostgreSQL, lihat. Menangani ekstensi dan pembungkus data asing

- Driver jTDS sumber terbuka yang dirancang sebagai alternatif driver JDBC Microsoft tidak didukung.

## Memahami konfigurasi dan arsitektur Babelfish

Anda mengelola klaster DB Edisi yang Kompatibel dengan Aurora PostgreSQL yang menjalankan Babelfish seperti halnya klaster DB Aurora. Artinya, Anda mendapat manfaat dari skalabilitas, ketersediaan tinggi dengan dukungan failover, dan replikasi default yang disediakan oleh klaster DB Aurora. Untuk mempelajari lebih lanjut tentang kapabilitas ini, lihat Mengelola performa dan penskalaan untuk klaster DB Aurora, Ketersediaan yang tinggi untuk Amazon Aurora, dan Replikasi dengan Amazon Aurora. Anda juga memiliki akses ke banyak AWS alat dan utilitas lain, termasuk yang berikut ini:

- Amazon CloudWatch adalah layanan pemantauan dan observabilitas yang memberi Anda data dan wawasan yang dapat ditindaklanjuti. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Amazon CloudWatch.
- Wawasan Performa adalah fitur penyetelan dan pemantauan performa basis data yang membantu Anda menilai beban basis data dengan cepat. Untuk mempelajari selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Basis data global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan memberikan pemulihan cepat dari pemadaman langka yang mungkin

memengaruhi keseluruhan. Wilayah AWS Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

- Penambalan perangkat lunak otomatis menjaga database Anda up-to-date dengan keamanan terbaru dan tambalan fitur saat tersedia.
- Peristiwa Amazon RDS memberi tahu Anda melalui email atau pesan SMS tentang peristiwa basis data penting, seperti failover otomatis. Untuk informasi selengkapnya, lihat .

Berikut ini, Anda dapat mempelajari tentang arsitektur Babelfish dan bagaimana basis data SQL Server yang Anda migrasi ditangani oleh Babelfish. Saat membuat klaster DB Babelfish, Anda perlu membuat beberapa keputusan lebih awal tentang satu basis data atau beberapa basis data, kolasi, dan detail lainnya.

Topik

- Arsitektur Babelfish
- Pengaturan grup parameter klaster DB untuk Babelfish
- Memahami Kolasi di Babelfish untuk Aurora Postgre SQL
- Mengelola penanganan kesalahan Babelfish dengan escape hatch

## Arsitektur Babelfish

Saat Anda membuat klaster Aurora PostgreSQL dengan Babelfish diaktifkan, Aurora menyediakan klaster dengan basis data PostgreSQL bernama `babelfish_db`. Basis data ini adalah tempat semua objek dan struktur SQL Server yang dimigrasi berada.

> **Note**
>
> Dalam klaster Aurora PostgreSQL, nama basis data `babelfish_db` dicadangkan untuk Babelfish. Membuat basis data "babelfish_db" Anda sendiri pada klaster Babelfish DB mencegah Aurora berhasil menyediakan Babelfish.

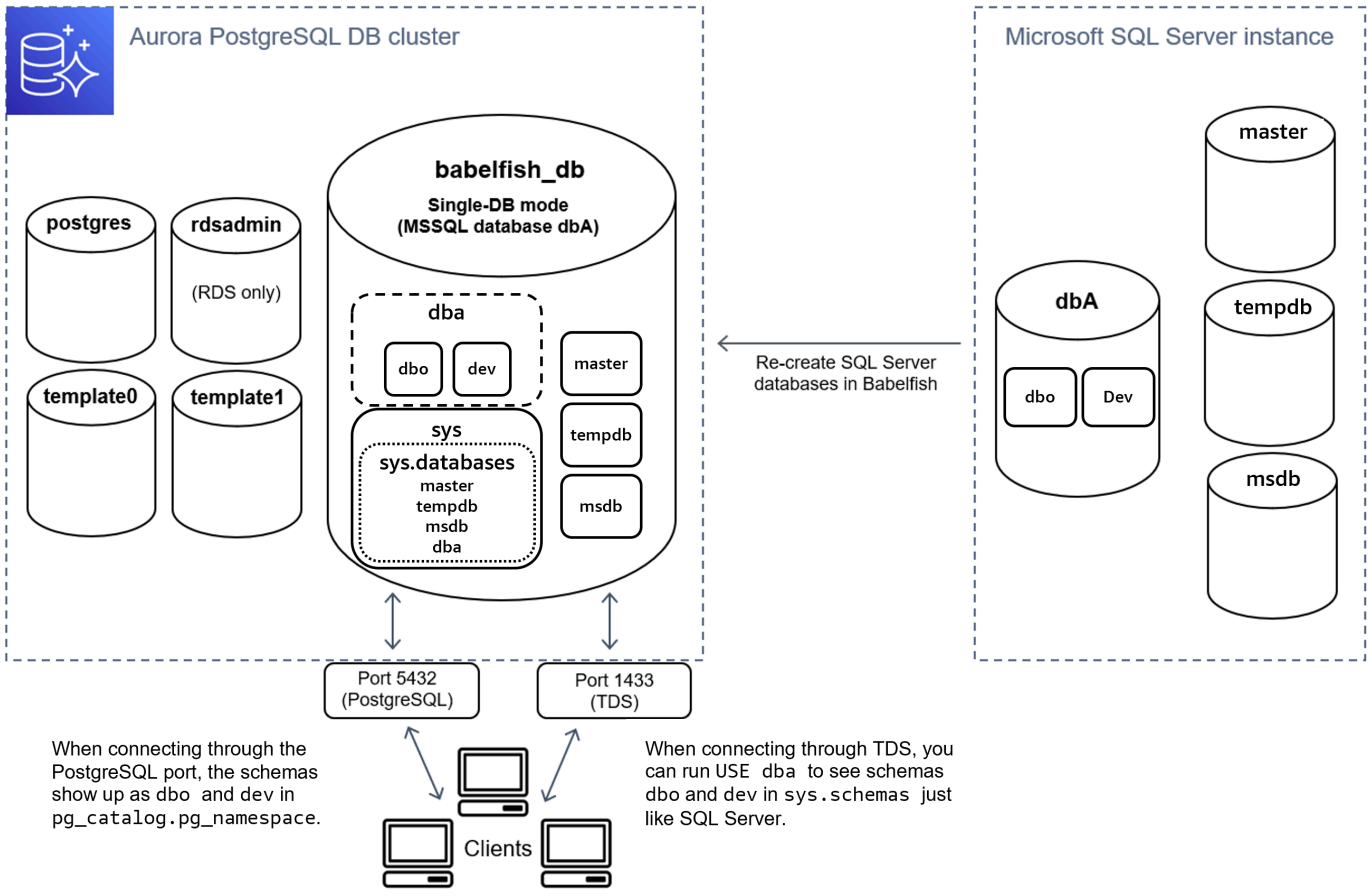
Saat Anda terhubung ke port TDS, sesi ditempatkan pada basis data `babelfish_db`. Dari T-SQL, strukturnya terlihat mirip dengan kondisi saat terhubung ke instans SQL Server. Anda dapat melihat basis data `master`, `msdb`, dan `tempdb`, serta katalog `sys.databases`. Anda dapat membuat basis data pengguna tambahan dan beralih antar-basis data dengan pernyataan USE. Ketika Anda

membuat basis data pengguna SQL Server, basis data tersebut akan diratakan ke dalam basis data PostgreSQL `babelfish_db`. Basis data Anda mempertahankan sintaks lintas-basis data dan semantik yang setara atau mirip dengan yang disediakan oleh SQL Server.

Menggunakan Babelfish dengan satu atau beberapa basis data

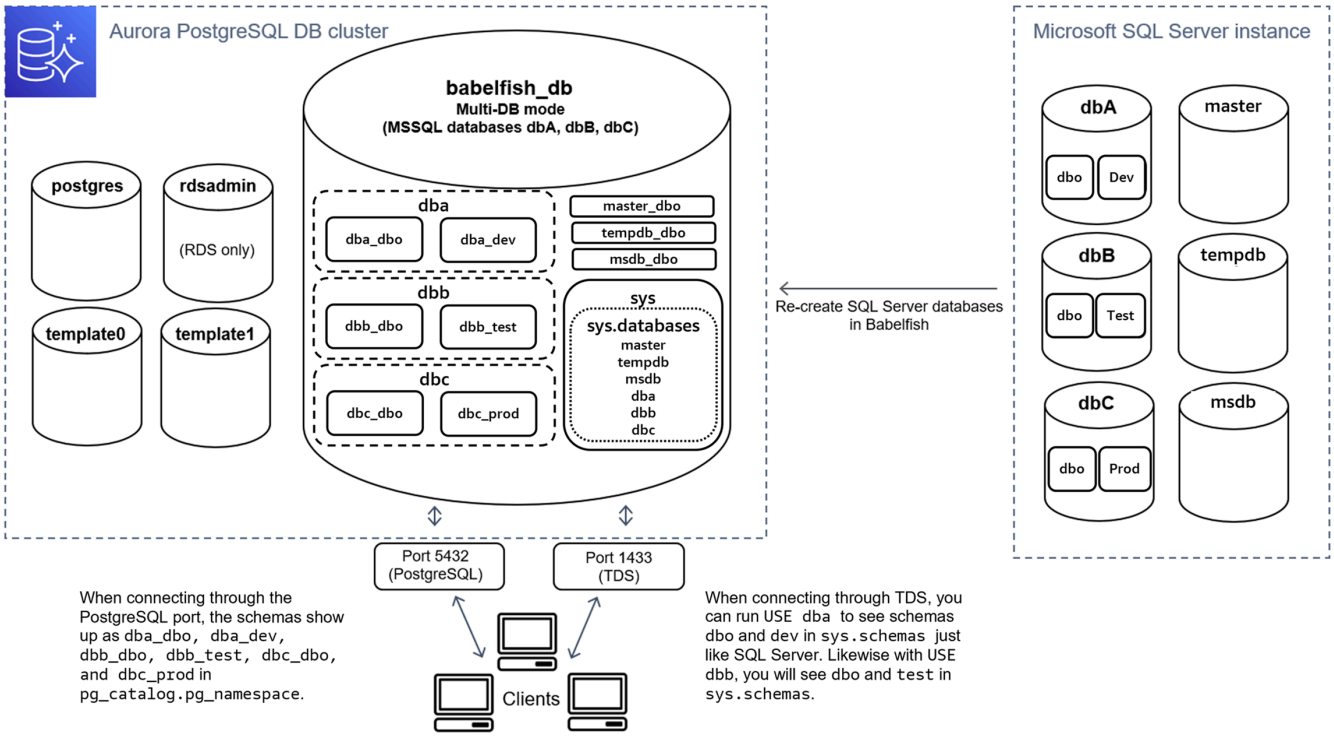
Saat Anda membuat klaster Aurora PostgreSQL untuk digunakan dengan Babelfish, Anda memilih antara menggunakan satu basis data SQL Server atau beberapa basis data SQL Server sekaligus. Pilihan Anda memengaruhi bagaimana nama skema SQL Server di dalam basis data `babelfish_db` muncul dari Aurora PostgreSQL. Mode migrasi disimpan dalam parameter `migration_mode`. Anda tidak boleh mengubah parameter ini setelah membuat klaster karena Anda bisa kehilangan akses ke semua objek SQL yang telah dibuat sebelumnya.

Dalam mode basis data tunggal, nama skema basis data SQL Server tetap sama dalam basis data `babelfish_db` PostgreSQL. Jika Anda memilih untuk hanya memigrasikan basis data tunggal, nama skema basis data pengguna yang dimigrasi dapat dirujuk di PostgreSQL dengan nama yang sama seperti di SQL Server. Misalnya, skema `dbo` dan `smith` berada dalam basis data `dbA` .



Saat menghubungkan melalui TDS, Anda dapat menjalankan `USE dba` untuk melihat skema dbo dan dev dari T-SQL, seperti yang Anda lakukan di SQL Server. Nama skema yang tidak berubah juga terlihat dari PostgreSQL.

Dalam modus multi-basis data, nama skema basis data pengguna menjadi `dbname_schemaname` ketika diakses dari PostgreSQL. Nama skema tetap sama ketika diakses dari T-SQL.




Seperti yang ditunjukkan pada gambar, mode multi-basis data dan mode basis data tunggal sama dengan SQL Server saat menghubungkan melalui port TDS dan menggunakan T-SQL. Misalnya, `USE dbA` mencantumkan skema dbo dan dev seperti halnya di SQL Server. Nama skema yang dipetakan, seperti `dba_dbo` dan `dba_dev`, terlihat dari PostgreSQL.

Setiap basis data masih berisi skema Anda. Nama setiap basis data ditambahkan ke nama skema SQL Server, menggunakan garis bawah sebagai pembatas, misalnya:

- `dba` berisi `dba_dbo` dan `dba_dev`.
- `dbb` berisi `dbb_dbo` dan `dbb_test`.
- `dbc` berisi `dbc_dbo` dan `dbc_prod`.

Di dalam basis data `babelfish_db`, pengguna T-SQL masih perlu menjalankan `USE dbname` untuk mengubah konteks basis data, sehingga tampilan dan nuansanya tetap mirip dengan SQL Server.

## Memilih mode migrasi

Setiap mode migrasi memiliki kelebihan dan kekurangan. Pilih mode migrasi berdasarkan jumlah basis data pengguna milik Anda, dan rencana migrasi Anda. Setelah Anda membuat klaster untuk digunakan dengan Babelfish, Anda tidak boleh mengubah mode migrasi karena Anda mungkin akan kehilangan akses ke semua objek SQL yang dibuat sebelumnya. Saat memilih mode migrasi, pertimbangkan persyaratan basis data dan klien pengguna Anda.

Saat Anda membuat klaster untuk digunakan dengan Babelfish, Aurora PostgreSQL membuat basis data sistem, `master` dan `tempdb`. Jika Anda membuat atau memodifikasi objek dalam basis data sistem (`master` atau `tempdb`), pastikan untuk membuat ulang objek tersebut di klaster baru. Tidak seperti SQL Server, Babelfish tidak menginisialisasi ulang `tempdb` setelah klaster mem-boot ulang.

Gunakan mode migrasi basis data tunggal dalam kasus berikut:

- Jika Anda memigrasi satu basis data SQL Server. Dalam mode basis data tunggal, nama skema yang dimigrasi saat diakses dari PostgreSQL identik dengan nama skema SQL Server asli. Hal ini akan mengurangi perubahan kode pada kueri SQL yang ada jika Anda ingin mengoptimalkannya untuk dijalankan dengan koneksi PostgreSQL.
- Jika tujuan akhir Anda adalah migrasi lengkap ke Aurora PostgreSQL asli. Sebelum bermigrasi, konsolidasikan skema Anda ke dalam satu skema (`dbo`) dan kemudian migrasikan ke dalam satu klaster untuk mengurangi perubahan yang diperlukan.

Gunakan beberapa mode migrasi basis data dalam kasus berikut:

- Jika Anda menginginkan pengalaman SQL Server default dengan beberapa basis data pengguna dalam instans yang sama.
- Jika beberapa basis data pengguna perlu dimigrasikan bersama.

## Pengaturan grup parameter klaster DB untuk Babelfish

Saat Anda membuat klaster DB Aurora PostgreSQL dan memilih Aktifkan Babelfish, grup parameter klaster DB akan dibuat untuk Anda secara otomatis jika Anda memilih Buat baru. Grup parameter klaster DB ini didasarkan pada grup parameter klaster Aurora PostgreSQL DB untuk versi Aurora PostgreSQL yang dipilih untuk instalasi, misalnya, Aurora PostgreSQL versi 14. Grup ini dinamai menggunakan pola umum berikut:

```
custom-aurora-postgresql14-babelfish-compat-3
```

Anda dapat mengubah pengaturan berikut selama proses pembuatan klaster tetapi beberapa di antaranya tidak dapat diubah setelah disimpan di grup parameter kustom, jadi pilih dengan cermat:

- Basis data tunggal atau Multibasis data
- Lokal kolasi default
- Nama kolasi
- Grup parameter DB

Untuk menggunakan klaster Aurora PostgreSQL DB versi 13 atau grup parameter yang lebih tinggi, edit grup dan atur parameter `babelfish_status` ke on. Tentukan opsi Babelfish sebelum membuat klaster Aurora PostgreSQL Anda. Untuk mempelajari selengkapnya, lihat .

Parameter berikut mengontrol preferensi Babelfish. Kecuali dinyatakan lain dalam Deskripsi, parameter dapat dimodifikasi. Nilai default disertakan dalam deskripsi. Untuk melihat nilai yang diizinkan untuk parameter apa pun, lakukan hal berikut:

> **Note**
>
> Ketika Anda mengaitkan grup parameter DB baru dengan instans DB, parameter statis dan dinamis yang dimodifikasi akan diterapkan hanya setelah instans DB di-boot ulang. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini diterapkan segera tanpa boot ulang.

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Grup parameter dari menu navigasi.

3. Pilih grup parameter klaster DB `default.aurora-postgresql14` dari daftar.
4. Masukkan nama parameter di kolom pencarian. Misalnya, masukkan `babelfishpg_tsql.default_locale` di kolom pencarian untuk menampilkan parameter ini dan nilai default-nya serta pengaturan yang diizinkan.

> **Note**
>
> Basis data global Babelfish untuk Aurora PostgreSQL bekerja di wilayah sekunder hanya jika parameter berikut dihidupkan di wilayah tersebut.

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
|---|---|---|---|
| babelfishpg_tsql.apg_enable_correlated_scalar_transform | Memungkinkan perencana untuk mengubah subquery skalar berkorelasi di Babelfish. (Default: on) (Diijinkan: on, off) | dinamis | true |
| babelfishpg_tsql.apg_enable_subquery_cache | Mengaktifkan penggunaan cache untuk subquery skalar berkorelasi di Babelfish. (Default: on) (Diijinkan: on, off) | dinamis | true |
| babelfishpg_tds.tds_default_numeric_scale | Menetapkan skala default tipe numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya. (Default: 8) (Diizinkan: 0–38) | dinamis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tds.tds_default_numeric_precision | Integer yang menetapkan presisi default tipe numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya. (Default: 38) (Diizinkan: 1–38) | dinamis | true |
| babelfishpg_tds.tds_default_packet_size | Integer yang menetapkan ukuran paket default untuk menghubungkan klien SQL Server. (Default: 4096) (Diizinkan: 512–32767) | dinamis | true |
| babelfishpg_tds.tds_default_protocol_version | Integer yang menetapkan versi protokol TDS default untuk menghubungkan klien. (Default: DEFAULT) (Diijinkan: TDSv7 .0, TDSv7 .1, TDSv7 .1.1, TDSv7 .2, TDSv7 .3A, .3B, .4, DEFAULT) TDSv7 TDSv7 | dinamis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tds.default_server_name | String yang mengidentifikasi nama default server Babelfish. (Default: Microsoft SQL Server) (Diizinkan: null) | dinamis | true |
| babelfishpg_tds.tds_debug_log_level | Integer yang menetapkan tingkat pencatatan log di TDS; 0 mematikan pencatatan log. (Default: 1) (Diizinkan: 0, 1, 2, 3) | dinamis | true |
| babelfishpg_tds.listen_addresses | String yang menetapkan nama host atau alamat IP atau alamat untuk mendengarkan TDS aktif. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: *) (Diizinkan: null) | – | false |
| babelfishpg_tds.port | Integer yang menentukan port TCP yang digunakan untuk permintaan dalam sintaks SQL Server. (Default: 1433) (Diizinkan: 1–65535) | statis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tds.tds_ssl_encrypt | Boolean yang mengaktifkan (0) atau menonaktifkan (1) enkripsi untuk data yang melintasi port pendengar TDS. Untuk informasi detail cara menggunakan SSL untuk koneksi klien, lihat Pengaturan SSL Babelfish dan koneksi klien. (Default: 0) (Diizinkan: 0, 1) | dinamis | true |
| babelfishpg_tds.tds_ssl_max_protocol_version | String yang menentukan versi SSL/TLS protokol tertinggi untuk digunakan untuk sesi TDS. (Default: 'TLSv1.2') (Diijinkan:' ',' TLSv1 TLSv1 .1', '.2') TLSv1 | dinamis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tds.tds_ssl_min_protocol_version | String yang menentukan versi SSL/TLS protokol minimum yang akan digunakan untuk sesi TDS. (Default: 'TLSv1.2' dari Aurora PostgreSQL versi 16,' 'untuk TLSv1 versi yang lebih lama dari Aurora PostgreSQL versi 16) (Diizinkan:' ',' .1', '.2') TLSv1 TLSv1 TLSv1 | dinamis | true |
| babelfishpg_tds.unix_socket_directories | String yang mengidentifikasi direktori soket Unix TDS. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: /tmp) (Diizinkan: null) | – | false |
| babelfishpg_tds.unix_socket_group | String yang mengidentifikasi grup soket Unix server TDS. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: rdsdb) (Diizinkan: null) | – | false |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tsql.default_locale | Sebuah string yang menentukan lokal default yang digunakan untuk koleksi Babelfish. Lokal default hanya lokal dan tidak menyertakan kualifikasi apa pun.<br><br>Tetapkan parameter ini saat Anda menyediakan klaster DB Babelfish. Setelah klaster DB disediakan, perubahan pada parameter ini diabaikan. (Default: en_US) (Diizinkan: Lihat tabel) | statis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tsql.migration_mode | Daftar yang tidak dapat dimodifikasi yang menentukan dukungan untuk basis data satu atau beberapa pengguna. Tetapkan parameter ini saat Anda menyediakan klaster DB Babelfish. Setelah klaster DB disediakan, Anda tidak dapat mengubah nilai parameter ini. (Default: multi-db dari Aurora PostgreSQL versi 16, single-db untuk versi yang lebih lama dari Aurora PostgreSQL versi 16) (Diizinkan: single-db, multi-db, null) | statis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| babelfishpg_tsql.server_collation_name | String yang menentukan nama pemeriksaan yang digunakan untuk tindakan tingkat server. Tetapkan parameter ini saat Anda menyediakan klaster DB Babelfish. Setelah klaster DB disediakan, jangan ubah nilai parameter ini. (Default: bbf_unicode_general_ci_as) (Diizinkan: Lihat tabel) | statis | true |
| babelfishpg_tsql.version | Sebuah string yang menetapkan output dari variabel @@VERSION. Jangan ubah nilai ini untuk klaster DB Aurora PostgreSQL. (Default: null) (Diizinkan: default) | dinamis | true |

| Parameter | Deskripsi | Tipe Penerapan | Dapat Dimodifikasi |
| --- | --- | --- | --- |
| rds.babelfish_status | String yang menetapkan status fungsionalitas Babelfish. Ketika parameter ini diatur ke `datatypesonly` , Babelfish dinonaktifkan tetapi tipe data SQL Server masih tersedia. (Default: nonaktif) (Diizinkan: aktif, nonaktif, khusus tipe data) | statis | true |
| unix_socket_permissions | Integer yang menetapkan izin soket Unix server TDS. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: 0700) (Diizinkan: 0–511) | – | false |

### Pengaturan SSL Babelfish dan koneksi klien

Untuk memerlukan SSL/TLS koneksi ke Babelfish Anda untuk klaster DB PostgreSQL Aurora, gunakan parameternya. `rds.force_ssl`

- Untuk memerlukan SSL/TLS koneksi, atur nilai `rds.force_ssl` parameter ke 1 (on).
- Untuk mematikan SSL/TLS koneksi yang diperlukan, atur nilai `rds.force_ssl` parameter ke 0 (off).

Nilai default parameter ini tergantung pada versi Aurora PostgreSQL:

- Untuk Aurora PostgreSQL versi 17 dan yang lebih baru: Nilai default adalah 1 (on).
- Untuk Aurora PostgreSQL versi 16 dan yang lebih lama: Nilai default adalah 0 (off).

> **Note**
>
> Saat Anda melakukan upgrade versi mayor dari Aurora PostgreSQL versi 16 atau versi sebelumnya ke versi 17 atau yang lebih baru, nilai default parameter berubah dari 0 (off) menjadi 1 (on). Perubahan ini dapat menyebabkan kegagalan konektivitas untuk aplikasi yang tidak dikonfigurasi untuk SSL. Anda dapat kembali ke perilaku default sebelumnya dengan menyetel parameter ini ke 0 (off).

Untuk detail spesifik pengemudi, lihat. Menghubungkan ke klaster DB Babelfish

Ketika klien terhubung ke port TDS (default `1433`), Babelfish membandingkan pengaturan Secure Sockets Layer (SSL) yang dikirim selama jabat tangan klien ke pengaturan parameter Babelfish SSL (`tds_ssl_encrypt`). Babelfish kemudian menentukan apakah koneksi diizinkan. Jika koneksi diizinkan, perilaku enkripsi diberlakukan atau tidak, tergantung pada pengaturan parameter Anda dan dukungan untuk enkripsi yang ditawarkan oleh klien.

Tabel berikut menunjukkan bagaimana Babelfish berperilaku untuk setiap kombinasi.

| Pengaturan SSL Klien | Pengaturan SSL Babelfish | rds.force_ssl | Koneksi diizinkan? | Nilai dikembalikan ke klien |
|---|---|---|---|---|
| ENCRYPT_ON | Setiap | Setiap | Diizinkan, seluruh koneksi dienkripsi | ENCRYPT_ON |
| ENCRYPT_OFF | tds_ssl_encrypt=1 | Setiap | Diizinkan, seluruh koneksi dienkripsi | ENCRYPT_REQ |
| ENCRYPT_OFF | tds_ssl_encrypt=0 | rds.force_ssl=0 | Diizinkan, paket login dienkripsi | ENCRYPT_OFF |
| ENCRYPT_OFF | tds_ssl_encrypt=0 | rds.force_ssl=1 | Tidak, koneksi ditutup | ENCRYPT_OFF |
| ENCRYPT_NOT_SUP | tds_ssl_encrypt=0 | rds.force_ssl=0 | Ya | ENCRYPT_NOT_SUP |
| ENCRYPT_NOT_SUP | tds_ssl_encrypt=1 | Setiap | Tidak, koneksi ditutup | ENCRYPT_REQ |
| ENCRYPT_NOT_SUP | tds_ssl_encrypt=0 | rds.force_ssl=1 | Tidak, koneksi ditutup | ENCRYPT_NOT_SUP |
| ENCRYPT_CLIENT_CERT | Setiap | Setiap | Tidak, koneksi ditutup | Tidak didukung |

# Memahami Kolasi di Babelfish untuk Aurora Postgre SQL

Saat Anda membuat cluster Aurora Postgre SQL DB dengan Babelfish, Anda memilih pemeriksaan untuk data Anda. Sebuah kolasi menentukan urutan dan pola bit yang menghasilkan teks atau karakter dalam bahasa manusia tertulis tertentu. Kolasi mencakup aturan yang membandingkan data untuk set pola bit tertentu. Kolasi terkait dengan lokalisasi. Lokal yang berbeda memengaruhi pemetaan karakter, urutan, dan sejenisnya. Atribut kolasi tercermin dalam nama berbagai kolasi. Untuk mengetahui informasi atribut, lihat Babelfish collation attributes table.

Babelfish memetakan koleksi SQL Server ke koleksi sebanding yang disediakan oleh Babelfish. Babelfish menentukan kolasi Unicode dengan urutan dan perbandingan string yang sensitif secara budaya. Babelfish juga menyediakan cara untuk menerjemahkan koleksi di SQL Server DB Anda ke koleksi Babelfish yang paling cocok. Kolasi khusus lokal disediakan untuk berbagai bahasa dan wilayah.

Beberapa kolasi menentukan halaman kode yang sesuai dengan pengodean sisi klien. Babelfish secara otomatis menerjemahkan dari pengodean server ke pengodean klien bergantung pada kolasi setiap kolom output.

Babelfish mendukung kolasi yang tercantum dalam Babelfish supported collations table. Babelfish memetakan koleksi SQL Server ke koleksi sebanding yang disediakan oleh Babelfish.

Babelfish menggunakan pustaka pengumpulan Komponen Internasional untuk Unicode () versi 153.80. ICU Untuk informasi selengkapnya tentang ICU susunan, lihat Pengumpulan dalam dokumentasi. ICU Untuk mempelajari lebih lanjut tentang Postgre SQL dan pemeriksaan, lihat Collation Support di dokumentasi Postgre. SQL

Topik

- Parameter klaster DB yang mengontrol kolasi dan lokal
- Kolasi deterministik dan nondeterministik di Babelfish
- Kolasi didukung pada tingkat database di Babelfish
- Server dan Objek Collations di Babelfish
- Perilaku Collation default di Babelfish
- Mengelola kolasi
- Batasan kolasi dan perbedaan perilaku

## Parameter klaster DB yang mengontrol kolasi dan lokal

Parameter berikut memengaruhi perilaku kolasi.

babelfishpg_tsql.default_locale

Parameter ini menentukan lokal default yang digunakan oleh kolasi. Parameter ini digunakan dalam kombinasi dengan atribut yang tercantum dalam Babelfish collation attributes table untuk menyesuaikan kolasi untuk bahasa dan wilayah tertentu. Nilai default untuk parameter ini adalah `en-US`.

Lokal default berlaku untuk semua nama pemeriksaan Babelfish yang dimulai dengan "BBF" dan untuk semua kumpulan SQL Server yang dipetakan ke kumpulan Babelfish. Mengubah pengaturan untuk parameter ini pada klaster DB Babelfish yang ada tidak memengaruhi lokal kolasi yang ada. Untuk daftar kolasi, lihat Babelfish supported collations table.

babelfishpg_tsql.server_collation_name

Parameter ini menentukan pemeriksaan default untuk server (Aurora Postgre SQL DB cluster instance) dan database. Nilai default-nya adalah `sql_latin1_general_cp1_ci_as`. Harus `server_collation_name` berupa `CI_AS` pemeriksaan karena dalam T-SQL, pemeriksaan server menentukan bagaimana pengidentifikasi dibandingkan.

Saat Anda membuat klaster DB Babelfish, Anda memilih Nama kolasi dari daftar yang dapat dipilih. Ini termasuk kolasi yang tercantum dalam Babelfish supported collations table. Jangan memodifikasi `server_collation_name` setelah basis data Babelfish dibuat.

Pengaturan yang Anda pilih saat membuat klaster Babelfish untuk Aurora Postgre SQL DB disimpan dalam grup parameter cluster DB yang terkait dengan cluster untuk parameter ini dan mengatur perilaku koleksinya.

Kolasi deterministik dan nondeterministik di Babelfish

Babelfish mendukung kolasi deterministik dan nondeterministik:

- Kolasi deterministik mengevaluasi karakter yang memiliki urutan byte identik sebagai sama. Itu berarti itu `x` dan `X` tidak sama dalam kolasi deterministik. Kolasi deterministik dapat berupa peka huruf besar/kecil (CS) dan peka karakter beraksen (AS).
- Kolasi nondeterministik tidak membutuhkan kecocokan yang identik. Kolasi nondeterministik mengevaluasi `x` dan `X` sebagai nilai yang setara. Kumpulan nondeterministik adalah case-insensitive (CI) atau aksen-insensitif (AI), atau keduanya.

Pada tabel berikut, Anda dapat menemukan beberapa perbedaan perilaku antara Babelfish dan Postgre SQL saat menggunakan kumpulan nondeterministik.

| Babelfish | Postgre SQL |
| --- | --- |
| Mendukung LIKE klausa untuk kumpulan CI_AS. | Tidak mendukung LIKE klausa pada kumpulan nondeterministik. |
| Mendukung LIKE klausa hanya pada kumpulan AI berikut dari Babelfish versi 4.2.0:<br>• bbf_unicode_cp1250_ci_ai<br>• bbf_unicode_cp1250_cs_ai<br>• bbf_unicode_cp1257_ci_ai<br>• bbf_unicode_cp1257_cs_ai<br>• bbf_unicode_cp1_ci_ai<br>• bbf_unicode_cp1_cs_ai<br>• estonian_ci_ai<br>• finnish_swedish_ci_ai<br>• french_ci_ai<br>• latin1_general_ci_ai<br>• latin1_general_cs_ai<br>• modern_spanish_ci_ai<br>• polish_ci_ai<br>• sql_latin1_general_cp1_ci_ai<br>• sql_latin1_general_cp1_cs_ai<br>• tradisional_spanish_ci_ai | Tidak mendukung LIKE klausa pada kumpulan nondeterministik. |

Untuk daftar batasan dan perbedaan perilaku lain untuk Babelfish dibandingkan dengan SQL Server dan SQL Postgre, lihat. Batasan kolasi dan perbedaan perilaku

Babelfish dan SQL Server mengikuti konvensi penamaan untuk koleksi yang menggambarkan atribut pemeriksaan, seperti yang ditunjukkan pada tabel berikut.

| Atribut | Deskripsi |
| --- | --- |
| AI | Tidak peka karakter beraksen. |
| AS | Peka karakter beraksen. |
| BIN2 | BIN2meminta data untuk diurutkan dalam urutan titik kode. Urutan titik kode unicode adalah urutan karakter yang sama untuk pengkodean UTF -8, UTF -16, dan UCS -2. Urutan titik kode adalah kolasi deterministik cepat. |
| CI | Tidak peka huruf besar/kecil. |
| CS | Peka huruf besar/kecil. |
| PREF | Untuk mengurutkan huruf besar sebelum huruf kecil, gunakan pemeriksaan. PREF Jika perbandingan tidak peka huruf besar/kecil, versi huruf besar akan diurutkan sebelum versi huruf kecil, jika tidak ada perbedaan lain. ICUPustaka mendukung preferensi huruf besar dengan`colCaseFirst=upper` , tetapi tidak untuk koleksi CI_AS.<br><br>PREFhanya dapat diterapkan pada kumpulan `CS_AS` deterministik. |

Kolasi didukung pada tingkat database di Babelfish

Kumpulan berikut didukung pada tingkat database di Babelfish:

- bbf_unicode_bin2
- bbf_unicode_cp1_ci_ai
- bbf_unicode_cp1_ci_as
- bbf_unicode_cp1250_ci_ai
- bbf_unicode_cp1250_ci_as
- bbf_unicode_cp1257_ci_ai
- bbf_unicode_cp1257_ci_as
- estonian_ci_ai
- estonian_ci_as
- finnish_swedish_ci_ai

- finnish_swedish_ci_as
- french_ci_ai
- french_ci_as
- latin1_general_bin2
- latin1_general_ci_ai
- latin1_general_ci_as
- latin1_general_90_bin2
- latin1_general_100_bin2
- latin1_general_140_bin2
- modern_spanish_ci_ai
- modern_spanish_ci_as
- polish_ci_ai
- polish_ci_as
- sql_latin1_general_cp1_ci_ai
- sql_latin1_general_cp1_ci_as
- sql_latin1_general_cp1250_ci_as
- sql_latin1_general_cp1251_ci_as
- sql_latin1_general_cp1257_ci_as
- tradisional_spanish_ci_ai
- tradisional_spanish_ci_as

> **Note**
>
> Untuk menggunakan pemeriksaan yang berbeda di tingkat database, pastikan itu cocok dengan pemeriksaan tingkat server. Untuk informasi selengkapnya, silakan lihat Server dan Objek Collations di Babelfish

Server dan Objek Collations di Babelfish

Gunakan kolasi berikut sebagai kolasi server atau kolasi objek.

| ID Kolasi | Catatan |
| --- | --- |
| bbf_unicode_general_ci_as | Mendukung perbandingan case-insensitive dan operator. LIKE |
| bbf_unicode_cp1_ci_as | Pengumpulan nondeterministik juga dikenal sebagai. CP1252 |
| CP125bbf_unicode_ 0_ci_as | Kolasi nondeterministik digunakan untuk mewakili teks dalam bahasa Eropa Tengah dan Eropa Timur yang menggunakan aksara Latin. |
| CP1251bbf_unicode_ _ci_as | Kolasi nondeterministik untuk bahasa yang menggunakan skrip Sirilik. |
| bbf_unicode_cp1253_ci_as | Kolasi nondeterministik digunakan untuk mewakili bahasa Yunani modern. |
| bbf_unicode_cp1254_ci_as | Kolasi nondeterministik yang mendukung bahasa Turki. |
| bbf_unicode_cp1255_ci_as | Kolasi nondeterministik yang mendukung bahasa Ibrani. |
| bbf_unicode_cp1256_ci_as | Kolasi nondeterministik digunakan untuk menulis bahasa yang menggunakan aksara Arab. |
| bbf_unicode_cp1257_ci_as | Kolasi nondeterministik digunakan untuk mendukung bahasa Estonia, Latvia, dan Lituania. |
| bbf_unicode_cp1258_ci_as | Kolasi nondeterministik digunakan untuk menulis karakter Vietnam. |
| bbf_unicode_cp874_ci_as | Kolasi nondeterministik digunakan untuk menulis karakter Thailand. |

| ID Kolasi | Catatan |
| --- | --- |
| sql_latin1_general_cp1250_ci_as | Pengodean karakter byte tunggal nondeterministik digunakan untuk mewakili karakter Latin. |
| sql_latin1_general_cp1251_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1253_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1254_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1255_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1256_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1257_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| sql_latin1_general_cp1258_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| chinese_prc_ci_as | Pengumpulan nondeterministik yang mendukung bahasa Mandarin (). PRC |
| cyrillic_general_ci_as | Kolasi nondeterministik yang mendukung Sirilik. |

| ID Kolasi | Catatan |
| --- | --- |
| finnish_swedish_ci_as | Kolasi nondeterministik yang mendukung bahasa Finlandia. |
| french_ci_as | Kolasi nondeterministik yang mendukung bahasa Prancis. |
| japanese_ci_as | Kolasi nondeterministik yang mendukung bahasa Jepang. Didukung di Babelfish 2.1.0 dan rilis yang lebih baru. |
| korean_wansung_ci_as | Kolasi nondeterministik yang mendukung bahasa Korea (dengan jenis kamus). |
| latin1_general_ci_as | Kolasi nondeterministik yang mendukung karakter Latin. |
| modern_spanish_ci_as | Kolasi nondeterministik yang mendukung bahasa Spanyol Modern. |
| polish_ci_as | Kolasi nondeterministik yang mendukung bahasa Polandia. |
| thai_ci_as | Kolasi nondeterministik yang mendukung bahasa Thailand. |
| tradisional_spanish_ci_as | Kolasi nondeterministik yang mendukung bahasa Spanyol (jenis tradisional). |
| turkish_ci_as | Kolasi nondeterministik yang mendukung bahasa Turki. |
| ukrainian_ci_as | Kolasi nondeterministik yang mendukung Ukraina. |
| vietnamese_ci_as | Kolasi nondeterministik yang mendukung bahasa Vietnam. |

Anda dapat menggunakan kolasi berikut sebagai kolasi objek.

| Dialek | Opsi deterministik | Opsi nondeterministik |
| --- | --- | --- |
| Arab | Arabic_CS_AS | Arabic_CI_AS<br>Arabic_ci_ai |
| Skrip Arab | BBF_Unicode_ _CS_AS CP1256<br>BBF_Unicode_Pref_ _CS_AS CP1256 | BBF_Unicode_ _CI_AI CP1256<br>BBF_Unicode_ _CS_AI CP1256 |
| Biner | latin1_general_bin2<br>BBF_Unicode_ BIN2 | – |
| Bahasa Eropa Tengah dan Eropa Timur yang menggunakan aksara Latin | BBF_Unicode_ 0_CS_AS CP125<br>BBF_Unicode_Pref_ 0_CS_AS CP125 | BBF_Unicode_ 0_CI_AI CP125<br>BBF_Unicode_ 0_CS_AI CP125 |
| Mandarin | Cina_ _CS_AS PRC | PRCCina_ _CI_AS<br>PRCCina_ _CI_AI |
| Cyrillic_Umum | Cyrillic_General_CS_AS | Cyrillic_General_CI_AS<br>Cyrillic_General_CI_AI |
| Skrip Sirilik | BBF_Unicode_ _CS_AS CP1251<br>BBF_Unicode_Pref_ _CS_AS CP1251 | BBF_Unicode_ _CI_AI CP1251<br>BBF_Unicode_ _CS_AI CP1251 |
| Estonia | Estonian_CS_AS | Estonia_CI_AS |

| Dialek | Opsi deterministik | Opsi nondeterministik |
| --- | --- | --- |
| | | Estonia_CI_AI |
| Estonia, Latvia, dan Lituania | BBF_Unicode_ _CS_AS CP1257<br>BBF_Unicode_Pref_ _CS_AS CP1257 | BBF_Unicode_ _CI_AI CP1257<br>BBF_Unicode_ _CS_AI CP1257 |
| Finlandia _Swedia | Finnish_Swedish_CS_AS | Finnish_Swedish_CI_AS<br>Finlandia_Swedia_CI_AI |
| Prancis | French_CS_AS | French_CI_AS<br>Perancis_CI_AI |
| Yunani | Greek_CS_AS | Yunani_ci_as<br>Yunani_ci_ai |
| Ibrani | BBF_Unicode_ _CS_AS CP1255<br>BBF_Unicode_Pref_ _CS_AS CP1255<br>Hebrew_CS_AS | BBF_Unicode_ _CI_AI CP1255<br>BBF_Unicode_ _CS_AI CP1255<br>Hebrew_CI_AS<br>Ibrani_CI_ai |
| Jepang (Babelfish 2.1.0 dan lebih tinggi) | Japanese_CS_AS | Jepang_CI_AI<br>Japanese_CI_AS |
| Korea_Wamsung | Korean_Wamsung_CS_AS | Korea_Wamsung_CI_AS<br>Korea_Wamsung_CI_AI |

| Dialek | Opsi deterministik | Opsi nondeterministik |
| --- | --- | --- |
| Karakter Latin untuk halaman kode CP1252 | latin1_general_cs_as<br>BBF_Unicode_General_CS_AS<br>BBF_Unicode_General_Pref_CS_AS<br>BBF_Unicode_Pref_ _CS_AS CP1<br>BBF_Unicode_ _CS_AS CP1 | latin1_general_ci_as<br>latin1_general_ci_ai<br>latin1_general_cs_ai<br>BBF_Unicode_Umum_CI_AI<br>BBF_Unicode_Umum_CS_AI<br>BBF_Unicode_ _CI_AI CP1<br>BBF_Unicode_ _CS_AI CP1 |
| Yunani modern | BBF_Unicode_ _CS_AS CP1253<br>BBF_Unicode_Pref_ _CS_AS CP1253 | BBF_Unicode_ _CI_AI CP1253<br>BBF_Unicode_ _CS_AI CP1253 |
| Modern_Spanyol | Modern_Spanish_CS_AS | Modern_Spanish_CI_AS<br>Modern_Spanyol_CI_AI |
| Mongolia | Mongolian_CS_AS | Mongolia_CI_AS<br>Mongolia_CI_ai |
| Polandia | Polish_CS_AS | Polish_CI_AS<br>Polandia_CI_AI |
| Thai | BBF_Unicode_ _CS_AS CP874<br>BBF_Unicode_Pref_ _CS_AS CP874<br>Thai_CS_AS | BBF_Unicode_ _CI_AI CP874<br>BBF_Unicode_ _CS_AI CP874<br>Thai_CI_AS, Thai_CI_AI |

| Dialek | Opsi deterministik | Opsi nondeterministik |
| --- | --- | --- |
| Tradision al_Spanyol | Traditional_Spanish_CS_AS | Tradisional_Spanish_CI_AS<br>Tradisional_Spanyol_CI_AI |
| Turki | BBF_Unicode_ _CS_AS CP1254<br>BBF_Unicode_Pref_ _CS_AS CP1254<br>Turkish_CS_AS | BBF_Unicode_ _CI_AI CP1254<br>BBF_Unicode_ _CS_AI CP1254<br>Turkish_CI_AS, Turkish_CI_AI |
| Ukrania | Ukranian_CS_AS | Ukranian_ci_as<br>Ukranian_ci_ai |
| Vietnam | BBF_Unicode_ _CS_AS CP1258<br>BBF_Unicode_Pref_ _CS_AS CP1258<br>Vietnamese_CS_AS | BBF_Unicode_ _CI_AI CP1258<br>BBF_Unicode_ _CS_AI CP1258<br>Vietnam_CI_AS<br>Vietnam_ci_ai |

Perilaku Collation default di Babelfish

Sebelumnya, kolasi default dari tipe data yang dapat dikolasi adalah `pg_catalog.default`. Tipe data dan objek yang bergantung pada tipe data ini mengikuti kolasi peka huruf besar/kecil. Kondisi ini berpotensi berdampak pada SQL objek T dari kumpulan data dengan pemeriksaan case-insensitive. Dimulai dengan Babelfish 2.3.0, pemeriksaan default untuk tipe data yang dapat dijaminkan (kecuali TEXT danNTEXT) sama dengan pemeriksaan dalam parameter. `babelfishpg_tsql.server_collation_name` Saat Anda meningkatkan ke Babelfish 2.3.0, kolasi default dipilih secara otomatis pada saat pembuatan klaster DB, yang tidak menciptakan dampak yang terlihat.

### Mengelola kolasi

Pustaka ICU menyediakan pelacakan versi pemeriksaan untuk memastikan bahwa indeks yang bergantung pada kolasi dapat diindeks ulang ketika versi baru ICU tersedia. Untuk melihat apakah basis data Anda saat ini memiliki kolasi yang perlu disegarkan, Anda dapat menggunakan kueri berikut setelah menghubungkan menggunakan `psql` atau `pgAdmin`:

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

Kueri ini menghasilkan output seperti berikut ini:

```
 Collation | Object
-----------+--------
(0 rows)
```

Dalam contoh ini, tidak ada kolasi yang perlu diperbarui.

Untuk mendapatkan daftar kolasi yang telah ditentukan dalam basis data Babelfish Anda, Anda dapat menggunakan `psql` atau `pgAdmin` dengan kueri berikut:

```
SELECT * FROM pg_collation;
```

Kolasi yang telah ditentukan disimpan dalam tabel `sys.fn_helpcollations`. Anda dapat menggunakan perintah berikut untuk menampilkan informasi tentang kolasi (seperti lcid, style, dan bendera kolasi). Untuk mendapatkan daftar semua kolasi dengan menggunakan `sqlcmd`, hubungkan ke port T-SQL (1433, secara default) dan jalankan kueri berikut:

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                  description
------------------------------------- ----------------------------------------
arabic_cs_as                          Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                          Arabic, case-insensitive, accent-insensi
```

```
arabic_ci_as                          Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                      Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai              Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as              Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as         Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai              Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as              Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai              Default locale, code page 1254, case-ins
...
(124 rows affected)
```

Baris 1 dan 2 yang ditunjukkan pada contoh mempersempit output hanya untuk tujuan keterbacaan dokumentasi.

```
1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

Batasan kolasi dan perbedaan perilaku

Babelfish menggunakan pustaka ICU untuk dukungan kolasi. PostgreSQL dibuat dengan versi ICU tertentu dan dapat mencocokkan paling banyak satu versi kolasi. Variasi antarversi tidak dapat dihindari, begitu juga variasi kecil sepanjang waktu seiring berkembangnya bahasa. Dalam daftar berikut, Anda dapat menemukan batasan dan variasi perilaku yang diketahui dari kolasi Babelfish:

- Ketergantungan tipe indeks dan pemeriksaan – Indeks pada tipe yang ditentukan pengguna yang bergantung pada pustaka kolasi International Components for Unicode (ICU) (pustaka yang digunakan oleh Babelfish) valid saat versi pustaka berubah.
- Fungsi COLLATIONPROPERTY – Properti kolasi diimplementasikan hanya untuk kolasi BBF Babelfish yang didukung. Untuk informasi selengkapnya, lihat Babelfish supported collations table.
- Perbedaan aturan pengurutan Unicode – Kolasi SQL untuk SQL Server mengurutkan data yang berenkode Unicode (`nchar` dan `nvarchar`) secara berbeda dibandingkan data yang tidak berenkode Unicode (`char` dan `varchar`). Basis data Babelfish selalu berenkode UTF-8 dan selalu

menerapkan aturan pengurutan Unicode secara konsisten, terlepas dari tipe data, jadi urutan untuk `char` atau `varchar` sama dengan urutan untuk `nchar` atau `nvarchar`.

- Perilaku pengurutan dan kolasi setara sekunder – Kolasi ICU Unicode setara sekunder default (`CI_AS`) mengurutkan tanda baca dan karakter nonalfanumerik lainnya sebelum karakter numerik, dan karakter numerik sebelum karakter abjad. Namun, urutan tanda baca dan karakter khusus lainnya berbeda.
- Kolasi tersier, solusi untuk ORDER BY – Kolasi SQL, seperti `SQL_Latin1_General_Pref_CP1_CI_AS`, mendukung fungsi `TERTIARY_WEIGHTS` dan kemampuan untuk mengurutkan string yang membandingkan secara merata dalam kolasi `CI_AS` yang akan diurutkan berdasarkan huruf besar terlebih dahulu: `ABC`, `ABc`, `AbC`, `Abc`, `aBC`, `aBc`, `abC`, dan terakhir `abc`. Dengan demikian, fungsi analitik `DENSE_RANK OVER (ORDER BY column)` menilai string-string ini sebagai memiliki peringkat yang sama tetapi mengurutkan string berdasarkan huruf besar terlebih dahulu dalam sebuah partisi.

  Anda bisa mendapatkan hasil yang serupa menggunakan Babelfish dengan menambahkan klausa `COLLATE` ke klausa `ORDER BY` yang menentukan kolasi `CS_AS` tersier yang menentukan `@colCaseFirst=upper`. Namun, pengubah `colCaseFirst` hanya berlaku untuk string setara tersier (bukan setara sekunder seperti kolasi `CI_AS`). Jadi, Anda tidak dapat mengemulasi kolasi SQL tersier menggunakan satu kolasi ICU.

  Sebagai solusinya, kami merekomendasikan agar Anda mengubah aplikasi yang menggunakan kolasi `SQL_Latin1_General_Pref_CP1_CI_AS` untuk menggunakan kolasi `BBF_SQL_Latin1_General_CP1_CI_AS` terlebih dahulu. Kemudian, tambahkan `COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS` ke klausa `ORDER BY` apa pun untuk kolom ini.
- Ekspansi karakter – Ekspansi karakter memperlakukan satu karakter sama dengan urutan karakter di tingkat primer. Kolasi `CI_AS` default SQL Server mendukung ekspansi karakter. Kolasi ICU mendukung ekspansi karakter hanya untuk kolasi yang tidak peka terhadap aksen.

  Ketika ekspansi karakter diperlukan, gunakan kolasi `AI` untuk perbandingan. Namun, kolasi tersebut saat ini tidak didukung oleh operator LIKE.
- enkode char dan varchar – Ketika kolasi SQL digunakan untuk tipe data `char` atau `varchar`, urutan karakter ASCII 127 sebelumnya ditentukan oleh halaman kode spesifik untuk kolasi SQL tersebut. Untuk kolasi SQL, string yang dideklarasikan sebagai `char` atau `varchar` mungkin diurutkan secara berbeda dibandingkan string yang dideklarasikan sebagai `nchar` atau `nvarchar`.

PostgreSQL mengenkode semua string dengan enkode basis data, sehingga semua karakter dikonversi ke UTF-8 dan diurutkan menggunakan aturan Unicode.

Karena kolasi SQL mengurutkan tipe data nchar dan nvarchar menggunakan aturan Unicode, Babelfish mengenkode semua string di server menggunakan UTF-8. Babelfish mengurutkan string nchar dan nvarchar dengan cara yang sama seperti mengurutkan string char dan varchar, yaitu menggunakan aturan Unicode.

- Karakter tambahan – Fungsi SQL Server `NCHAR`, `UNICODE`, dan `LEN` mendukung karakter untuk titik kode di luar Unicode Basic Multilingual Plane (BMP). Sebaliknya, kolasi non-SC menggunakan karakter pasangan pengganti untuk menangani karakter tambahan. Untuk tipe data Unicode, SQL Server dapat mewakili hingga 65.535 karakter menggunakan UCS-2, atau rentang Unicode penuh (1.114.114 karakter) jika karakter tambahan digunakan.
- Kolasi yang peka terhadap Kana (KS) – Kolasi yang peka terhadap Kana (KS) adalah salah satu kolasi yang memperlakukan karakter Kana Jepang `Hiragana` dan `Katakana` secara berbeda. ICU mendukung standar kolasi Jepang `JIS X 4061`. Pengubah lokal `colhiraganaQ [on | off]` yang kini tidak digunakan lagi mungkin menyediakan fungsi yang sama dengan kolasi KS. Namun, kolasi KS untuk nama yang sama dengan SQL Server saat ini tidak didukung oleh Babelfish.
- Kolasi yang peka terhadap lebar (WS) – Ketika karakter byte tunggal (lebar setengah) dan karakter sama yang direpresentasikan sebagai karakter byte ganda (lebar penuh) diperlakukan secara berbeda, kolasi ini disebut sebagai peka terhadap lebar (WS). Kolasi WS untuk nama yang sama dengan SQL Server saat ini tidak didukung oleh Babelfish.
- Kolasi yang peka terhadap pemilih variasi (VSS) – Kolasi yang peka terhadap pemilih variasi (VSS) membedakan antara pemilih variasi ideografis dalam kolasi Jepang `Japanese_Bushu_Kakusu_140` dan `Japanese_XJIS_140`. Urutan variasi terdiri dari karakter dasar ditambah pemilih variasi tambahan. Jika Anda tidak memilih opsi `_VSS`, pemilih variasi tidak dipertimbangkan dalam perbandingan.

  Kolasi VSS saat ini tidak didukung oleh Babelfish.
- Kolasi BIN dan BIN2 – Kolasi BIN2 mengurutkan karakter menurut urutan titik kode. Urutan biner byte-per-byte UTF-8 mempertahankan urutan titik kode Unicode, sehingga kolasi ini juga cenderung menjadi kolasi dengan performa terbaik. Jika urutan titik kode Unicode berfungsi untuk aplikasi, pertimbangkan untuk menggunakan kolasi BIN2. Namun, penggunaan kolasi BIN2 dapat mengakibatkan data ditampilkan kepada klien dalam urutan yang tidak terduga secara budaya. Pemetaan baru untuk karakter huruf kecil ditambahkan ke Unicode seiring berjalannya waktu,

sehingga fungsi `LOWER` mungkin memiliki performa yang berbeda pada versi ICU yang berbeda. Kasus ini bersifat khusus untuk masalah versioning kolasi yang lebih umum daripada sebagai satu kasus spesifik untuk kolasi BIN2.

Babelfish menyediakan kolasi `BBF_Latin1_General_BIN2` dengan distribusi Babelfish untuk membuat kolasi dalam urutan titik kode Unicode. Dalam kolasi BIN, hanya karakter pertama yang diurutkan sebagai wchar. Karakter yang tersisa diurutkan byte-per-byte, secara efektif dalam urutan titik kode sesuai dengan enkodenya. Pendekatan ini tidak mengikuti aturan kolasi Unicode dan tidak didukung oleh Babelfish.

- Kolasi nondeterministik dan batasan CHARINDEX – Untuk rilis Babelfish yang lebih lama dari versi 2.1.0, Anda tidak dapat menggunakan CHARINDEX dengan kolasi nondeterministik. Secara default, Babelfish menggunakan kolasi yang tidak peka terhadap huruf besar/kecil (nondeterministik). Menggunakan CHARINDEX untuk versi Babelfish yang lebih lama akan menimbulkan kesalahan runtime berikut:

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> Batasan dan solusi ini hanya berlaku untuk Babelfish versi 1.x (versi Aurora PostgreSQL 13.x). Babelfish 2.1.0 dan rilis yang lebih tinggi tidak memiliki masalah ini.

Anda dapat mengatasi masalah ini dengan salah satu cara berikut:

- Ubah ekspresi secara eksplisit menjadi kolasi yang peka terhadap huruf besar/kecil dan seragamkan huruf besar/kecil pada kedua argumen dengan menerapkan LOWER atau UPPER. Misalnya, `SELECT charindex('x', a) FROM t1` akan menjadi berikut:

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- Buat fungsi SQL f_charindex, lalu ganti panggilan CHARINDEX dengan panggilan ke fungsi berikut:

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
```

```
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

## Mengelola penanganan kesalahan Babelfish dengan escape hatch

Babelfish meniru SQL perilaku untuk aliran kontrol dan status transaksi bila memungkinkan. Ketika Babelfish menemukan kesalahan, ia mengembalikan kode kesalahan yang mirip dengan kode kesalahan SQL Server. Jika Babelfish tidak dapat memetakan kesalahan ke kode SQL Server, ia mengembalikan kode kesalahan tetap (`33557097`) dan mengambil tindakan spesifik berdasarkan jenis kesalahan, sebagai berikut:

- Untuk kesalahan waktu kompilasi, Babelfish akan melakukan rollback pada transaksi.
- Untuk kesalahan runtime, Babelfish mengakhiri batch dan melakukan rollback pada transaksi.
- Untuk kesalahan protokol antara klien dan server, tidak akan ada rollback pada transaksi.

Jika kode kesalahan tidak dapat dipetakan menjadi kode yang setara dan kode untuk kesalahan serupa tersedia, kode kesalahan akan dipetakan menjadi kode alternatif. Misalnya, perilaku yang menyebabkan kode SQL Server `8143` dan keduanya `8144` dipetakan. `8143`

Kesalahan yang tidak dapat dipetakan tidak mematuhi konsep `TRY... CATCH`.

Anda dapat menggunakan `@@ERROR` untuk mengembalikan kode kesalahan SQL Server, atau `@@PGERROR` fungsi untuk mengembalikan kode SQL kesalahan Postgre. Anda juga dapat menggunakan fungsi `fn_mapped_system_error_list` untuk menampilkan daftar kode kesalahan yang dipetakan. Untuk informasi tentang kode SQL kesalahan Postgre, lihat situs web SQLPostgre.

Memodifikasi pengaturan escape hatch Babelfish

Untuk menangani pernyataan yang mungkin gagal, Babelfish mendefinisikan opsi tertentu yang disebut escape hatch. Escape hatch adalah opsi yang menentukan perilaku Babelfish ketika menemukan fitur atau sintaks yang tidak didukung.

Anda dapat menggunakan prosedur tersimpan `sp_babelfish_configure` untuk mengontrol pengaturan escape hatch. Gunakan skrip untuk mengatur escape hatch ke `ignore` atau `strict`. Jika disetel ke `strict`, Babelfish menampilkan kesalahan yang perlu Anda perbaiki sebelum melanjutkan.

Untuk menerapkan perubahan pada sesi saat ini dan pada tingkat klaster, sertakan kata kunci `server`.

Penggunaannya dapat dilihat sebagai berikut:

- Untuk mencantumkan semua escape hatch dan statusnya, serta informasi penggunaan, jalankan `sp_babelfish_configure`.
- Untuk membuat daftar penetasan escape bernama dan nilainya, untuk sesi saat ini atau di seluruh cluster, jalankan perintah `sp_babelfish_configure 'hatch_name'` di mana pengenal satu atau *`hatch_name`* lebih lubang escape. *`hatch_name`*dapat menggunakan SQL wildcard, seperti '%'.
- Untuk mengatur satu escape hatch atau lebih ke nilai yang ditentukan, jalankan `sp_babelfish_configure ['hatch_name' [, 'strict'|'ignore' [, 'server']]]`. Untuk membuat pengaturan menjadi permanen di tingkat klaster, sertakan kata kunci `server`, seperti yang ditunjukkan sebagai berikut:

  ```
  EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
  ```

  Untuk mengaturnya hanya untuk sesi ini, jangan gunakan `server`.
- Untuk mengatur ulang semua escape hatch ke nilai default-nya, jalankan `sp_babelfish_configure 'default'` (Babelfish versi 1.2.0 dan yang lebih tinggi).

String yang mengidentifikasi palka (atau palka) dapat mencakup SQL wildcard. Misalnya, berikut ini menetapkan semua penetasan escape sintaks `ignore` untuk cluster Aurora Postgre. SQL

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

Dalam tabel berikut, Anda dapat menemukan deskripsi dan nilai default untuk escape hatch Babelfish yang telah ditentukan sebelumnya.

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| `escape_hatch_checkpoint` | Mengizinkan penggunaan CHECKPOINT pernyataan dalam kode prosedural, tetapi CHECKPOINT pernyataan tersebut saat ini tidak diterapkan. | abaikan |
| `escape_hatch_constraint_name_for_default` | | abaikan |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| | Mengontrol perilaku Babelfish yang terkait dengan nama batasan default. | |
| `escape_hatch_database_misc_options` | Mengontrol perilaku Babelfish yang terkait dengan opsi berikut di CREATEDATABASE:CONTAINMENT, DB_,CHAINING, TRUSTWORTHY _ _. PERSISTENT LOG BUFFER | abaikan |
| `escape_hatch_for_replication` | Mengontrol perilaku Babelfish yang terkait dengan FOR REPLICATION klausa [NOT] saat membuat atau mengubah tabel. | ketat |
| `escape_hatch_fulltext` | Mengontrol perilaku Babelfish yang terkait dengan FULLTEXT fitur, seperti DEFAULT _ FULLTEXT _ LANGUAGE inCREATE/, ALTER DATABASE CREATE FULLTEXTINDEX, atau sp_fulltext_database. | abaikan |
| `escape_hatch_ignore_dup_key` | Mengontrol perilaku Babelfish terkait CREATE ALTER TABLE dengan/dan. CREATE INDEX Ketika IGNORE _ DUP _ KEY =ON, memunculkan kesalahan saat disetel ke `strict` (default) atau mengabaikan kesalahan saat disetel ke `ignore` (Babelfish versi 1.2.0 dan lebih tinggi). | ketat |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| `escape_hatch_index_clustering` | Mengontrol perilaku Babelfish terkait dengan CLUSTERED atau NONCLUSTERED kata kunci untuk indeks dan PRIMARY KEY atau kendala. UNIQUE Ketika CLUSTERED diabaikan, indeks atau kendala masih NONCLUSTERED dibuat seolah-olah telah ditentukan. | abaikan |
| `escape_hatch_index_columnstore` | Mengontrol perilaku Babelfish terkait dengan klausa. COLUMNSTORE Jika Anda menentukan `ignore`, Babelfish membuat indeks B-tree biasa. | ketat |
| `escape_hatch_join_hints` | Mengontrol perilaku kata kunci dalam JOIN operator:LOOP,HASH,MERGE,REMOTE,REDUCE,REDISTRIBUTE,REPLICATE. | abaikan |
| `escape_hatch_language_non_english` | Mengontrol perilaku Babelfish yang terkait dengan bahasa selain bahasa Inggris untuk pesan di layar. Babelfish saat ini hanya mendukung `us_english` untuk pesan di layar. SETLANGUAGEmungkin menggunakan variabel yang berisi nama bahasa, sehingga bahasa sebenarnya yang disetel hanya dapat dideteksi pada waktu berjalan. | ketat |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| escape_hatch_login_hashed_password | Ketika diabaikan, menekan kesalahan untuk kata kunci HASHED untuk CREATE LOGIN dan ALTER LOGIN. | ketat |
| escape_hatch_login_misc_options | Ketika diabaikan, menekan kesalahan untuk kata kunci lain selain HASHED, MUST_CHANGE , OLD_PASSWORD , dan UNLOCK untuk CREATE LOGIN dan ALTER LOGIN. | ketat |
| escape_hatch_login_old_password | Ketika diabaikan, menekan kesalahan untuk kata kunci OLD_PASSWORD untuk CREATE LOGIN dan ALTER LOGIN. | ketat |
| escape_hatch_login_password_must_change | Ketika diabaikan, menekan kesalahan untuk kata kunci MUST_CHANGE untuk CREATE LOGIN dan ALTER LOGIN. | ketat |
| escape_hatch_login_password_unlock | Ketika diabaikan, menekan kesalahan untuk kata kunci UNLOCK untuk CREATE LOGIN dan ALTER LOGIN. | ketat |
| escape_hatch_nocheck_add_constraint | Mengontrol perilaku Babelfish terkait dengan WITH CHECK atau NOCHECK klausa untuk kendala. | ketat |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| `escape_hatch_nocheck_existing_constraint` | Mengontrol perilaku Babelfish yang terkait dengan FOREIGN KEY atau CHECK kendala. | ketat |
| `escape_hatch_query_hints` | Mengontrol perilaku Babelfish yang terkait dengan petunjuk kueri. Ketika opsi ini diatur untuk mengabaikan, server mengabaikan petunjuk yang menggunakan klausa OPTION (...) untuk menentukan aspek pemrosesan kueri. Contohnya termasuk SELECTFROM... OPTION(MERGEJOINHASH, MAXRECURSION 10)). | abaikan |
| `escape_hatch_rowversion` | Mengontrol perilaku tipe ROWVERSION dan TIMESTAMP tipe data. Untuk informasi penggunaan, lihat Menggunakan fitur Babelfish dengan implementasi terbatas. | ketat |
| `escape_hatch_schemabinding_function` | Mengontrol perilaku Babelfish terkait dengan klausa. WITH SCHEMABINDING Secara default, WITH SCHEMABINDING klausa diabaikan ketika ditentukan dengan ALTER FUNCTION perintah CREATE or. | abaikan |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| escape_hatch_schemabinding_procedure | Mengontrol perilaku Babelfish terkait dengan klausa. WITH SCHEMABINDING Secara default, WITH SCHEMABINDING klausa diabaikan ketika ditentukan dengan ALTER PROCEDURE perintah CREATE or. | abaikan |
| escape_hatch_rowguidcol_column | Mengontrol perilaku Babelfish terkait dengan ROWGUIDCOL klausa saat membuat atau mengubah tabel. | ketat |
| escape_hatch_schemabinding_trigger | Mengontrol perilaku Babelfish terkait dengan klausa. WITH SCHEMABINDING Secara default, WITH SCHEMABINDING klausa diabaikan ketika ditentukan dengan ALTER TRIGGER perintah CREATE or. | abaikan |
| escape_hatch_schemabinding_view | Mengontrol perilaku Babelfish terkait dengan klausa. WITH SCHEMABINDING Secara default, WITH SCHEMABINDING klausa diabaikan ketika ditentukan dengan ALTER VIEW perintah CREATE or. | abaikan |
| escape_hatch_session_settings | Mengontrol perilaku Babelfish terhadap pernyataan tingkat sesi yang tidak didukung. SET | abaikan |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| `escape_hatch_showplan_all` | Mengontrol perilaku Babelfish terkait dengan SET SHOWPLAN _ ALL dan. SET STATISTICS PROFILE Ketika diatur untuk mengabaikan, mereka berperila ku seperti SET BABELFISH _ SHOWPLAN _ ALL dan SET BABELFISH _ STATISTIC SPROFILE; ketika diatur ke ketat, mereka diam-diam diabaikan. | ketat |
| `escape_hatch_storage_on_par tition` | Mengontrol perilaku Babelfish yang terkait dengan klausa ON `partition_scheme column` saat mendefinisikan partisi. Babelfish saat ini tidak menerapka n partisi. | ketat |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| `escape_hatch_storage_options` | Escape hatch pada opsi penyimpanan apa pun yang digunakan dalamCREATE,, ALTER DATABASETABLE,INDEX. Ini termasuk klausa (LOG) ON, TEXTIMAGE _ON, FILESTREAM _ON yang menentukan lokasi penyimpanan (partisi, grup file) untuk tabel, indeks, dan batasan, dan juga untuk database. Pengaturan escape hatch ini berlaku untuk semua klausa ini (termasuk ON [PRIMARY] dan ON "“DEFAULT). Pengecualiannya adalah ketika partisi ditentukan untuk tabel atau indeks dengan ON partition_scheme (kolom). | abaikan |
| `escape_hatch_table_hints` | Mengontrol perilaku petunjuk tabel yang ditentukan menggunakan klausa WITH (...). | abaikan |

| Escape hatch | Deskripsi | Default |
| --- | --- | --- |
| `escape_hatch_unique_constraint` | Ketika diatur ke ketat, perbedaan semantik yang tidak jelas antara SQL Server dan Postgre SQL dalam menangani NULL nilai pada kolom yang diindeks dapat menimbulkan kesalahan . Perbedaan semantik hanya muncul dalam kasus penggunaa n yang tidak realistis, sehingga Anda dapat mengatur escape hatch ini menjadi 'abaikan' untuk menghindari melihat kesalahan.<br><br>Usang dari versi berikut: 3.6.0 dan versi yang lebih tinggi, 4.2.0 dan versi yang lebih tinggi | ketat |

# Membuat klaster DB Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL didukung pada Aurora PostgreSQL versi 13.4 dan lebih baru.

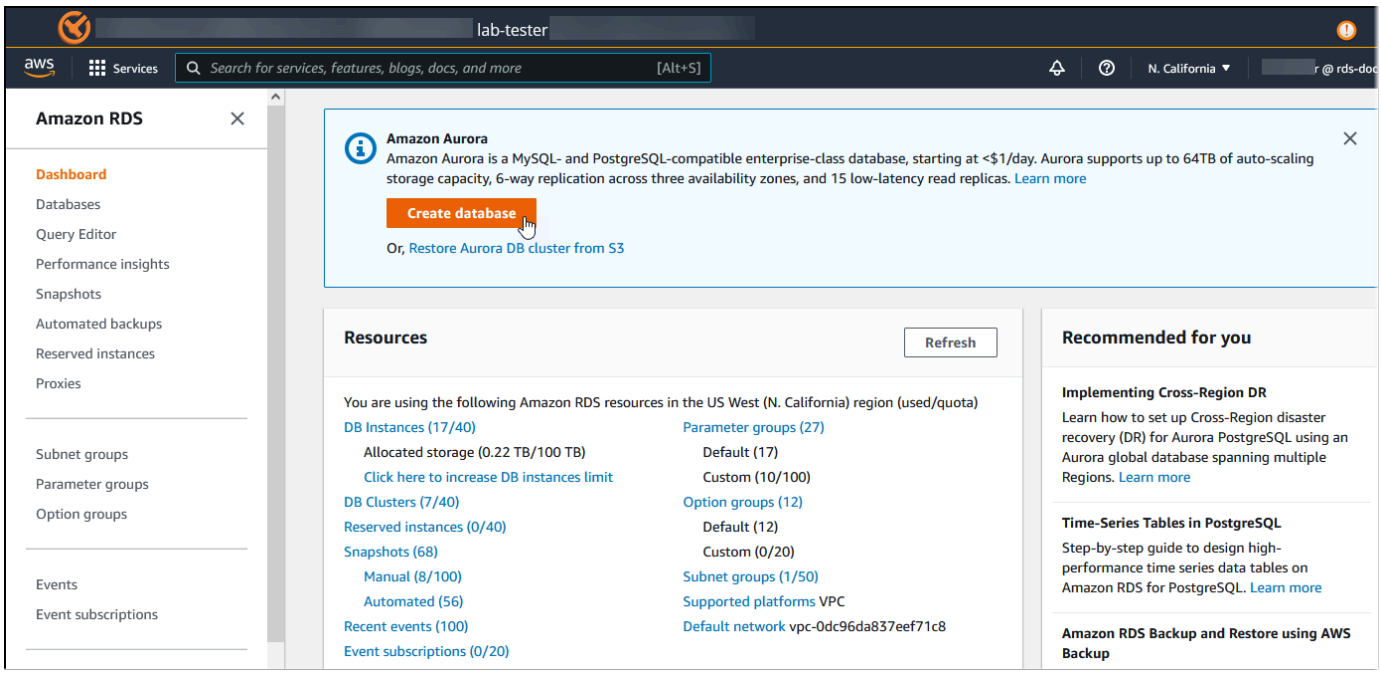
Anda dapat menggunakan AWS Management Console atau AWS CLI untuk membuat cluster Aurora PostgreSQL dengan Babelfish.

> **Note**
>
> Dalam klaster Aurora PostgreSQL, nama basis data `babelfish_db` dicadangkan untuk Babelfish. Membuat basis data "babelfish_db" Anda sendiri pada Babelfish for Aurora PostgreSQL mencegah Aurora berhasil menyediakan Babelfish.

Konsol

Untuk membuat klaster dengan Babelfish menjalankan dengan AWS Management Console

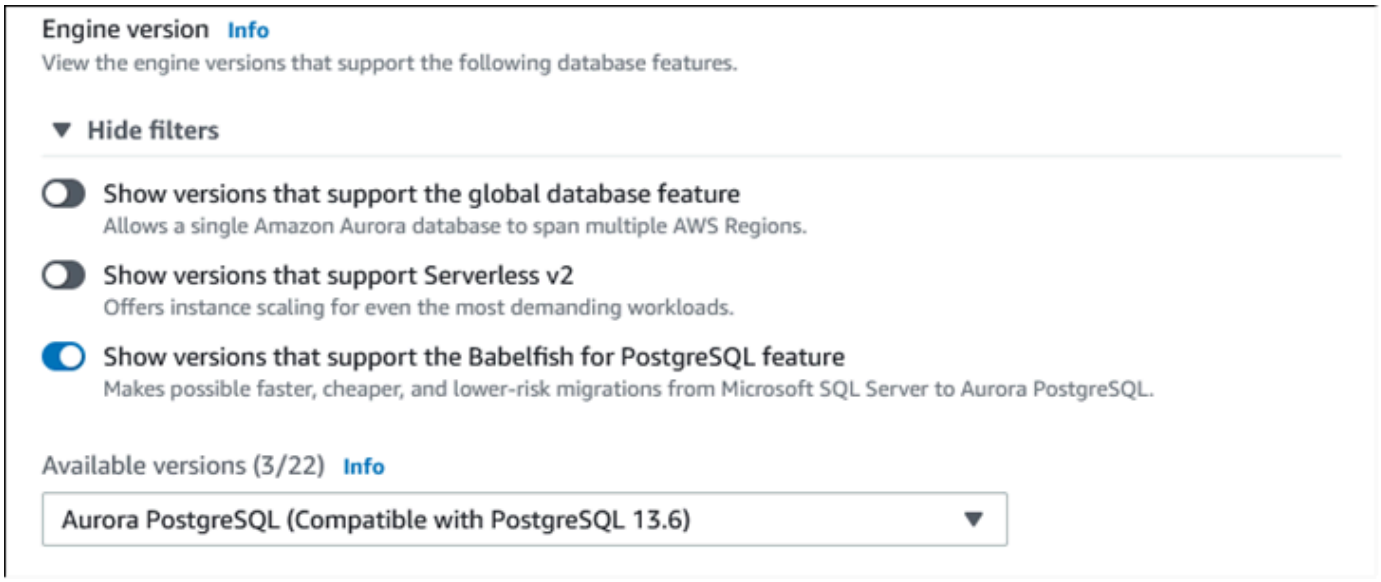
1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/, dan pilih Buat database.




2. Untuk Pilih metode pembuatan basis data, lakukan salah satu hal berikut:

   - Untuk menentukan opsi mesin terperinci, pilih Buat standar.
   - Untuk menggunakan opsi yang telah dikonfigurasi sebelumnya yang mendukung praktik terbaik untuk klaster Aurora, pilih Buat mudah.

3. Untuk Jenis mesin, pilih Aurora (Kompatibel dengan PostgreSQL).
4. Pilih Tampilkan filter, lalu pilih Tampilkan versi yang mendukung fitur Babelfish for PostgreSQL untuk mencantumkan jenis mesin yang mendukung Babelfish. Babelfish saat ini didukung pada Aurora PostgreSQL 13.4 dan versi yang lebih baru.
5. Untuk Versi yang tersedia, pilih versi Aurora PostgreSQL. Untuk mendapatkan fitur Babelfish terbaru, pilih versi utama Aurora PostgreSQL tertinggi.



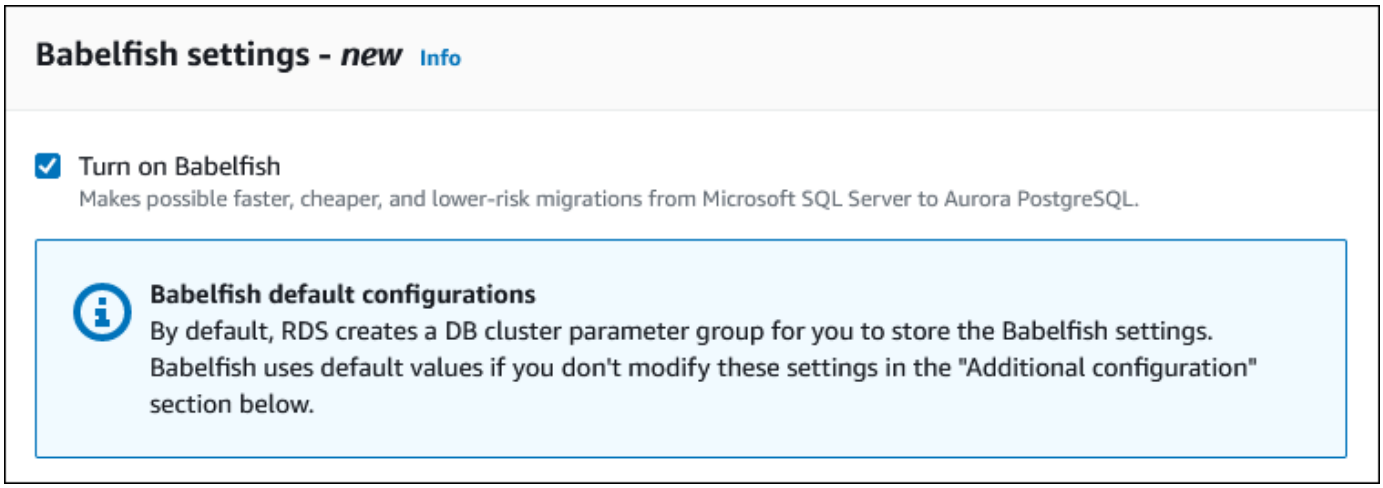

6. Untuk Templat, pilih templat yang sesuai dengan kasus penggunaan Anda.
7. Untuk Pengidentifikasi klaster DB, masukkan nama yang dapat Anda temukan dengan mudah nanti di daftar klaster DB.
8. Untuk Nama pengguna utama, masukkan nama pengguna administrator. Nilai default untuk Aurora PostgreSQL adalah `postgres`. Anda dapat menerima default atau memilih nama yang berbeda. Misalnya, untuk mengikuti konvensi penamaan yang digunakan pada basis data SQL Server Anda, Anda dapat memasukkan `sa` (administrator sistem) untuk Nama pengguna utama.

   Jika Anda tidak membuat nama pengguna `sa` saat ini, Anda dapat membuatnya nanti dengan klien pilihan Anda. Setelah membuat pengguna, gunakan perintah `ALTER SERVER ROLE` untuk menambahkannya ke grup (peran) `sysadmin` untuk klaster.

   > **Warning**
   >
   > Nama pengguna utama harus selalu menggunakan karakter huruf kecil yang gagal yang klaster DB tidak dapat terhubung ke Babelfish melalui port TDS.

9. Untuk Kata sandi utama, buat kata sandi yang kuat dan konfirmasi kata sandi.

10. Untuk opsi yang mengikuti, hingga bagian Pengaturan Babelfish, tentukan pengaturan klaster DB Anda. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.
11. Untuk membuat fungsionalitas Babelfish tersedia, pilih kotak Aktifkan Babelfish.




12. Untuk grup parameter klaster DB, lakukan salah satu hal berikut:

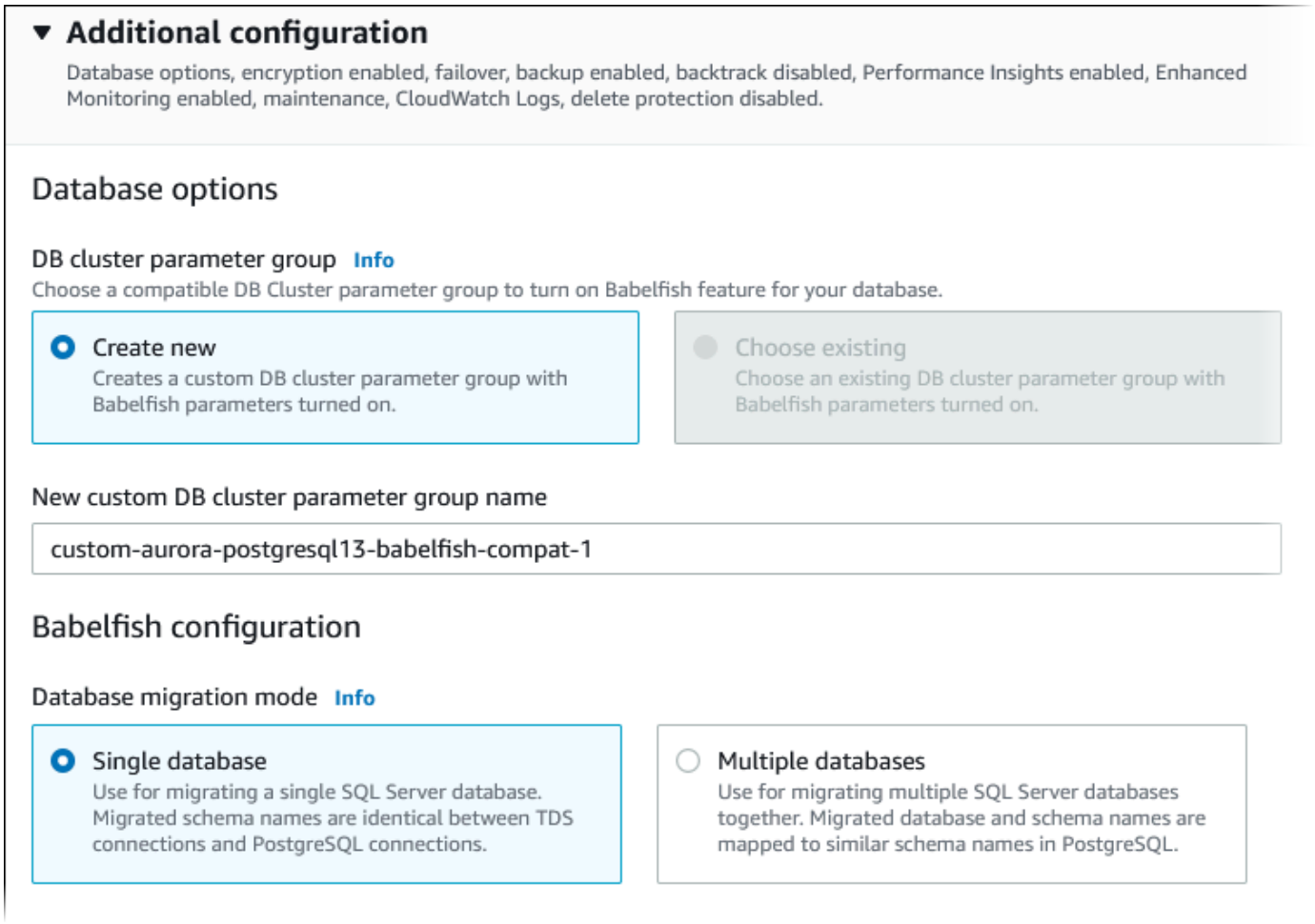
    - Pilih Buat baru untuk membuat grup parameter baru dengan Babelfish diaktifkan.
    - Klik Pilih yang ada untuk menggunakan grup parameter yang ada. Jika Anda menggunakan grup yang ada, pastikan untuk memodifikasi grup sebelum membuat klaster dan menambahkan nilai untuk parameter Babelfish. Untuk informasi tentang parameter Babelfish, lihat Pengaturan grup parameter klaster DB untuk Babelfish.

      Jika Anda menggunakan grup yang ada, berikan nama grup di kotak berikut.
13. Untuk Mode migrasi basis data, pilih salah satu langkah berikut:

    - Basis data tunggal untuk memigrasi basis data SQL Server tunggal.

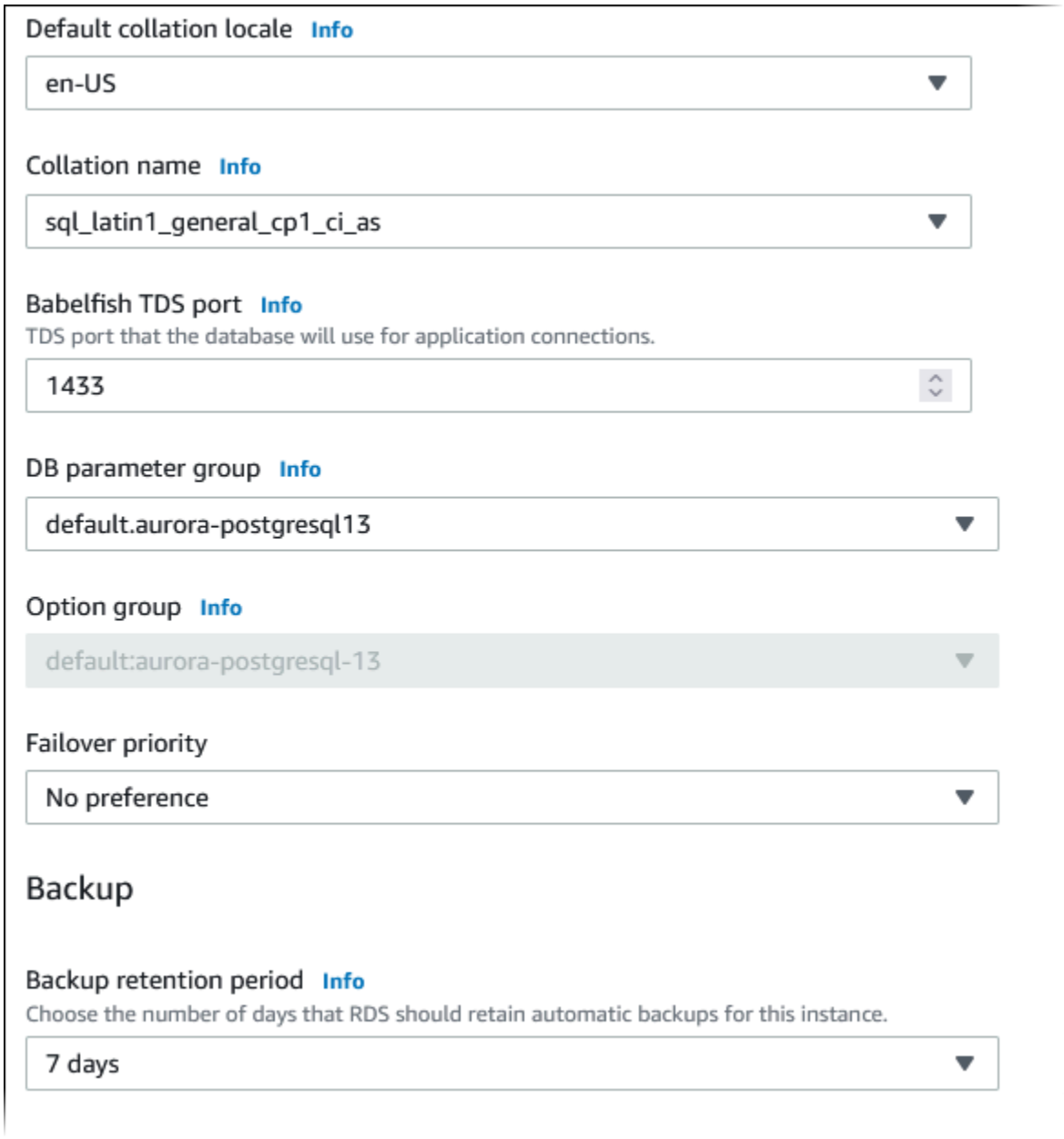
      Dalam beberapa kasus, Anda dapat memigrasikan beberapa basis data pengguna bersama-sama, dengan tujuan akhir Anda migrasi lengkap ke Aurora PostgreSQL asli tanpa Babelfish. Jika aplikasi akhir memerlukan skema konsolidasi (skema dbo tunggal), pastikan untuk terlebih dahulu mengkonsolidasikan basis data SQL Server Anda ke dalam basis data SQL server tunggal. Kemudian bermigrasi ke Babelfish menggunakan mode Basis data tunggal.
    - Beberapa basis data untuk memigrasi beberapa basis data SQL Server (berasal dari instalasi SQL Server tunggal). Beberapa mode basis data tidak mengkonsolidasikan beberapa basis data yang tidak berasal dari instalasi SQL Server tunggal. Untuk informasi tentang memigrasi beberapa basis data, lihat Menggunakan Babelfish dengan satu atau beberapa basis data.

> **Note**
>
> Dari versi Aurora PostgreSQL 16, Beberapa database dipilih secara default sebagai mode migrasi Database.




14. Untuk Lokal kolasi default, masukkan lokal server Anda. Default-nya adalah `en-US`. Untuk informasi mendetail tentang kolasi, lihat Memahami Kolasi di Babelfish untuk Aurora Postgre SQL.
15. Untuk kolom Nama kolasi, masukkan kolasi default Anda. Default-nya adalah `sql_latin1_general_cp1_ci_as`. Untuk detail informasi, lihat Memahami Kolasi di Babelfish untuk Aurora Postgre SQL.
16. Untuk Port TDS Babelfish, masukkan port default `1433`. Saat ini, Babelfish hanya mendukung port `1433` untuk klaster DB Anda.

17. Untuk grup parameter DB, pilih grup parameter atau minta Aurora membuat grup baru untuk Anda dengan pengaturan default.
18. Untuk Prioritas failover, pilih prioritas failover untuk instans. Jika Anda tidak memilih suatu nilai, maka nilai default-nya adalah `tier-1`. Prioritas ini menentukan urutan di mana replika dipromosikan saat pulih dari kegagalan instans utama. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.
19. Untuk Periode retensi cadangan, pilih durasi waktu (1–35 hari) Aurora mempertahankan salinan cadangan basis data. Anda dapat menggunakan salinan cadangan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua. Periode penyimpanan default adalah tujuh hari.



20. Pilih Salin tag ke snapshot untuk menyalin tag instans DB apa pun ke snapshot DB saat Anda membuat snapshot.

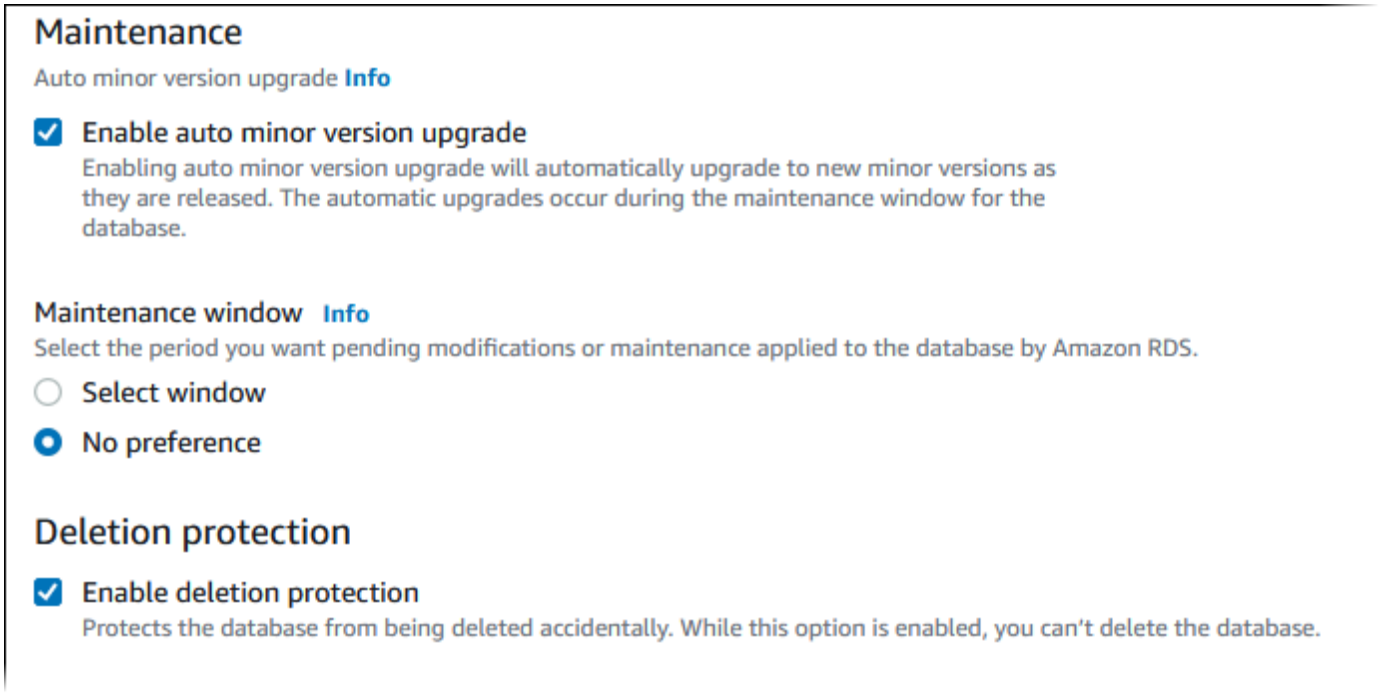
    > **Note**
    >
    > Saat memulihkan cluster DB dari snapshot, itu tidak memulihkan sebagai Babelfish untuk cluster DB PostgreSQL Aurora. Anda perlu mengaktifkan parameter yang mengontrol preferensi Babelfish di grup parameter cluster DB untuk mengaktifkan Babelfish lagi. Untuk informasi lebih lanjut tentang parameter Babelfish, lihat. Pengaturan grup parameter klaster DB untuk Babelfish

21. Pilih Aktifkan enkripsi untuk mengaktifkan enkripsi saat istirahat (enkripsi penyimpanan Aurora) untuk klaster DB ini.
22. Pilih Aktifkan Wawasan Performa untuk mengaktifkan Wawasan Performa Amazon RDS.
23. Pilih Aktifkan Pemantauan lanjutan untuk mulai mengumpulkan metrik secara waktu nyata untuk sistem operasi tempat klaster DB Anda dijalankan.
24. Pilih log PostgreSQL untuk mempublikasikan file log ke Amazon Logs. CloudWatch
25. Pilih Aktifkan peningkatan versi minor otomatis untuk memperbarui klaster DB Aurora Anda secara otomatis saat peningkatan versi minor tersedia.
26. Untuk Jendela pemeliharaan, lakukan hal berikut:

    - Untuk memilih waktu bagi Amazon RDS untuk melakukan modifikasi atau melakukan pemeliharaan, klik Pilih jendela.
    - Untuk melakukan pemeliharaan Amazon RDS pada waktu yang tidak terjadwal, pilih Tidak ada preferensi.

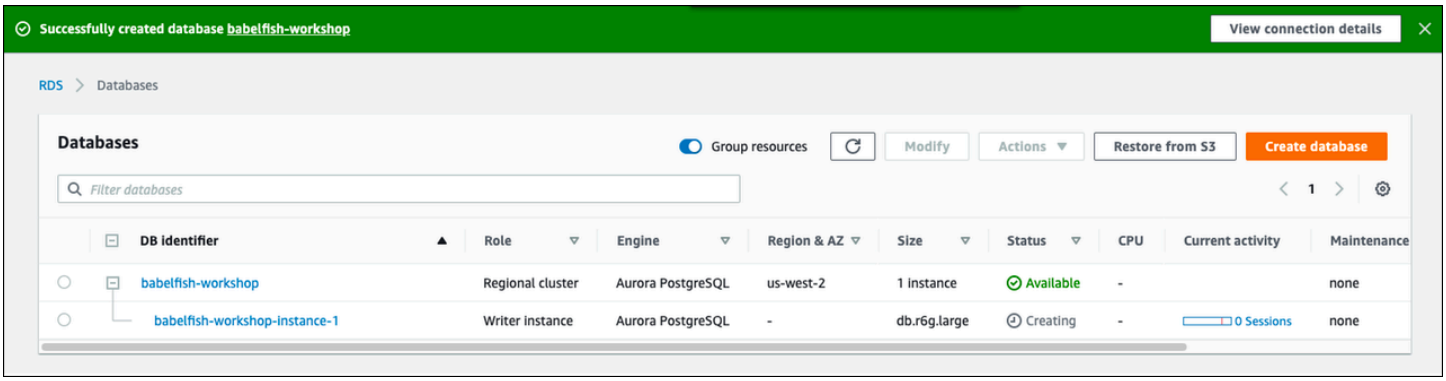
27. Pilih kotak Aktifkan perlindungan penghapusan untuk melindungi basis data Anda agar tidak terhapus secara tidak sengaja.

    Jika Anda mengaktifkan fitur ini, Anda tidak dapat langsung menghapus basis data. Namun, Anda perlu memodifikasi klaster basis data dan mematikan fitur ini sebelum menghapus basis data.




28. Pilih Buat basis data.

Anda dapat menemukan basis data baru yang disiapkan untuk Babelfish di daftar Basis data. Kolom Status ditampilkan Tersedia saat deployment selesai.




AWS CLI

Saat Anda membuat Babelfish untuk Aurora PostgreSQL; menggunakan AWS CLI, Anda harus meneruskan perintah nama grup parameter cluster DB yang akan digunakan untuk cluster. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Sebelum Anda dapat menggunakan AWS CLI untuk membuat cluster Aurora PostgreSQL dengan Babelfish, lakukan hal berikut:

- Pilih URL titik akhir Anda dari daftar layanan di titik akhir dan kuota Amazon Aurora.

- Buat grup parameter untuk klaster. Untuk informasi selengkapnya tentang grup parameter, lihat .
- Ubah grup parameter, tambahkan parameter yang mengaktifkan Babelfish.

Untuk membuat cluster DB PostgreSQL Aurora dengan Babelfish menggunakan AWS CLI

Contoh yang mengikuti menggunakan Nama pengguna utama default, `postgres`. Ganti sesuai kebutuhan dengan nama pengguna yang Anda buat untuk klaster DB Anda, seperti `sa` atau nama pengguna apa pun yang Anda pilih jika Anda tidak menerima default.

1. Buat grup parameter.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster-parameter-group \
   --endpoint-url endpoint-url \
   --db-cluster-parameter-group-name parameter-group \
   --db-parameter-group-family aurora-postgresql14 \
   --description "description"
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster-parameter-group ^
   --endpoint-url endpoint-URL ^
   --db-cluster-parameter-group-name parameter-group ^
   --db-parameter-group-family aurora-postgresql14 ^
   --description "description"
   ```

2. Ubah grup parameter Anda untuk mengaktifkan Babelfish.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds modify-db-cluster-parameter-group \
   --endpoint-url endpoint-url \
   --db-cluster-parameter-group-name parameter-group \
   --parameters
    "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
   ```

   Untuk Windows:

   ```
   aws rds modify-db-cluster-parameter-group ^
   ```

```
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. Identifikasi grup subnet DB Anda dan ID grup keamanan virtual private cloud (VPC) untuk cluster DB baru Anda, lalu panggil create-db-clusterperintahnya.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster \
   --db-cluster-identifier  cluster-name\
   --master-username postgres \
   --manage-master-user-password \
   --engine aurora-postgresql \
   --engine-version 14.3            \
   --vpc-security-group-ids security-group \
   --db-subnet-group-name subnet-group-name \
   --db-cluster-parameter-group-name parameter-group
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster ^
   --db-cluster-identifier cluster-name ^
   --master-username postgres ^
   --manage-master-user-password ^
   --engine aurora-postgresql ^
   --engine-version 14.3 ^
   --vpc-security-group-ids security-group ^
   --db-subnet-group-name subnet-group ^
   --db-cluster-parameter-group-name parameter-group
   ```

   Contoh ini menentukan `--manage-master-user-password` opsi untuk menghasilkan kata sandi pengguna utama dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Atau, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

4. Buat instans utama untuk klaster DB Anda. Gunakan nama cluster yang Anda buat di langkah 3 untuk `--db-cluster-identifier` argumen saat Anda memanggil create-db-instanceperintah, seperti yang ditunjukkan berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
--db-instance-identifier instance-name \
--db-instance-class db.r6g \
--db-subnet-group-name subnet-group \
--db-cluster-identifier cluster-name \
--engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
--db-instance-identifier instance-name ^
--db-instance-class db.r6g ^
--db-subnet-group-name subnet-group ^
--db-cluster-identifier cluster-name ^
--engine aurora-postgresql
```

# Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL

Anda dapat menggunakan Babelfish for Aurora PostgreSQL untuk memigrasi dari basis data SQL Server ke klaster DB Amazon Aurora PostgreSQL. Sebelum bermigrasi, tinjau Menggunakan Babelfish dengan satu atau beberapa basis data.

Topik

- Ikhtisar proses migrasi
- Mengevaluasi dan menangani perbedaan antara SQL Server dan Babelfish
- Alat impor/ekspor untuk bermigrasi dari SQL Server ke Babelfish

## Ikhtisar proses migrasi

Ringkasan berikut mencantumkan langkah-langkah yang diperlukan untuk menyukseskan proses migrasi aplikasi SQL Server Anda dan membuatnya bekerja dengan Babelfish. Untuk informasi tentang alat yang dapat Anda gunakan untuk proses ekspor dan impor dan untuk detail selengkapnya, lihat Alat impor/ekspor untuk bermigrasi dari SQL Server ke Babelfish. Untuk memuat data, sebaiknya gunakan AWS DMS cluster DB PostgreSQL Aurora sebagai titik akhir target.

1. Buat klaster DB Aurora PostgreSQL baru dengan kondisi Babelfish diaktifkan. Untuk mempelajari caranya, lihat Membuat klaster DB Babelfish for Aurora PostgreSQL.

   Untuk mengimpor berbagai artefak SQL yang diekspor dari basis data SQL Server Anda, sambungkan ke klaster Babelfish menggunakan alat SQL Server seperti sqlcmd. Untuk informasi selengkapnya, lihat Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda.
2. Pada basis data SQL Server yang ingin Anda migrasikan, ekspor bahasa definisi data (DDL). DDL adalah kode SQL yang menggambarkan objek basis data yang berisi data pengguna (seperti tabel, indeks, dan tampilan) dan kode basis data yang ditulis pengguna (seperti prosedur tersimpan, fungsi yang ditentukan pengguna, dan pemicu).

   Untuk informasi selengkapnya, lihat Menggunakan SQL Server Management Studio (SSMS) untuk memigrasi ke Babelfish.
3. Jalankan alat penilaian untuk mengevaluasi ruang lingkup perubahan apa pun yang mungkin perlu Anda buat sehingga Babelfish dapat secara efektif mendukung aplikasi yang berjalan di SQL Server. Untuk informasi selengkapnya, lihat Mengevaluasi dan menangani perbedaan antara SQL Server dan Babelfish.

4. Tinjau batasan titik akhir AWS DMS target dan perbarui skrip DDL seperlunya. Untuk informasi selengkapnya, lihat Batasan menggunakan titik akhir target PostgreSQL dengan tabel Babelfish di Menggunakan Babelfish untuk Aurora PostgreSQL sebagai target.
5. Pada klaster DB Babelfish baru Anda, jalankan DDL dalam basis data T-SQL yang Anda tentukan untuk membuat hanya skema, tipe data yang ditentukan pengguna, dan tabel dengan batasan kunci primernya.
6. Gunakan AWS DMS untuk memigrasikan data Anda dari SQL Server ke tabel Babelfish. Untuk replikasi berkelanjutan menggunakan SQL Server Change Data Capture atau SQL Replication, gunakan Aurora PostgreSQL alih-alih Babelfish sebagai titik akhir. Untuk melakukannya, lihat Menggunakan Babelfish for Aurora PostgreSQL sebagai target untuk AWS Database Migration Service.
7. Saat pemuatan data selesai, buat semua objek T-SQL yang tersisa yang mendukung aplikasi di klaster Babelfish Anda.
8. Konfigurasikan ulang aplikasi klien Anda untuk terhubung ke titik akhir Babelfish alih-alih basis data SQL Server Anda. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Babelfish.
9. Ubah aplikasi Anda sesuai kebutuhan dan uji ulang. Untuk informasi selengkapnya, lihat Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server.

Anda masih perlu menilai kueri SQL sisi klien Anda. Skema yang dihasilkan dari instans SQL Server Anda hanya mengonversi kode SQL sisi server. Sebaiknya Anda melakukan tindakan berikut:

- Tangkap kueri sisi klien dengan menggunakan SQL Server Profiler dengan templat yang telah ditentukan TSQL_Replay. Templat ini menangkap informasi pernyataan T-SQL yang kemudian dapat Anda putar ulang untuk penyesuaian dan pengujian berulang. Anda dapat memulai profiler dalam SQL Server Management Studio dari menu Alat. Pilih SQL Server Profiler untuk membuka profiler dan memilih templat TSQL_Replay.

  Untuk penggunaan pada migrasi Babelfish Anda, mulai jejak dan kemudian jalankan aplikasi Anda menggunakan pengujian fungsional Anda. Profiler menangkap pernyataan T-SQL. Saat Anda selesai menguji, hentikan jejaknya. Simpan hasilnya ke file XML dengan kueri sisi klien Anda (File > Save as > Trace XML File for Replay).

  Untuk informasi selengkapnya, lihat SQL Server Profiler di dokumentasi Microsoft. Untuk informasi selengkapnya tentang templat TSQL_Replay, lihat Templat SQL Server Profiler.

- Untuk aplikasi dengan kueri SQL sisi klien yang kompleks, kami menyarankan Anda menggunakan Babelfish Compass untuk menganalisis kueri ini untuk kompatibilitas Babelfish. Jika analisis menunjukkan bahwa pernyataan SQL sisi klien berisi fitur SQL yang tidak didukung, tinjau aspek SQL dalam aplikasi klien dan modifikasi sesuai kebutuhan.
- Anda juga dapat menangkap kueri SQL sebagai peristiwa yang diperpanjang (format .xel). Untuk melakukannya, gunakan SSMS XEvent Profiler. Setelah membuat file .xel, ekstrak pernyataan SQL ke dalam file .xml yang kemudian dapat diproses oleh Compass. Untuk informasi selengkapnya, lihat Menggunakan XEvent Profler SSMS di dokumentasi Microsoft.

Ketika Anda puas dengan semua pengujian, analisis, dan modifikasi apa pun yang diperlukan untuk aplikasi yang dimigrasi, Anda dapat mulai menggunakan basis data Babelfish untuk produksi. Untuk melakukannya, hentikan basis data asli dan alihkan aplikasi klien langsung untuk menggunakan port TDS Babelfish.

> **Note**
>
> AWS DMS sekarang mendukung replikasi data dari Babelfish. Untuk informasi lebih lanjut, lihat AWS DMS sekarang mendukung Babelfish untuk Aurora PostgreSQL sebagai sumber.

## Mengevaluasi dan menangani perbedaan antara SQL Server dan Babelfish

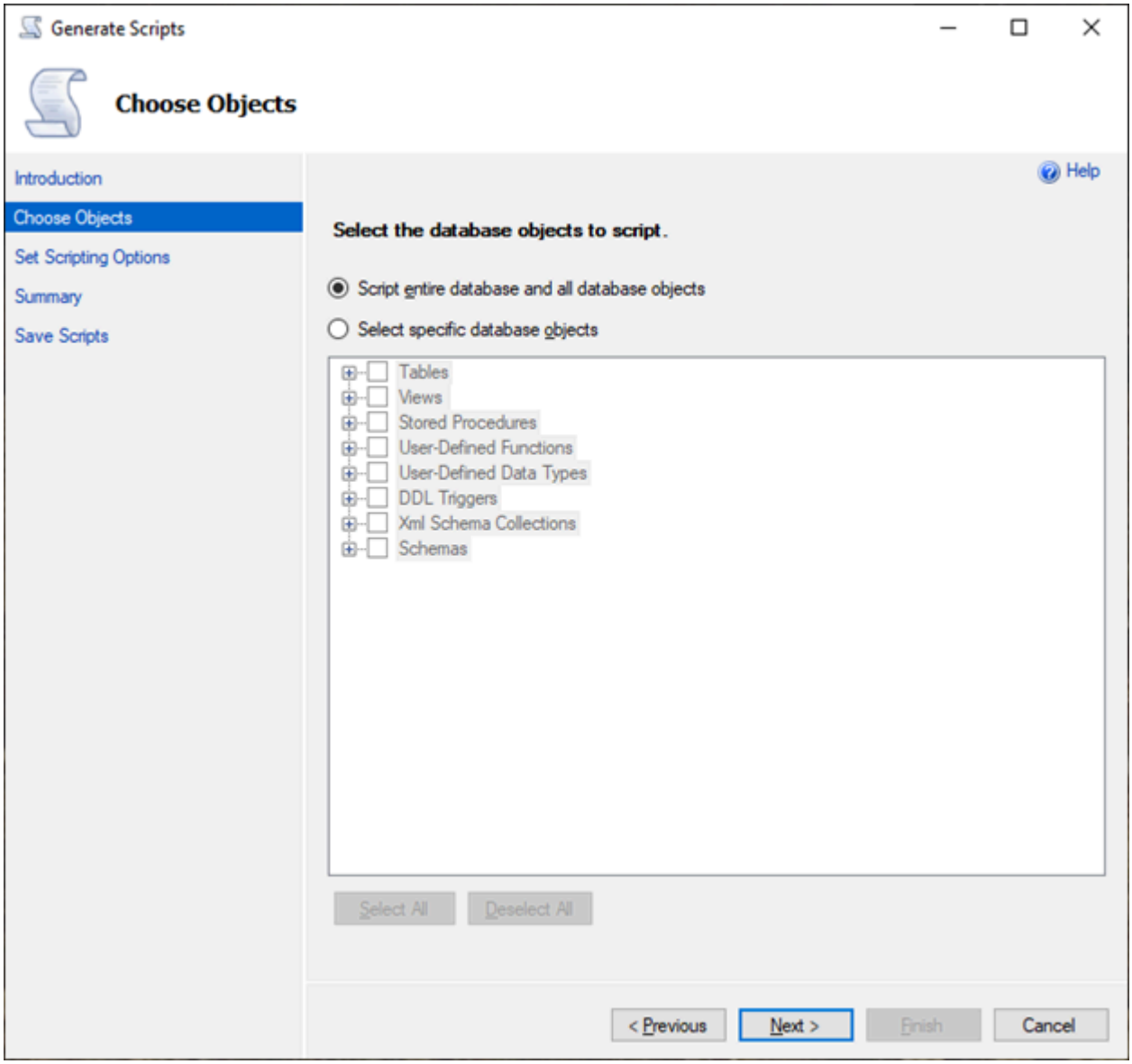
Untuk hasil terbaik, kami sarankan Anda mengevaluasi kode kueri yang dihasilkan DDL/DML dan klien sebelum benar-benar memigrasikan aplikasi database SQL Server Anda ke Babelfish. Bergantung pada versi Babelfish dan fitur spesifik SQL Server yang diterapkan aplikasi Anda, Anda mungkin perlu memfaktor ulang aplikasi Anda atau menggunakan alternatif untuk fungsionalitas yang belum sepenuhnya didukung di Babelfish.

- Untuk menilai kode aplikasi SQL Server Anda, gunakan Babelfish Compass pada DDL yang dihasilkan untuk menentukan berapa banyak kode T-SQL yang didukung oleh Babelfish. Identifikasi kode T-SQL yang mungkin memerlukan modifikasi sebelum berjalan di Babelfish. Untuk informasi lebih lanjut tentang alat ini, lihat Alat Kompas Babelfish di. GitHub

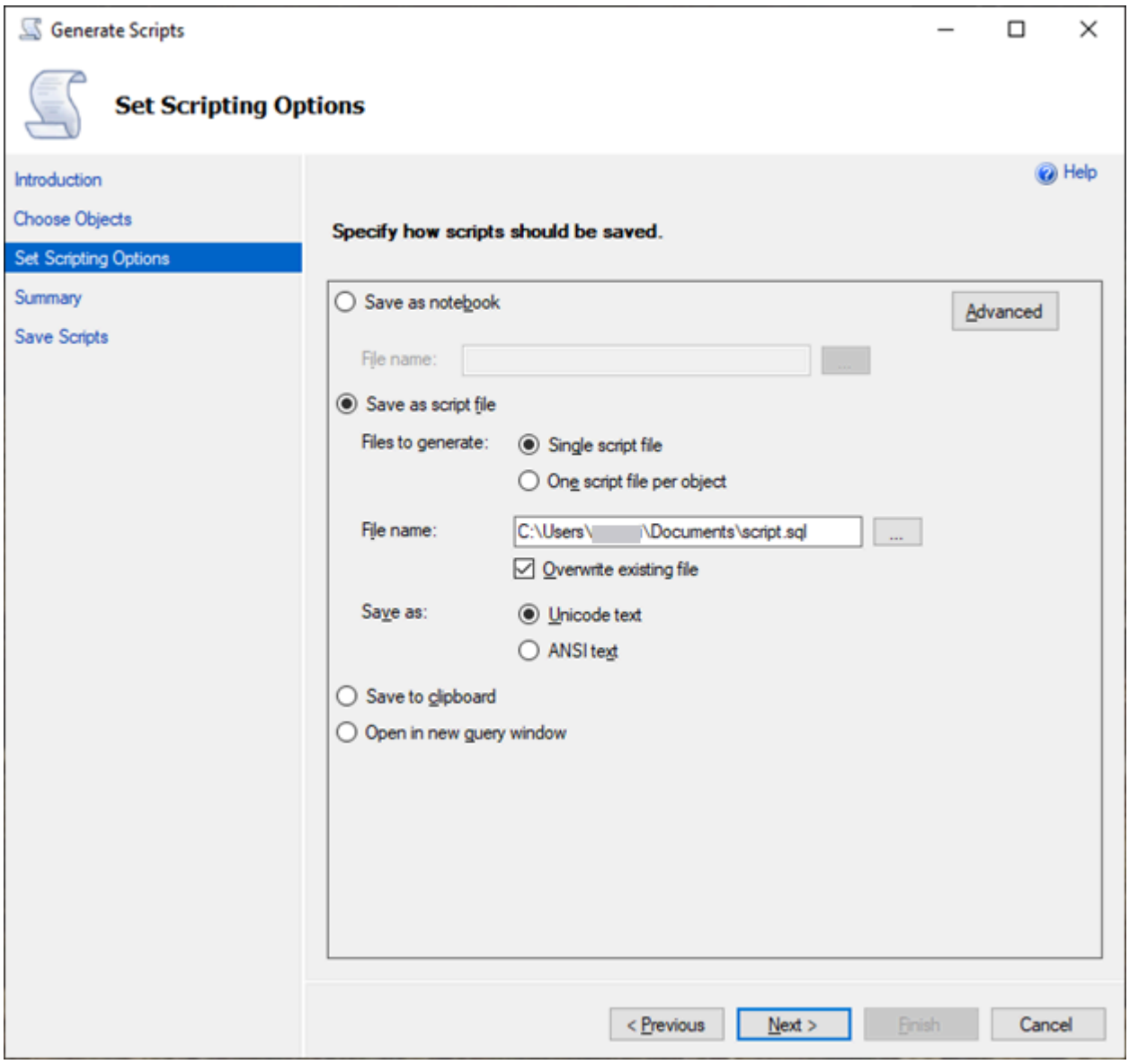
  > **Note**
  >
  > Babelfish Compass adalah alat sumber terbuka. Laporkan masalah apa pun dengan Babelfish Compass melalui GitHub alih-alih melalui Support. AWS

Anda dapat menggunakan Wizard Generate Script dengan SQL Server Management Studio (SSMS) untuk menghasilkan file SQL yang dinilai oleh Babelfish Compass atau CLI AWS Schema Conversion Tool . Kami merekomendasikan langkah-langkah berikut untuk mengefisienkan penilaian.

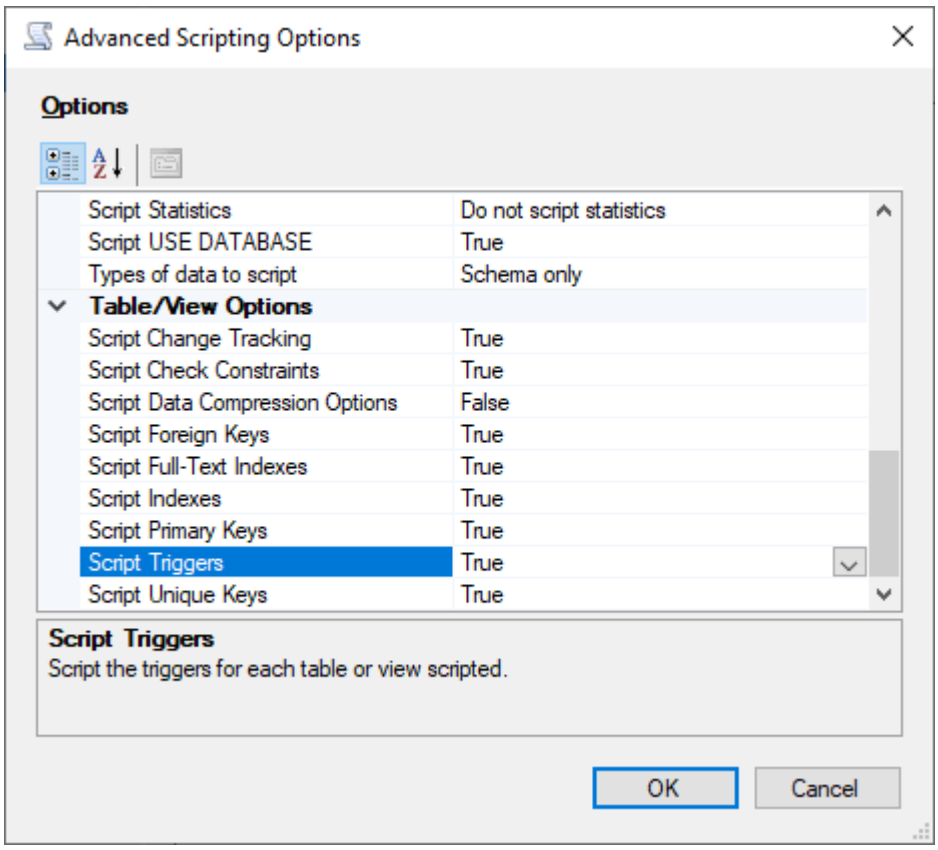
1. Pada halaman Choose Objects, pilih Script entire database and all database objects.




2. Untuk Set Scripting Options, pilih Save as script file sebagai Single script file.




3. Pilih Advanced untuk mengubah opsi pembuatan skrip default untuk mengidentifikasi fitur yang biasanya disetel ke false untuk penilaian lengkap:
   - Script Change Tracking ke True
   - Script Full-Text Indexes ke True
   - Script Triggers ke True
   - Script Logins ke True
   - Script Owner ke True
   - Script Object-Level Permissions ke True
   - Script Collations ke True




4. Lakukan langkah-langkah selanjutnya di wizard untuk membuat file.

## Alat impor/ekspor untuk bermigrasi dari SQL Server ke Babelfish

Sebaiknya gunakan AWS DMS sebagai alat utama untuk bermigrasi dari SQL Server ke Babelfish. Namun, Babelfish mendukung beberapa cara lain untuk memigrasikan data menggunakan alat SQL Server yang mencakup yang berikut ini.

- SQL Server Integration Services (SSIS) untuk semua versi Babelfish. Untuk informasi selengkapnya, lihat Migrasi dari SQL Server ke Aurora PostgreSQL menggunakan SSIS dan Babelfish.
- Gunakan Import/Export Wizard SSMS untuk Babelfish versi 2.1.0 dan yang lebih baru. Alat ini tersedia melalui SSMS, tetapi juga tersedia sebagai alat mandiri. Untuk informasi selengkapnya, lihat Welcome to SQL Server Import and Export Wizard di dokumentasi Microsoft.
- Utilitas program salinan data massal (bcp) Microsoft memungkinkan Anda menyalin data dari instans Microsoft SQL Server ke file data dalam format yang Anda tentukan. Untuk informasi selengkapnya, lihat bcp Utility di dokumentasi Microsoft. Dukungan untuk migrasi data dengan menggunakan klien BCP dan utilitas bcp kini mendukung bendera `-E` (untuk kolom identitas) dan

bendera -b (untuk sisipan batch). Opsi bcp tertentu tidak didukung, termasuk `-C`, `-T`, `-G`, `-K`, `-R`, `-V`, dan `-h`.

Menggunakan SQL Server Management Studio (SSMS) untuk memigrasi ke Babelfish

Sebaiknya buat file terpisah untuk masing-masing jenis objek tertentu. Anda dapat menggunakan wizard Generate Scripts di SSMS untuk setiap set pernyataan DDL terlebih dahulu, dan kemudian memodifikasi objek sebagai grup untuk memperbaiki masalah yang ditemukan selama penilaian.

Lakukan langkah-langkah ini untuk memigrasikan data menggunakan AWS DMS atau metode migrasi data lainnya. Jalankan jenis skrip create ini terlebih dahulu untuk pendekatan yang lebih baik dan lebih cepat untuk memuat data pada tabel Babelfish di Aurora PostgreSQL.

1. Jalankan pernyataan `CREATE SCHEMA`.
2. Jalankan pernyataan `CREATE TYPE` untuk membuat tipe data yang ditentukan pengguna.
3. Jalankan pernyataan `CREATE TABLE` dasar dengan kunci primer atau batasan unik.

Lakukan pemuatan data menggunakan import/export alat yang direkomendasikan. Jalankan skrip yang dimodifikasi untuk langkah-langkah berikut untuk menambahkan objek basis data yang tersisa. Anda memerlukan pernyataan create table untuk menjalankan skrip ini untuk batasan, pemicu, dan indeks. Setelah skrip dibuat, hapus pernyataan create table.

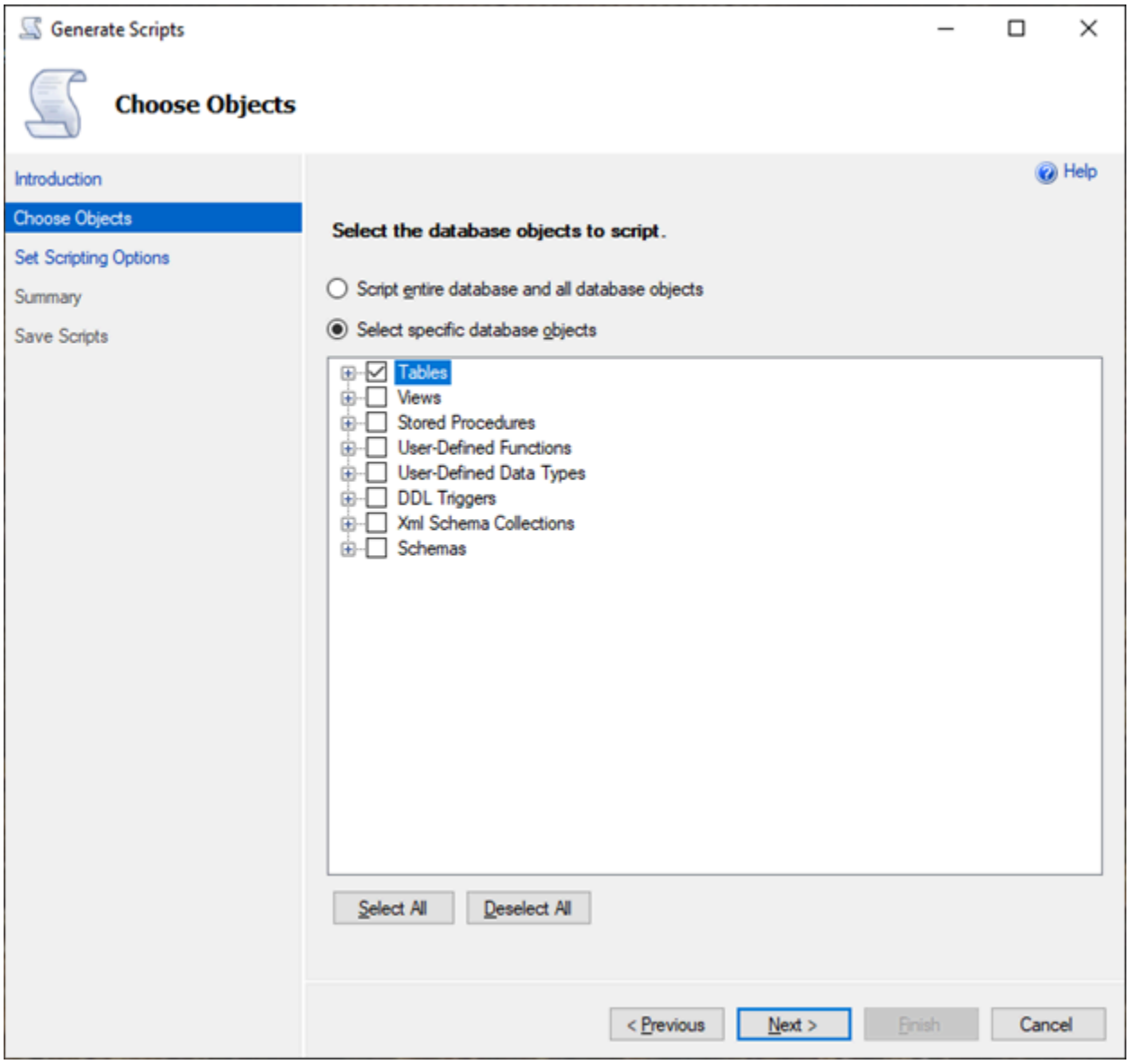
1. Jalankan pernyataan `ALTER TABLE` untuk batasan check, batasan foreign key, batasan default.
2. Jalankan pernyataan `CREATE TRIGGER`.
3. Jalankan pernyataan `CREATE INDEX`.
4. Jalankan pernyataan `CREATE VIEW`.
5. Jalankan pernyataan `CREATE STORED PROCEDURE`.

Untuk membuat skrip untuk setiap jenis objek

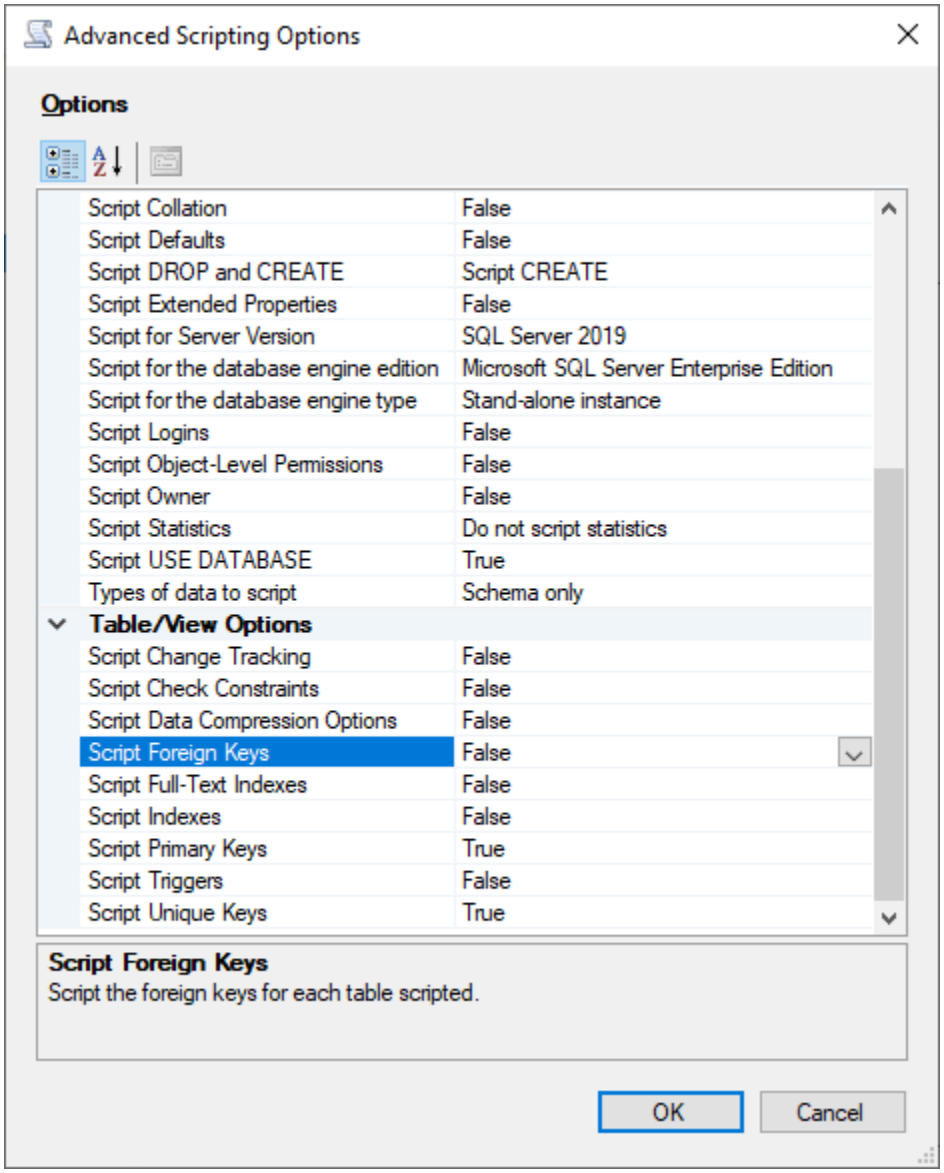
Gunakan langkah-langkah berikut untuk membuat pernyataan create table dasar menggunakan wizard Generate Scripts di SSMS. Ikuti langkah yang sama untuk membuat skrip untuk tipe objek yang berbeda.

1. Hubungkan ke instans SQL Server yang ada.
2. Buka menu konteks (klik kanan) untuk nama basis data.

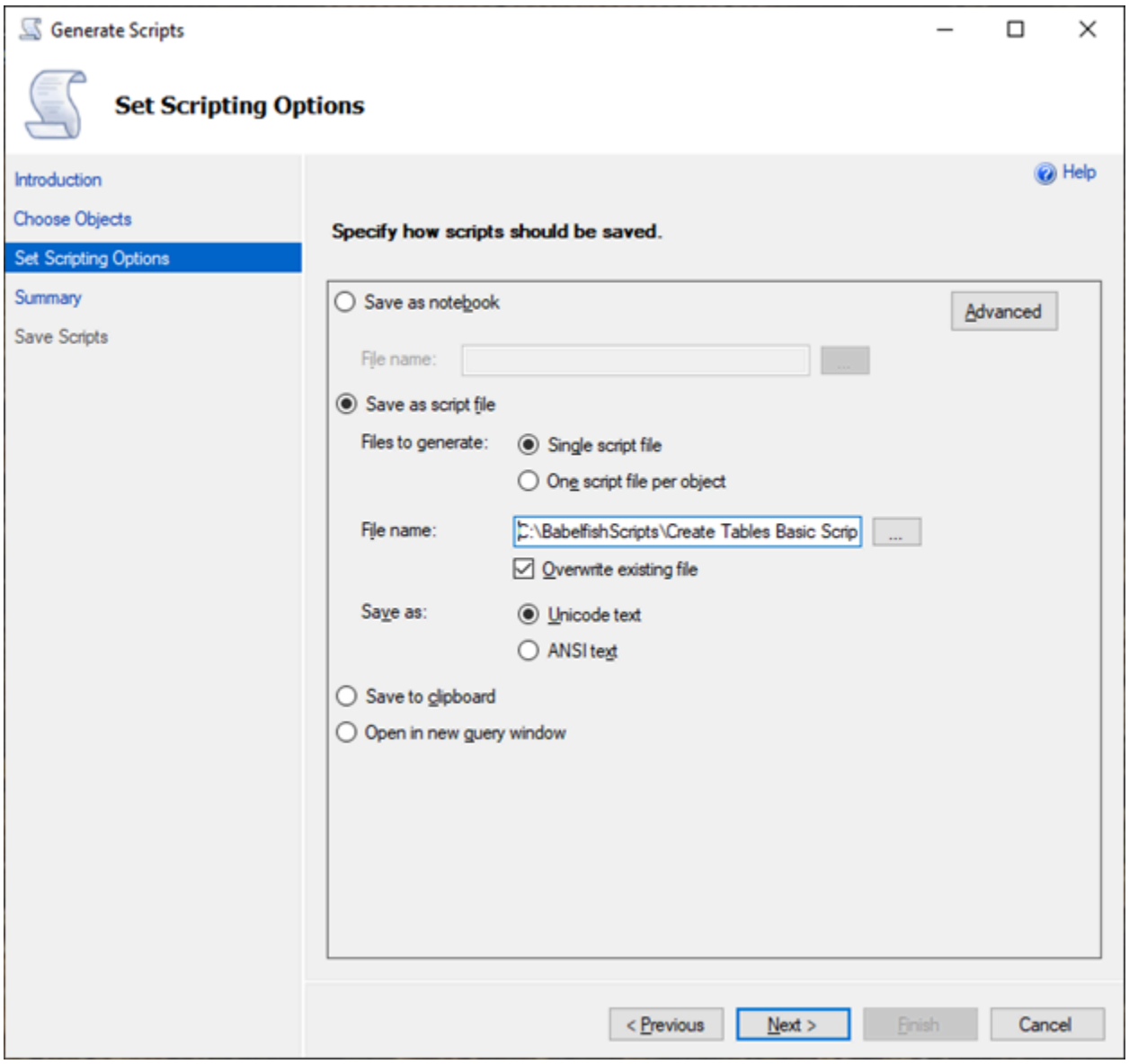
3. Pilih Tasks, lalu pilih Generate Scripts....
4. Pada panel Choose Objects, pilih Select specific database objects. Pilih Tabel, pilih semua tabel. Pilih Next untuk melanjutkan.




5. Pada halaman Set Scripting Options, pilih Advanced untuk membuka pengaturan Options. Untuk membuat pernyataan create table dasar, ubah nilai default berikut:
   - Script Defaults ke False.
   - Script Extended Properties ke False. Babelfish tidak mendukung properti yang diperluas.
   - Script Check Constraints ke False. Script Foreign Keys ke False.




6. Pilih OK.
7. Pada halaman Set Scripting Options, pilih Save as script file lalu pilih opsi Single script file. Masukkan nama File name Anda.




8. Pilih Next untuk melihat halaman Summary wizard.
9. Pilih Next untuk memulai pembuatan skrip.

   Anda dapat terus menghasilkan skrip untuk jenis objek lain di wizard. Alih-alih memilih Finish setelah file disimpan, pilih tombol Previous tiga kali untuk kembali ke halaman Choose Objects. Kemudian ulangi langkah-langkah di wizard untuk membuat skrip untuk jenis objek lainnya.

# Autentikasi basis data dengan Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL mendukung dua cara untuk mengautentikasi pengguna basis data. Autentikasi kata sandi tersedia secara default untuk semua klaster DB Babelfish. Anda juga dapat menambahkan autentikasi Kerberos untuk klaster DB yang sama.

Topik


- Autentikasi kata sandi dengan Babelfish
- Autentikasi Kerberos dengan Babelfish
- Menyiapkan otentikasi Kerberos menggunakan grup keamanan Active Directory untuk Babelfish


## Autentikasi kata sandi dengan Babelfish

Babelfish for Aurora PostgreSQL mendukung autentikasi kata sandi. Kata sandi disimpan dalam bentuk terenkripsi pada disk. Untuk informasi selengkapnya tentang autentikasi pada klaster Aurora PostgreSQL, lihat Keamanan dengan Amazon Aurora PostgreSQL.

Anda mungkin diminta untuk kredensial setiap kali Anda terhubung ke Babelfish. Setiap pengguna yang bermigrasi ke atau dibuat di Aurora PostgreSQL dapat menggunakan kredensial yang sama pada port SQL Server dan port PostgreSQL. Babelfish tidak menerapkan kebijakan kata sandi, namun sebaiknya Anda melakukan hal berikut:

- Memerlukan kata sandi kompleks yang panjangnya setidaknya delapan (8) karakter.
- Menerapkan kebijakan kedaluwarsa kata sandi.

Untuk meninjau daftar lengkap pengguna basis data, gunakan perintah`SELECT * FROM pg_user;`.

## Autentikasi Kerberos dengan Babelfish

Babelfish untuk Aurora Postgre SQL 15.2 versi mendukung otentikasi ke cluster DB Anda menggunakan Kerberos. Metode ini memungkinkan Anda menggunakan Microsoft Windows Authentication untuk mengautentikasi pengguna saat mereka terhubung ke basis data Babelfish Anda. Untuk melakukannya, Anda harus terlebih dahulu mengkonfigurasi cluster DB Anda untuk digunakan AWS Directory Service for Microsoft Active Directory untuk otentikasi Kerberos. Untuk informasi lebih lanjut, lihat Apa itu AWS Directory Service? di AWS Directory Service Panduan Administrasi.

### Mengatur autentikasi Kerberos

Babelfish untuk Aurora Postgre SQL DB cluster dapat terhubung menggunakan dua port yang berbeda, tetapi pengaturan otentikasi Kerberos adalah proses satu kali. Oleh karena itu, Anda harus terlebih dahulu mengatur autentikasi Kerberos untuk klaster DB Anda. Untuk informasi selengkapnya, lihat Mengatur autentikasi Kerberos. Setelah menyelesaikan pengaturan, pastikan Anda dapat terhubung dengan SQL klien Postgre menggunakan Kerberos. Untuk informasi selengkapnya, lihat Mengatur autentikasi Kerberos.

### Login dan penyediaan pengguna di Babelfish

Login Windows yang dibuat dari port Tabular Data Stream (TDS) dapat digunakan baik dengan TDS port atau port SQL Postgre. Pertama, login yang dapat menggunakan Kerberos untuk otentikasi harus disediakan dari TDS port sebelum digunakan oleh SQL pengguna T dan aplikasi untuk terhubung ke database Babelfish. Saat membuat login Windows, administrator dapat memberikan login menggunakan nama domain atau nama DNS BIOS domain Net. Biasanya, Net BIOS Domain adalah subdomain dari nama DNS domain. Misalnya, jika nama DNS domainnya`CORP.EXAMPLE.COM`, maka BIOS domain Net bisa`CORP`. Jika format nama BIOS domain Net disediakan untuk login, pemetaan harus ada ke nama DNS domain.

### Mengelola nama BIOS domain Net ke pemetaan nama DNS domain

Untuk mengelola pemetaan antara nama BIOS domain Net dan DNS nama domain, Babelfish menyediakan prosedur tersimpan sistem untuk menambah, menghapus, dan memotong pemetaan. Hanya pengguna dengan peran `sysadmin` yang dapat menjalankan prosedur ini.

Untuk membuat pemetaan antara Net BIOS dan nama DNS domain, gunakan prosedur tersimpan sistem yang disediakan Babelfish. `babelfish_add_domain_mapping_entry` Kedua argumen harus memiliki nilai yang valid dan tidak`NULL`.

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

Contoh berikut menunjukkan cara membuat pemetaan antara BIOS nama Net CORP dan nama DNS CORP domain. EXAMPLE. COM.

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

Untuk menghapus entri pemetaan yang ada, gunakan prosedur tersimpan sistem babelfish_remove_domain_mapping_entry.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

Contoh berikut menunjukkan cara menghapus pemetaan untuk BIOS nama CORP Net.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

Untuk menghapus entri pemetaan yang ada, gunakan prosedur tersimpan sistem babelfish_truncate_domain_mapping_table:

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

Untuk melihat semua pemetaan antara Net BIOS dan nama DNS domain, gunakan kueri berikut.

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

Mengatur Login

Membuat login

Connect ke DB melalui TDS endpoint menggunakan login yang memiliki izin yang benar. Jika tidak ada pengguna basis data yang dibuat untuk login, maka login dipetakan ke pengguna tamu. Jika pengguna tamu tidak diaktifkan, maka upaya login gagal.

Buat login Windows menggunakan kueri berikut. Opsi `FROM WINDOWS` ini memungkinkan autentikasi menggunakan Active Directory.

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

Contoh berikut menunjukkan cara membuat login untuk pengguna Active Directory [corp\test1] dengan basis data default db1.

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

Contoh ini mengasumsikan bahwa ada pemetaan antara BIOS domain Net CORP dan nama DNS domain. CORP EXAMPLE. COM. Jika tidak ada pemetaan, maka Anda harus memberikan nama DNS domain [CORP. EXAMPLE. COM\ test1].

> **Note**
>
> Login berdasarkan pengguna Active Directory, dibatasi nama kurang dari 21 karakter.

### Menjatuhkan login

Untuk menjatuhkan login, gunakan sintaks yang sama seperti untuk login lainnya, seperti yang ditunjukkan pada contoh berikut:

```
DROP LOGIN [DNS domain name\login]
```

### Mengubah login

Untuk mengubah login, gunakan sintaks yang sama seperti untuk login lainnya, seperti yang ditunjukkan pada contoh berikut:

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

ALTERLOGINPerintah ini mendukung opsi terbatas untuk login Windows, termasuk yang berikut ini:

- DISABLE— Untuk menonaktifkan login. Anda tidak dapat menggunakan login yang dinonaktifkan untuk autentikasi.
- ENABLE— Untuk mengaktifkan login yang dinonaktifkan.
- DEFAULT_ DATABASE — Untuk mengubah database default login.

> **Note**
>
> Semua manajemen kata sandi dilakukan melalui AWS Directory Service, sehingga ALTER LOGIN perintah tidak mengizinkan administrator database untuk mengubah atau mengatur kata sandi untuk login Windows.

Menghubungkan ke Babelfish untuk Aurora Postgre dengan otentikasi Kerberos SQL

Biasanya, pengguna basis data yang mengautentikasi menggunakan Kerberos melakukannya dari mesin klien mereka. Mesin ini adalah anggota domain Active Directory. Mereka menggunakan Windows Authentication dari aplikasi klien mereka untuk mengakses server Babelfish untuk Aurora SQL Postgre di port. TDS

Menghubungkan ke Babelfish untuk Aurora Postgre di port Postgre dengan otentikasi SQL Kerberos SQL

Anda dapat menggunakan login yang dibuat dari TDS port dengan port atau TDS port PostgreSQL. Namun, Postgre SQL menggunakan perbandingan case-sensitive secara default untuk nama pengguna. Agar Aurora Postgre SQL menafsirkan nama pengguna Kerberos sebagai case-insensitive, Anda harus mengatur parameter seperti pada grup parameter cluster Babelfish kustom. `krb_caseins_users` `true` Parameter ini diatur ke secara `false` default. Untuk informasi selengkapnya, lihat Mengonfigurasi nama pengguna yang tidak peka huruf besar/kecil. Selain itu, Anda harus menentukan nama pengguna login dalam format <login@ nama DNS domain> dari aplikasi klien Postgre. SQL Anda tidak dapat menggunakan format < nama DNS domain\ login>.

Kesalahan yang sering terjadi

Anda dapat mengonfigurasi hubungan trust hutan antara Microsoft Active Directory lokal dan AWS Managed Microsoft AD. Untuk informasi selengkapnya, lihat Membuat hubungan kepercayaan. Kemudian, Anda harus terhubung menggunakan titik akhir khusus domain khusus alih-alih menggunakan domain Amazon `rds.amazonaws.com` di titik akhir host. Jika Anda tidak menggunakan titik akhir khusus domain yang benar, Anda mungkin mengalami kesalahan berikut:

```
Error: “Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

Kesalahan ini terjadi ketika TDS klien tidak dapat menyimpan tiket layanan untuk titik akhir URL yang disediakan. Untuk informasi selengkapnya, lihat Mengatur autentikasi Kerberos.

## Menyiapkan otentikasi Kerberos menggunakan grup keamanan Active Directory untuk Babelfish

Mulai dari Babelfish versi 4.2.0, Anda dapat mengatur otentikasi Kerberos untuk Babelfish dengan grup keamanan Active Directory. Berikut ini adalah prasyarat yang harus diselesaikan untuk menyiapkan otentikasi Kerberos menggunakan Active Directory:

- Anda harus mengikuti semua langkah yang disebutkan diAutentikasi Kerberos dengan Babelfish.
- Pastikan bahwa instans DB dikaitkan dengan Active Directory. Untuk memverifikasi ini, Anda dapat melihat status keanggotaan domain di konsol atau dengan menjalankan describe-db-instances AWS CLIperintah.

  Status instans DB harus diaktifkan kerberos. Untuk informasi selengkapnya tentang memahami keanggotaan domain, lihatMemahami keanggotaan Domain.
- Verifikasi pemetaan antara nama BIOS domain Net dan nama DNS domain menggunakan kueri berikut:

  ```
  SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
  ```
- Sebelum melanjutkan lebih jauh, verifikasi otentikasi Kerberos menggunakan login individu berfungsi seperti yang diharapkan. Koneksi menggunakan otentikasi Kerberos sebagai pengguna Active Directory harus berhasil. Jika Anda menghadapi masalah apa pun, lihatKesalahan yang sering terjadi.

### Menyiapkan ekstensi pg_ad_mapping

Anda harus mengikuti semua langkah yang disebutkan diMenyiapkan ekstensi pg_ad_mapping. Untuk memverifikasi bahwa ekstensi diinstal, jalankan kueri berikut dari TDS titik akhir:

```
1> SELECT extname, extversion FROM pg_extension where extname like 'pg_ad_mapping';
2> GO
extname       extversion
```

```
------------ ----------
pg_ad_mapping 0.1

(1 rows affected)
```

Mengelola Login Grup

Buat login grup dengan mengikuti langkah-langkah yang disebutkan diMengatur Login. Kami menyarankan agar nama login sama dengan nama grup keamanan Active Directory (AD) untuk perawatan yang lebih mudah, meskipun itu tidak wajib. Sebagai contoh:

```
CREATE LOGIN [corp\accounts-group] FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Memetakan login SQL grup T dengan grup keamanan AD

Anda perlu secara eksplisit menyediakan Login Grup T- SQL Windows untuk setiap grup keamanan AD yang memerlukan akses ke server database. Pengguna AD, yang merupakan bagian dari setidaknya satu grup keamanan AD yang disediakan, akan mendapatkan akses ke server database.

> **Note**
>
> SQLLogin T ini tidak dapat lagi mengautentikasi menggunakan otentikasi berbasis kata sandi.

Misalnya, grup akun adalah grup keamanan di AD dan jika Anda ingin menyediakan grup keamanan ini di Babelfish, Anda harus menggunakan format [corp\ accounts-group].

- Grup Keamanan AD: grup akun
- TSQLMasuk: [corp\ akun-grup]
- Peran PG setara untuk TSQL Login yang diberikan: CORP accounts-group@. EXAMPLE. COM

Admin sekarang dapat melanjutkan untuk membuat pemetaan antara grup keamanan AD dan T- SQL login dari SQL titik akhir Postgre melalui perintah psql berikut. Untuk informasi selengkapnya tentang penggunaan fungsi, lihatMenggunakan fungsi dari `pg_ad_mapping` ekstensi.

> **Note**
>
> T- SQL login harus ditentukan dalam FQDN format login_name@ sambil menambahkan pemetaan. Bobot diabaikan saat menghubungkan melalui titik TDS akhir. Untuk informasi

selengkapnya tentang penggunaan bobot, lihatMenghubungkan ke Babelfish melalui titik SQL akhir Postgre di port Postgre SQL.

```
postgres=>select pgadmap_set_mapping('accounts-group', 'accounts-
group@CORP.EXAMPLE.COM', <SID>, <Weight>);
```

Untuk informasi tentang pengambilan SID grup keamanan AD, lihatMengambil Grup SID Direktori Aktif di PowerShell.

Tabel berikut menunjukkan contoh pemetaan dari grup keamanan AD ke SQL login T-:

| Grup Keamanan AD | TSQLLogin | Peran PG setara untuk TSQL Login yang diberikan | Berat Badan |
|---|---|---|---|
| akun-grup | [korp\ akun-grup] | akun-grup@CORP. EXAMPLE. COM | 7 |
| kelompok penjualan | [corp\ kelompok penjualan] | kelompok penjualan @CORP. EXAMPLE. COM | 10 |
| kelompok pengembang | [korp\ dev-grup] | dev-group@CORP. EXAMPLE. COM | 7 |

```
postgres=> select admap.ad_sid, admap.ad_grp, lgn.orig_loginname, lgn.rolname,
 admap.weight from pgadmap_read_mapping() as admap, sys.babelfish_authid_login_ext as
 lgn where admap.pg_role = lgn.rolname;
    ad_sid    |     ad_grp     |    orig_loginname   |            rolname
 | weight
--------------+----------------+---------------------+--------------------------------
+--------
 S-1-5-67-890 | accounts-group | corp\accounts-group  | accounts-group@CORP.EXAMPLE.COM
 |     7
 S-1-2-34-560 | sales-group    | corp\sales-group     | sales-group@CORP.EXAMPLE.COM
 |     10
```

```
 S-1-8-43-612 | dev-group      | corp\dev-group       | dev-group@CORP.EXAMPLE.COM
 |     7
 (7 rows)
```

Menghubungkan ke Babelfish melalui titik akhir TDS

Dalam contoh berikut, user1 adalah anggota accounts-group dan sales-group, user2 adalah anggota accounts-group dan dev-group.

| nama pengguna | Keanggotaan Grup Keamanan AD |
|---|---|
| user1 | akun-grup, grup penjualan |
| user2 | akun-grup, dev-group |

Connect ke server database Babelfish menggunakan utilitas sqlcmd. Anda dapat memverifikasi apakah pengguna (user1 dalam contoh ini) telah diautentikasi menggunakan Kerberos dengan mengikuti contoh di bawah ini:

```
1> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();
2>  GO
principal               gss_authenticated
----------------------  -----------------
user1@CORP.EXAMPLE.COM  1

((1 rows affected))
1> select suser_name();
2>  GO
suser_name
----------
corp\user1

(1 rows affected)
```

Dalam contoh ini, user1 akan mewarisi hak istimewa akun-grup dan grup penjualan. Anda dapat memverifikasi keanggotaan grup menggunakan tampilan `sys.login_token` sistem.

```
1> SELECT name, type FROM sys.login_token;
```

```
2>  GO
name                type
------------------- ----
corp\accounts-group WINDOWS GROUP
corp\sales-group    WINDOWS GROUP

(2 rows affected)
```

Audit dan Pencatatan

Untuk menentukan identitas utama keamanan AD, gunakan perintah berikut:

```
1> select suser_name();
2> GO
suser_name
----------
corp\user1

(1 rows affected)
```

Saat ini, identitas pengguna AD tidak terlihat di log. Anda dapat mengaktifkan `log_connections` parameter untuk mencatat pembentukan sesi DB. Untuk informasi selengkapnya, lihat log_connections. Outputnya mencakup identitas pengguna AD sebagai prinsipal seperti yang ditunjukkan pada contoh berikut. Backend PID yang terkait dengan output ini kemudian dapat membantu tindakan atribut kembali ke pengguna AD yang sebenarnya.

```
bbf_group_ad_login@babelfish_db:[615]:LOG: connection authorized:
 user=bbf_group_ad_login database=babelfish_db application_name=sqlcmd GSS
 (authenticated=yes, encrypted=yes, principal=user1@CORP.EXAMPLE.COM)
```

Memanfaatkan hak istimewa keanggotaan grup keamanan AD

Mewarisi hak istimewa tingkat server

Pengguna AD yang merupakan anggota grup keamanan AD yang diberikan akan mewarisi hak istimewa tingkat server yang diberikan kepada login grup Windows yang dipetakan. Misalnya, pertimbangkan grup keamanan `accounts-group` AD, yang diberikan keanggotaan ke peran `sysadmin` server di Babelfish. Anda dapat mewarisi hak istimewa tingkat server menggunakan perintah berikut:

```
1> ALTER SERVER ROLE sysadmin ADD MEMBER [corp\accounts-group];
```

Akibatnya, setiap pengguna Active Directory yang merupakan anggota grup keamanan `accounts-group` AD akan mewarisi hak istimewa tingkat server yang terkait dengan peran tersebut. `sysadmin` Ini berarti bahwa pengguna seperti`corp\user1`, menjadi anggota`accounts-group`, sekarang akan memiliki kemampuan untuk melakukan operasi tingkat server dalam Babelfish.

> **Note**
>
> Untuk melakukan tingkat serverDDLs, login Windows untuk pengguna AD individu harus ada. Untuk informasi selengkapnya, lihat Batasan.

Mewarisi hak istimewa tingkat basis data

Untuk memberikan hak istimewa tingkat database, pengguna database harus dibuat dan dipetakan dengan login grup Windows. Pengguna AD yang merupakan anggota grup keamanan AD yang diberikan akan mewarisi hak istimewa tingkat database yang diberikan kepada pengguna database tersebut. Dalam contoh berikut, Anda dapat melihat bagaimana hak istimewa tingkat basis data untuk grup Windows [corp\ accounts-group] ditetapkan.

```
1> CREATE DATABASE db1;
2> GO
1> USE db1;
2> GO
Changed database context to 'db1'.
1> CREATE TABLE dbo.t1(a int);
2> GO
```

Buat pengguna database [corp\ sales-group] untuk login grup Windows [corp\ accounts-group]. Untuk melakukan langkah ini, hubungkan melalui TDS endpoint menggunakan login yang merupakan anggota sysadmin.

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group];
2> GO
```

Sekarang, hubungkan sebagai pengguna AD user1 untuk memeriksa akses tabel t1. Karena kami belum memberikan hak istimewa tingkat basis data, itu akan menghasilkan kesalahan izin ditolak.

```
1> SELECT * FROM dbo.t1;
2> GO
Msg 33557097, Level 16, State 1, Server db-inst, Line 1
permission denied for table t1
```

Berikan SELECT pada tabel t1 kepada pengguna database [corp\ accounts-group]. Untuk melakukan langkah ini, hubungkan melalui TDS endpoint menggunakan login yang merupakan anggota sysadmin.

```
1> GRANT SELECT ON dbo.t1 TO [corp\accounts-group];
2> GO
```

Connect as AD user user1 untuk memvalidasi akses.

```
1> SELECT * FROM dbo.t1;
2> GO
a
-----------

(0 rows affected)
```

Menangani perilaku DDL Pernyataan berdasarkan skema default atau eksplisit

Saat menggunakan sesi yang diautentikasi AD, skema default untuk sesi saat ini ditentukan oleh kondisi berikut:

- Jika pengguna database individu ada, skema default pengguna dianggap sebagai skema default sesi saat ini.
- Jika skema default untuk pengguna database grup ada, skema default pengguna database grup dianggap sebagai skema default sesi saat ini dengan id utama terkecil.

Memahami perilaku CREATE DDL pernyataan

Jika tidak ada skema eksplisit yang ditentukan dalam CREATE DDL pernyataan maka pembuatan objek akan berlangsung dalam skema default sesi saat ini. Jika skema tidak dapat ditentukan apakah default atau eksplisit, maka DDL pernyataan tersebut akan memunculkan kesalahan berikut:

```
"Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command."
```

Example : Skema default tidak ada untuk pengguna grup Windows

Pengguna grup Windows [corp\ accounts-group] memiliki skema NULL default dan pengguna AD user1 mencoba melakukan tanpa menentukan skema secara eksplisit. DDL Karena login windows individu dan pengguna tidak ada untuk user1, itu akan mendapatkan hak istimewa tingkat database pengguna grup Windows [corp\ accounts-group] saja.

```
1> create TABLE t2(a int);
2> GO

Msg 33557097, Level 16, State 1, Server db-inst, Line 1
Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command.
```

> **Note**
>
> Login dan pengguna windows individual tidak ada untuk pengguna pengguna AD1

Example : Skema default ada untuk pengguna grup Windows

Buat pengguna grup Windows untuk login [corp\ accounts-group] dengan skema default menggunakan sysadmin.

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group] WITH
 DEFAULT_SCHEMA = sch_acc;
2> GO
1> CREATE SCHEMA sch_acc AUTHORIZATION [gad\accounts-group];
2> GO
1> SELECT name, principal_id, default_schema_name FROM sys.database_principals WHERE
 name = 'corp\accounts-group';
2> GO

name               principal_id default_schema_name
------------------ ------------ ------------------
corp\accounts-group 24162        sch_acc

(1 rows affected)
```

Coba buat objek tanpa menentukan skema secara eksplisit menggunakan pengguna AD user1. Tabel t2 akan dibuat dalam skema default [corp\ accounts-group] Pengguna grup Windows Pemilik objek ini akan sama dengan pemilik skema sch_acc.

```
1> CREATE TABLE t_group(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't_group';
2> GO

name    schema_name
------- -----------
t_group sch_acc

(1 rows affected)
```

> **Note**
>
> Login dan pengguna windows individual tidak ada untuk pengguna pengguna AD1

Example : Pengguna database individu juga ada untuk pengguna AD

Jika pengguna database individu juga ada untuk pengguna AD maka objek akan selalu dibuat dalam skema yang terkait dengan pengguna database individu. Jika skema tidak ada untuk pengguna database maka skema dbo akan digunakan. Buat login windows individual dan pengguna database untuk pengguna pengguna AD1. Connect melalui TDS endpoint menggunakan login sysadmin

```
1> CREATE LOGIN [corp\user1] FROM WINDOWS;
2> GO
1> CREATE USER [corp\user1] FOR LOGIN [corp\user1] WITH DEFAULT_SCHEMA = sch1;
2> GO
1> CREATE SCHEMA sch1 AUTHORIZATION [corp\user1];
2> GO
1> SELECT name, default_schema_name FROM sys.database_principals WHERE name = 'corp
\user1';
2> GO

name      default_schema_name
--------- -------------------
corp\user1 sch1

(1 rows affected)
```

Connect menggunakan pengguna AD user1 dan coba buat objek tanpa menentukan skema secara eksplisit. Tabel t2 akan dibuat dalam skema sch1. Perhatikan juga bahwa pemilik objek ini akan sama dengan pemilik skema sch1.

```
1> CREATE TABLE t2(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't2';
2> GO

name schema_name
---- -----------
t2   sch1

(1 rows affected)
```

Batasan

- Utilitas Dump/Restore tidak mendukung dumping pemetaan ekstensi pg_ad_mapping. Anda perlu membuat ulang pemetaan tersebut setelah pemulihan.
- Penerapan Biru-Hijau tidak didukung untuk instance Babelfish dan Aurora Postgre dengan. SQL `pg_ad_mapping`
- Pembuatan skema implisit tidak didukung. DDLpernyataan yang membutuhkan pembuatan skema implisit tidak didukung.
- DDLsALTERAUTHORIZATIONON tingkat serverDATABASE,,, CREATE DATABASE CREATE LOGIN ALTER LOGIN ALTER SERVERROLE, tidak ALTER DATABASE didukung dalam sesi otentikasi Grup AD ketika login Windows individu tidak ada, hanya login Windows grup yang ada. Untuk mengatasi batasan ini, Disarankan untuk melakukan operasi ini dalam sesi otentikasi kata sandi atau membuat login Windows individual.
- Pembuatan pengguna implisit tidak didukung. SQLPerilaku T yang ideal [belum didukung di Babelfish]; Dalam beberapa kasus seperti DDL dan pernyataan kontrol akses GRANT seperti/ REVOKE di mana nama pengguna AD ditentukan dalam perintah tetapi tidak ada dalam database maka pengguna database bernama sebagai pengguna AD dibuat secara implisit.
- Untuk DDLs dalam SQL Prosedur atau Fungsi PL/PG yang dibuat dari PSQL titik akhir dan dieksekusi dari titik akhir di sesi TDS otentikasi Grup AD:
  - ALTER/DROPpernyataan akan didukung.
  - CREATETABLE, CREATEVIEW, CREATEINDEX, CREATEFUNCTION/PROC, CREATETYPE, CREATESEQUENCE, CREATETRIGGER, SELECTINTO, CREATE FULLTEXTINDEX, CREATE UNIQUE INDEX akan menimbulkan kesalahan jika skema tidak disediakan secara eksplisit dan skema default adalah nol untuk sesi saat ini.
  - CREATEDATABASE, CREATE EXTENSION dan semua CREATE pernyataan lain untuk PG (bukan dalam T-SQL) objek tertentu CREATE langganan, CREATE tablespace, CREATE kebijakan, CREATE konversi tidak akan didukung.
- DDLsdari SQL titik akhir Postgre tidak didukung dalam sesi yang diautentikasi Grup AD. Sebagai solusinya, Anda selalu dapat terhubung menggunakan pengguna master atau pengguna lain menggunakan mekanisme otentikasi berbasis kata sandi.
- Objek sistem seperti SUSER _ SID (), IS_ (), IS_ SRVROLEMEMBER MEMBER (), sys.dm_exec_sessions memiliki batasan berikut.

- SUSER_ SID () tidak akan mengembalikan SID kapan Pengguna AD atau Grup Keamanan AD disediakan.
- IS_ SRVROLEMEMBER () tidak akan mempertimbangkan keanggotaan peran jika pengguna AD saat ini mewarisi keanggotaan peran server dari keanggotaan peran server login grup Windows apa pun.
- IS_ MEMBER () akan mengembalikan false untuk kueri terkait Grup Windows apa pun.
- sys.dm_exec_sessions tidak akan menampilkan nilai yang diharapkan login_name, kolom nt_user_name.

Menghubungkan ke Babelfish melalui titik SQL akhir Postgre di port Postgre SQL

Anda dapat menggunakan login grup yang dibuat dari TDS port untuk terhubung melalui port Postgre SQL juga. Untuk terhubung melalui SQL port Postgre, Anda perlu menentukan nama pengguna AD dalam format `<ad_username@FQDN>` dari aplikasi klien SQL Postgre. Anda tidak dapat menggunakan `<DNS domain name\ad_username>` format.

Postgre SQL menggunakan perbandingan case-sensitive secara default untuk nama pengguna. Agar Aurora Postgre SQL menafsirkan nama pengguna Kerberos sebagai case-insensitive, Anda harus menyetel parameter krb_caseins_users sebagai true di grup parameter cluster Babelfish kustom. Parameter ini diatur ke false secara default. Untuk informasi selengkapnya, lihat Mengonfigurasi klaster DB Aurora PostgreSQL Anda untuk nama pengguna yang tidak peka huruf besar/kecil.

Perbedaan perilaku antara SQL titik akhir T- SQL dan Postgre ketika pengguna AD adalah bagian dari beberapa grup

Pertimbangkan bahwa pengguna AD user1 adalah bagian dari dua grup keamanan AD [corp\ accounts-group] dan [corp\ sales-group] dan admin DB telah mengatur pemetaan pengguna dengan cara berikut.

```
postgres=> select * from pgadmap_read_mapping();

ad_sid       | pg_role                          | weight | ad_grp
-------------+----------------------------------+--------+---------------
S-1-5-67-980 | accounts-group@CORP.EXAMPLE.COM | 7      | accounts-group
S-1-2-34-560 | sales-group@CORP.EXAMPLE.COM    | 10     | sales-group
(2 rows)
```

Jika pengguna terhubung dari SQL titik akhir T- maka selama otorisasi, itu akan mewarisi hak istimewa dari semua login T terkait. SQL Dalam contoh ini user1 akan mewarisi gabungan hak istimewa dari kedua login SQL T-group dan bobot akan diabaikan. Ini sejalan dengan SQL perilaku T standar.

Namun jika pengguna yang sama terhubung dari SQL titik akhir Postgre maka ia dapat mewarisi hak istimewa hanya dari satu SQL login T terkait dengan bobot tertinggi. Jika dua login SQL grup T diberi bobot yang sama maka pengguna AD akan mewarisi hak istimewa SQL login T yang sesuai dengan pemetaan yang ditambahkan baru-baru ini. Untuk PostgreSQL, rekomendasinya adalah menentukan bobot yang mencerminkan izin/hak istimewa relatif dari peran DB individu untuk menghindari ambiguitas. Dalam contoh di bawah ini, user1 terhubung melalui PSQL titik akhir dan hanya mewarisi hak istimewa grup penjualan.

```
babelfish_db=> select session_user, current_user;

   session_user               |   current_user
------------------------------+--------------------------
 sales-group@CORP.EXAMPLE.COM | sales-group@CORP.EXAMPLE.COM
(1 row)


babelfish_db=> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();

     principal          | gss_authenticated
------------------------+-------------------
 user1@CORP.EXAMPLE.COM | t
(1 row)
```

# Menghubungkan ke klaster DB Babelfish

Untuk terhubung ke Babelfish, Anda terhubung ke titik akhir klaster Aurora PostgreSQL yang menjalankan Babelfish. Klien Anda dapat menggunakan salah satu driver klien berikut yang sesuai dengan TDS versi 7.1 hingga 7.4:

- Open Database Connectivity (ODBC)
- Driver DB OLE/MSOLEDBSQL
- Java Database Connectivity (JDBC) versi 8.2.2 (mssql-jdbc-8.2.2) dan lebih baru
- Penyedia SqlClient Data Microsoft untuk SQL Server
- Penyedia Data .NET untuk SQL Server
- SQL Server Native Client 11.0 (tidak digunakan lagi)
- OLE DB Provider/SQLOLEDB (usang)

Dengan Babelfish, Anda menjalankan hal-hal berikut:

- Alat, aplikasi, dan sintaks SQL Server pada port TDS, secara default port 1433.
- Alat, aplikasi, dan sintaks PostgreSQL pada port PostgreSQL, secara default port 5432.

Untuk mempelajari selengkapnya tentang menghubungkan ke Aurora PostgreSQL secara umum, lihat Menghubungkan ke klaster DB Amazon Aurora PostgreSQL..

> **Note**
>
> Alat pengembang pihak ketiga yang menggunakan penyedia SQL Server OLEDB untuk mengakses metadata tidak didukung. Kami menyarankan Anda untuk menggunakan koneksi klien SQL Server JDBC, ODBC, atau SQL Native untuk alat ini.

Dimulai dengan Babelfish versi 5.1.0, enkripsi end-to-end koneksi diberlakukan secara default. Untuk memastikan konektivitas berkelanjutan:

- Konfigurasikan SSL/TLS enkripsi untuk koneksi Anda. Untuk informasi selengkapnya, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .
- Impor sertifikat yang diperlukan di komputer klien Anda. Untuk informasi selengkapnya, lihat Menggunakan SSL dengan instans Microsoft SQL Server DB.

Jika Anda ingin terus menggunakan pengaturan enkripsi dari versi Babelfish sebelumnya (sebelum versi 5.1.0), Anda dapat mengatur `rds.force_ssl` parameter ke `0` dalam grup parameter cluster DB Anda.

Topik

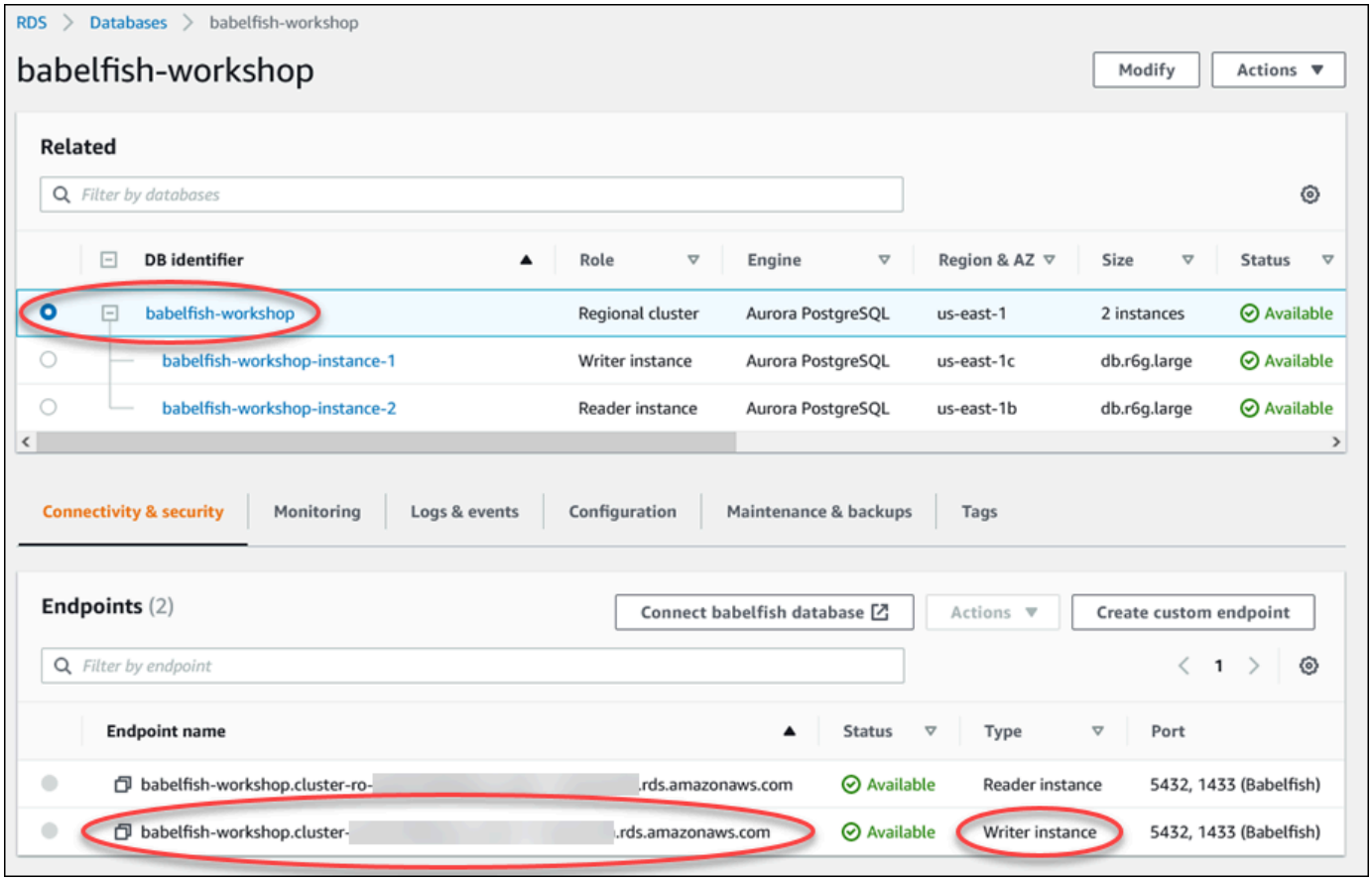
- Menemukan titik akhir dan nomor port penulis
- Membuat koneksi klien C# atau JDBC ke Babelfish
- Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda
- Menggunakan klien PostgreSQL untuk terhubung ke klaster DB Anda

## Menemukan titik akhir dan nomor port penulis

Untuk terhubung ke klaster DB Babelfish, Anda menggunakan titik akhir yang terkait dengan instans penulis (utama) klaster DB. Instans harus memiliki status Tersedia. Diperlukan waktu hingga 20 menit agar instans tersedia setelah membuat klaster DB Babelfish for Aurora PostgreSQL.

Menemukan titik akhir basis data Anda

1. Buka konsol untuk Babelfish.
2. Pilih Basis data dari panel navigasi.
3. Pilih klaster DB Babelfish for Aurora PostgreSQL Anda dari yang terdaftar untuk melihat detailnya.
4. Pada tab Konektivitas & keamanan, perhatikan nilai Titik Akhir klaster yang tersedia. Gunakan titik akhir klaster untuk instans penulis dalam string koneksi Anda untuk aplikasi apa pun yang melakukan operasi tulis atau baca pada basis data.




Untuk informasi selengkapnya tentang detail klaster DB Aurora, lihat Membuat klaster DB Amazon Aurora.

## Membuat koneksi klien C# atau JDBC ke Babelfish

Berikut ini Anda dapat menemukan beberapa contoh menggunakan kelas C# dan JDBC untuk terhubung ke Babelfish for Aurora PostgreSQL.

Example : Menggunakan kode C # untuk terhubung ke klaster DB

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionString = connectionString + ";ServerCertificate=/path/to/certificate.pem";


SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example : Menggunakan kelas dan antarmuka API JDBC generik untuk terhubung ke klaster DB

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionUrl = connectionUrl + ";serverCertificate=/path/to/certificate.pem";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example : Menggunakan kelas dan antarmuka JDBC khusus SQL Server untuk terhubung ke klaster DB

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
```

```
String babelfishServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";

ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");

// [optional] To validate server certificate during TLS/SSL connection
ds.setServerCertificate("/path/to/certificate.pem");

Connection con = ds.getConnection();
```

**Important**

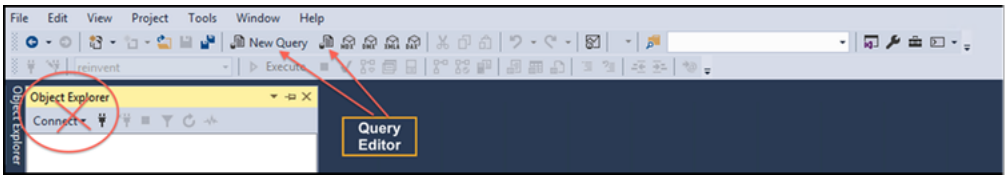
Pastikan sertifikat cocok dengan Otoritas Sertifikat yang ditampilkan dalam konfigurasi cluster DB Anda pada AWS Management Console.

## Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda

Anda dapat menggunakan klien SQL Server untuk terhubung dengan Babelfish pada port TDS. Pada Babelfish 2.1.0 dan rilis yang lebih tinggi, Anda dapat menggunakan SSMS Object Explorer atau SSMS Query Editor untuk terhubung ke klaster Babelfish Anda.

Batasan

- Di Babelfish 2.1.0 dan versi yang lebih lama, menggunakan `PARSE` untuk memeriksa sintaks SQL tidak berfungsi sebagaimana mestinya. Daripada memeriksa sintaks tanpa menjalankan kueri, perintah `PARSE` menjalankan kueri tetapi tidak menampilkan hasil apa pun. Menggunakan <Ctrl><F5>kombinasi tombol SMSS untuk memeriksa sintaks memiliki perilaku anomali yang sama, yaitu, Babelfish secara tak terduga menjalankan kueri tanpa memberikan output apa pun.
- Babelfish tidak mendukung MARS (Multiple Active Result Sets). Pastikan bahwa setiap aplikasi klien yang Anda gunakan untuk terhubung ke Babelfish tidak diatur untuk menggunakan MARS.
- Untuk Babelfish 1.3.0 dan versi yang lebih lama, hanya Editor Kueri yang didukung untuk SSMS. Untuk menggunakan SSMS dengan Babelfish, pastikan untuk membuka dialog koneksi Editor Kueri di SSMS, dan bukan Object Explorer. Jika dialog Object Explorer terbuka, batalkan dialog dan buka kembali Editor Kueri. Pada gambar berikut, Anda dapat menemukan opsi menu untuk dipilih saat menghubungkan ke Babelfish 1.3.0 atau versi yang lebih lama.




Untuk informasi selengkapnya tentang interoperabilitas dan perbedaan perilaku antara SQL Server dan Babelfish, lihat Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server.

Menggunakan sqlcmd untuk terhubung ke klaster DB

Anda dapat terhubung ke dan berinteraksi dengan cluster Aurora PostgreSQL DB yang mendukung Babelfish dengan hanya menggunakan versi 19.1 dan klien baris perintah SQL Server sebelumnya. `sqlcmd` SSMS versi 19.2 tidak didukung untuk terhubung ke cluster Babelfish. Gunakan perintah berikut ini untuk terhubung.

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

Opsinya adalah sebagai berikut:

- `-S` adalah titik akhir dan port TDS (opsional) dari klaster DB.
- `-U` adalah nama login pengguna.
- `-P` adalah kata sandi yang terkait dengan pengguna.
- `-d` adalah nama basis data Babelfish Anda.

Setelah terhubung, Anda dapat menggunakan banyak perintah yang sama yang Anda gunakan dengan SQL Server. Sebagai contoh, lihat Mendapatkan informasi dari katalog sistem Babelfish.

Menggunakan SSMS untuk terhubung ke klaster DB

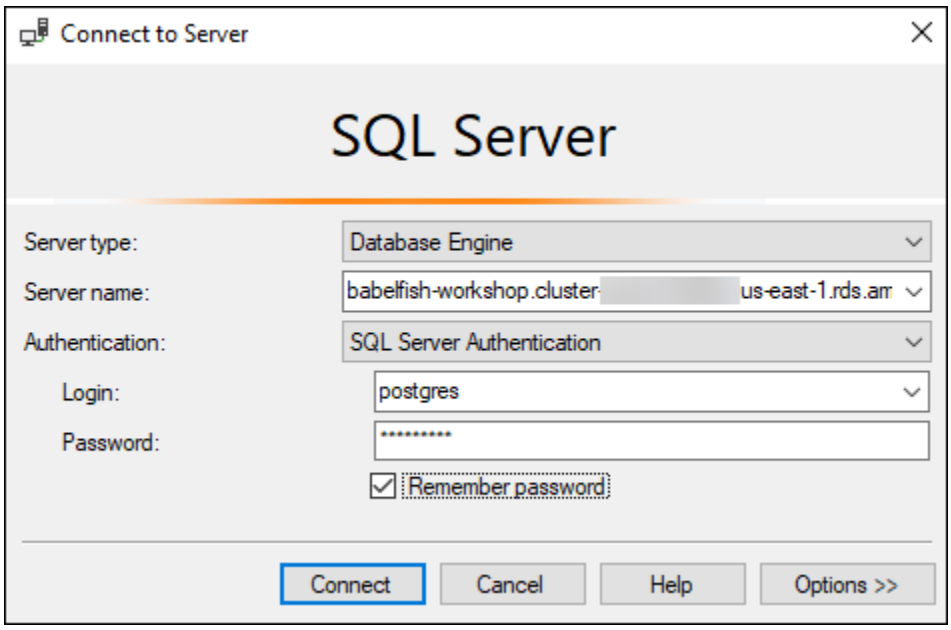
Anda dapat terhubung ke klaster DB Aurora PostgreSQL yang menjalankan Babelfish menggunakan Microsoft SQL Server Management Studio (SSMS). SSMS mencakup berbagai alat, termasuk Wizard Impor dan Ekspor SQL Server dibahas di Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL. Untuk informasi selengkapnya tentang SSMS, lihat Download SQL Server Management Studio (SSMS) dalam dokumentasi Microsoft. Untuk mengonfigurasi SSL/TLS, lihat Menggunakan SSL dengan instans Microsoft SQL Server DB.

Untuk terhubung ke basis data Babelfish Anda dengan SSMS

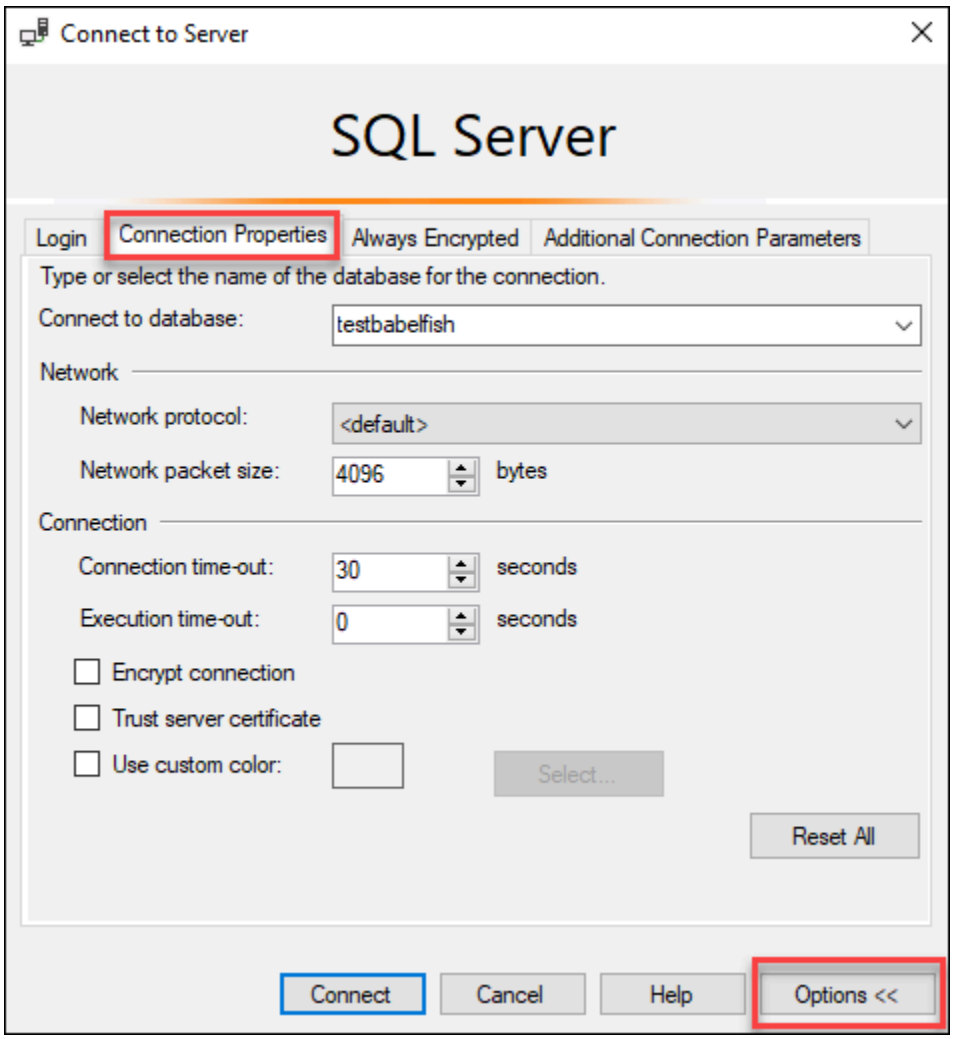
1. Mulai SSMS.
2. Buka kotak dialog Hubungkan ke Server. Untuk melanjutkan koneksi, lakukan salah satu hal berikut:
   - Pilih Kueri Baru.
   - Jika Editor Kueri terbuka, pilih Kueri, Koneksi, Hubungkan.
3. Berikan informasi berikut untuk basis data Anda:
   a. Untuk Jenis server, pilih Mesin Basis Data.
   b. Untuk Nama server, masukkan nama DNS. Misalnya, nama server Anda akan terlihat seperti berikut.

      ```
      cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
      ```

   c. Untuk Autentikasi, pilih Autentikasi SQL Server.
   d. Untuk Login, masukkan nama pengguna yang dipilih saat Anda membuat basis data.
   e. Untuk Kata Sandi, masukkan kata sandi yang dipilih saat Anda membuat basis data.




4. (Opsional) Pilih Opsi, lalu pilih tab Properti Koneksi.




5. (Opsional) Untuk Hubungkan ke basis data, tentukan nama basis data SQL Server yang dimigrasi untuk disambungkan, dan pilih Hubungkan.

   Jika muncul pesan yang menunjukkan bahwa SSMS tidak dapat menerapkan string koneksi, pilih OK.

   Jika Anda mengalami kesulitan menghubungkan ke Babelfish, lihat Kegagalan koneksi.

   Untuk informasi selengkapnya tentang masalah koneksi SQL Server, lihat Memecahkan masalah koneksi ke instans DB SQL Server Anda di Panduan Pengguna RDS Amazon.

## Menggunakan klien PostgreSQL untuk terhubung ke klaster DB Anda

Anda dapat menggunakan klien PostgreSQL untuk terhubung ke Babelfish pada port PostgreSQL. Dimulai dengan versi 5.1.0, server Babelfish memberlakukan enkripsi end-to-end koneksi secara default. Perbarui aplikasi Anda untuk bekerja dengan SSL/TLS certificates. For more information about configuring SSL/TLS sertifikat, lihatthe section called “Mengamankan data Aurora PostgreSQL dengan SSL/TLS”.

Menggunakan psql untuk terhubung ke klaster DB

Anda dapat mengunduh klien PostgreSQL dari situs web PostgreSQL. Ikuti petunjuk khusus untuk versi sistem operasi Anda untuk menginstal psql.

Anda dapat menanyakan klaster DB PostgreSQL Aurora yang mendukung Babelfish dengan klien baris perintah `psql`. Saat menghubungkan, gunakan port PostgreSQL (secara default, port 5432). Biasanya, Anda tidak perlu menentukan nomor port kecuali Anda mengubahnya dari default. Gunakan perintah berikut untuk terhubung ke Babelfish dari klien `psql`:

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

Parameter tersebut adalah sebagai berikut:

- `-h` – Nama host klaster DB (titik akhir klaster) yang ingin Anda akses.
- `-p` – Nomor port PostgreSQL yang digunakan untuk terhubung ke instans DB Anda.
- `-d` – Basis data yang ingin Anda hubungkan. Default-nya adalah `babelfish_db`.
- `-U` – Akun pengguna basis data yang ingin Anda akses. (Contoh menunjukkan nama pengguna utama default.)

Ketika Anda menjalankan perintah SQL pada klien psql, Anda mengakhiri perintah dengan titik koma. Misalnya, perintah SQL berikut meminta tampilan sistem pg_tables untuk mengembalikan informasi tentang setiap tabel di basis data.

```
SELECT * FROM pg_tables;
```

Klien psql juga memiliki satu set metacommand default. Metacommand adalah pintasan yang menyesuaikan pemformatan atau menyediakan pintasan yang mengembalikan meta-data dalam

format. easy-to-use Misalnya, metacommand berikut mengembalikan informasi yang mirip dengan perintah SQL sebelumnya:

```
\d
```

Metacommand tidak perlu diakhiri dengan titik koma (;).

Untuk keluar dari klien psql, masukkan \q.

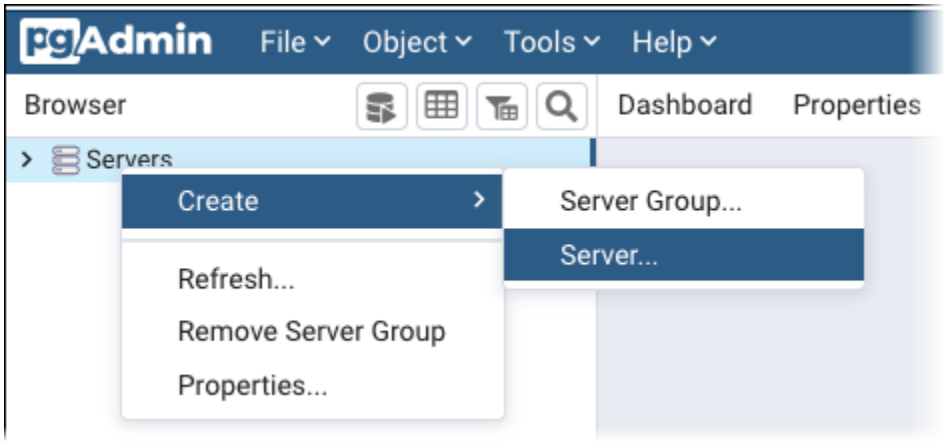
Untuk informasi selengkapnya tentang penggunaan klien psql untuk menanyakan klaster Aurora PostgreSQL, lihat dokumentasi PostgreSQL.

Menggunakan pgAdmin untuk terhubung ke klaster DB

Anda dapat menggunakan klien pgAdmin untuk mengakses data Anda dalam dialek PostgreSQL asli.

Untuk terhubung ke klaster dengan klien pgAdmin

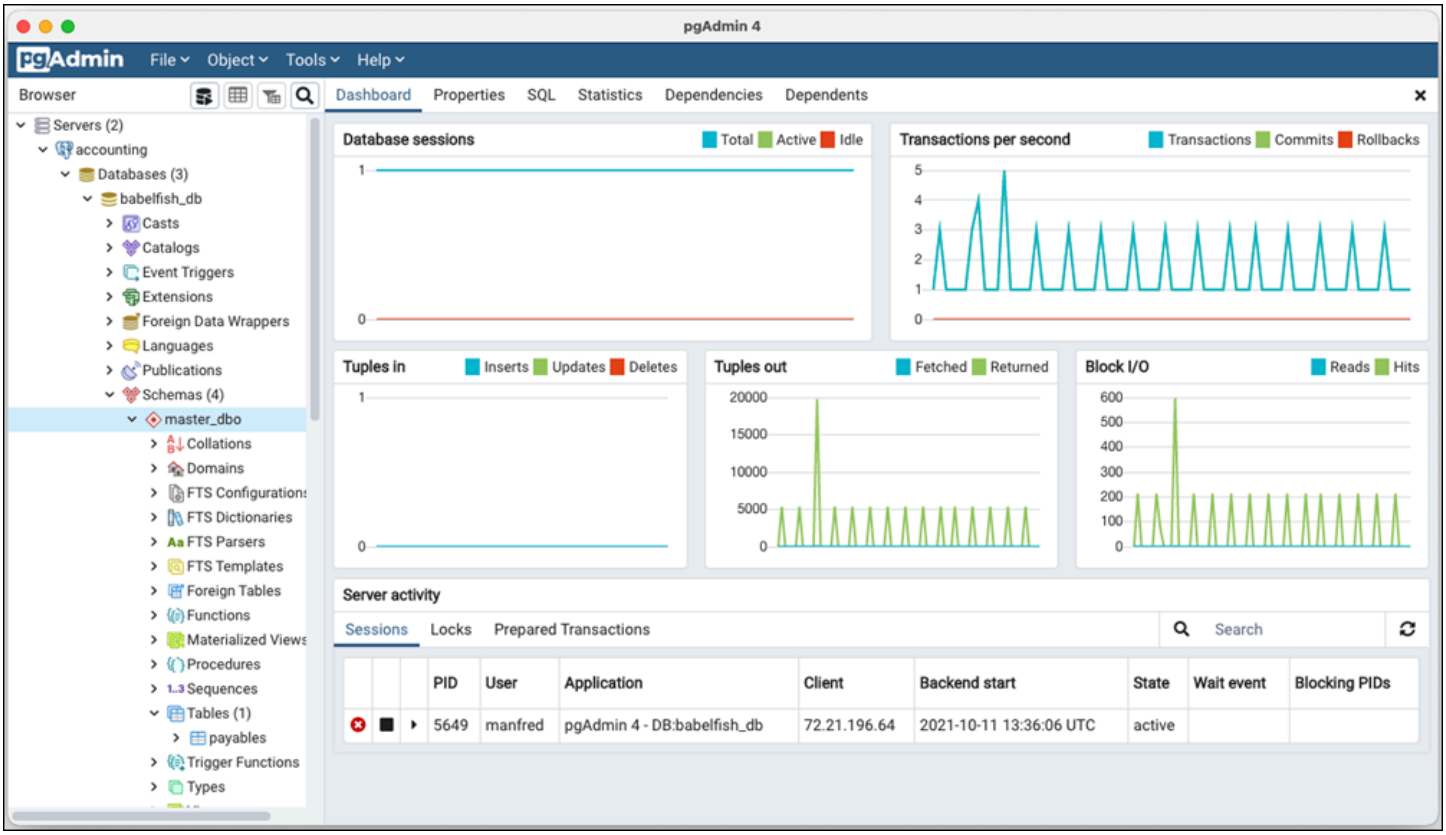
1. Unduh dan instal klien pgadmin dari situs web pgAdmin.
2. Buka klien dan autentikasi dengan pgAdmin.
3. Buka menu konteks (klik kanan) untuk Server, lalu pilih Buat, Server.




4. Masukkan informasi di kotak dialog Buat - Server.

   Pada tab Koneksi, tambahkan alamat klaster Aurora PostgreSQL untuk Host dan nomor port PostgreSQL (secara default, 5432) untuk Port. Berikan detail autentikasi, dan pilih Simpan.

Setelah terhubung, Anda dapat menggunakan fungsionalitas pgAdmin untuk memantau dan mengelola klaster Aurora PostgreSQL Anda di port PostgreSQL.




Untuk mempelajari lebih lanjut, lihat halaman web pgAdmin.

# Menggunakan Babelfish

Pada bagian berikut, Anda dapat menemukan informasi penggunaan Babelfish, termasuk beberapa perbedaan antara cara menggunakan Babelfish dan SQL Server serta antara basis data Babelfish dan PostgreSQL.

Topik


- Mendapatkan informasi dari katalog sistem Babelfish
- Mengelola izin dan kontrol akses di Babelfish untuk Aurora PostgreSQL
- Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server
- Menggunakan fitur Babelfish dengan implementasi terbatas
- Meningkatkan performa kueri Babelfish
- Menggunakan ekstensi Aurora PostgreSQL dengan Babelfish
- Babelfish mendukung server tertaut
- Menggunakan Pencarian Teks Lengkap di Babelfish
- Babelfish mendukung tipe data Geospasial
- Memahami partisi di Babelfish


## Mendapatkan informasi dari katalog sistem Babelfish

Anda dapat memperoleh informasi tentang objek basis data yang disimpan di klaster Babelfish Anda dengan menjalankan kueri banyak tampilan sistem yang sama seperti yang digunakan dalam SQL Server. Setiap rilis baru Babelfish menambahkan dukungan untuk lebih banyak tampilan sistem. Untuk daftar tampilan yang tersedia saat ini, lihat tabel SQL Server system catalog views.

Tampilan sistem ini memberikan informasi dari katalog sistem (`sys.schemas`). Dalam kasus Babelfish, tampilan ini berisi skema sistem SQL Server dan PostgreSQL. Untuk menjalankan kueri seputar informasi katalog sistem pada Babelfish, Anda dapat menggunakan port TDS atau port PostgreSQL, seperti yang ditunjukkan pada contoh berikut.

- Jalankan kueri pada port T-SQL menggunakan **`sqlcmd`** atau klien SQL Server lainnya.

```
1> SELECT * FROM sys.schemas
2> GO
```

Kueri ini menampilkan skema sistem SQL Server dan Aurora PostgreSQL, seperti yang ditunjukkan pada yang berikut ini.

```
name
------------------------------------------------------------
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- Jalankan kueri pada port PostgreSQL menggunakan **`psql`** atau **`pgAdmin`**. Contoh ini menggunakan metacommand (`\dn`) skema daftar `psql`:

```
babelfish_db=> \dn
```

Kueri menampilkan set hasil yang sama seperti yang ditampilkan oleh `sqlcmd` pada port T-SQL.

```
        List of schemas
             Name
------------------------------

 demographic_dbo

 public
 sys
 master_dbo
 tempdb_dbo
...
```

Katalog sistem SQL Server yang tersedia di Babelfish

Dalam tabel berikut, Anda dapat menemukan tampilan SQL Server yang saat ini diimplementasikan di Babelfish. Untuk informasi selengkapnya tentang katalog sistem di SQL Server, lihat System Catalog Views (Transact-SQL) di dokumentasi Microsoft.

| Nama tampilan | Keterangan atau batasan Babelfish (jika ada) |
| --- | --- |
| `sys.all_columns` | Semua kolom di semua tabel dan tampilan |
| `sys.all_objects` | Semua objek di semua skema |
| `sys.all_sql_modules` | Penyatuan `sys.sql_modules` dan `sys.system_sql_modules` |
| `sys.all_views` | Semua tampilan di semua skema |
| `sys.columns` | Semua kolom dalam tabel dan tampilan yang ditentukan pengguna |
| `sys.configurations` | Dukungan Babelfish terbatas pada satu konfigurasi hanya baca. |
| `sys.data_spaces` | Berisi baris untuk setiap ruang data. Ini bisa berupa filegroup, skema partisi, atau filegroup data FILESTREAM. |
| `sys.database_files` | Tampilan per basis data yang berisi satu baris untuk setiap file basis data seperti yang disimpan dalam basis data itu sendiri. |
| `sys.database_mirroring` | Untuk info selengkapnya, lihat sys.database_mirroring di dokumentasi Microsoft Transact-SQL. |
| `sys.database_principals` | Untuk info selengkapnya, lihat sys.database_principals di dokumentasi Microsoft Transact-SQL. |
| `sys.database_role_members` | Untuk info selengkapnya, lihat sys.database_role_members di dokumentasi Microsoft Transact-SQL. |
| `sys.databases` | Semua basis data di semua skema |

| Nama tampilan | Keterangan atau batasan Babelfish (jika ada) |
| --- | --- |
| `sys.dm_exec_connections` | Untuk info selengkapnya, lihat sys.dm_exec_connections di dokumentasi Microsoft Transact-SQL. |
| `sys.dm_exec_sessions` | Untuk info selengkapnya, lihat sys.dm_exec_sessions di dokumentasi Microsoft Transact-SQL. |
| `sys.dm_hadr_database_replica_states` | Untuk info selengkapnya, lihat sys.dm_hadr_database_replica_states di dokumentasi Microsoft Transact-SQL. |
| `sys.dm_os_host_info` | Untuk info selengkapnya, lihat sys.dm_os_host_info di dokumentasi Microsoft Transact-SQL. |
| `sys.endpoints` | Untuk info selengkapnya, lihat sys.endpoints di dokumentasi Microsoft Transact-SQL. |
| `sys.indexes` | Untuk info selengkapnya, lihat sys.indexes di dokumentasi Microsoft Transact-SQL. |
| `sys.languages` | Untuk info selengkapnya, lihat sys.languages di dokumentasi Microsoft Transact-SQL. |
| `sys.schemas` | Semua skema |
| `sys.server_principals` | Semua kredensial masuk dan peran |
| `sys.sql_modules` | Untuk info selengkapnya, lihat sys.sql_modules di dokumentasi Microsoft Transact-SQL. |
| `sys.sysconfigures` | Dukungan Babelfish terbatas pada satu konfigurasi hanya baca. |
| `sys.syscurconfigs` | Dukungan Babelfish terbatas pada satu konfigurasi hanya baca. |

| Nama tampilan | Keterangan atau batasan Babelfish (jika ada) |
| --- | --- |
| `sys.sysprocesses` | Untuk info selengkapnya, lihat sys.sysprocesses di dokumentasi Microsoft Transact-SQL. |
| `sys.system_sql_modules` | Untuk info selengkapnya, lihat sys.system_sql_modules di dokumentasi Microsoft Transact-SQL. |
| `sys.table_types` | Untuk info selengkapnya, lihat sys.table_types di dokumentasi Microsoft Transact-SQL. |
| `sys.tables` | Semua tabel dalam skema |
| `sys.xml_schema_collections` | Untuk info selengkapnya, lihat sys.xml_schema_collections di dokumentasi Microsoft Transact-SQL. |

PostgreSQL mengimplementasikan katalog sistem yang mirip dengan tampilan katalog objek SQL Server. Untuk daftar lengkap katalog sistem, lihat System Catalogs di dokumentasi PostgreSQL.

Ekspor DDL didukung oleh Babelfish

Dari versi Babelfish 2.4.0 dan 3.1.0, Babelfish mendukung ekspor DDL menggunakan berbagai alat. Misalnya, Anda dapat menggunakan fungsi ini dari SQL Server Management Studio (SSMS) untuk menghasilkan skrip definisi data untuk berbagai objek dalam basis data Babelfish for Aurora PostgreSQL. Anda kemudian dapat menggunakan perintah DDL yang dihasilkan dalam skrip ini untuk membuat objek yang sama di Babelfish for Aurora PostgreSQL lain atau basis data SQL Server.

Babelfish mendukung ekspor DDL untuk objek berikut dalam versi yang ditentukan.

| Daftar objek | 2.4.0 | 3.1.0 |
| --- | --- | --- |
| Tabel pengguna | Ya | Ya |
| Kunci primer | Ya | Ya |

| Daftar objek | 2.4.0 | 3.1.0 |
|---|---|---|
| Kunci asing | Ya | Ya |
| Batasan unik | Ya | Ya |
| Indeks | Ya | Ya |
| Batasan pemeriksaan | Ya | Ya |
| Tampilan | Ya | Ya |
| Prosedur tersimpan | Ya | Ya |
| Fungsi yang ditetapkan pengguna | Ya | Ya |
| Fungsi bernilai tabel | Ya | Ya |
| Pemicu | Ya | Ya |
| Tipe Data yang Ditetapkan Pengguna | Tidak | Tidak |
| Tipe Tabel yang Ditetapkan Pengguna | Tidak | Tidak |
| Pengguna | Tidak | Tidak |
| Kredensial Masuk | Tidak | Tidak |
| Urutan | Tidak | Tidak |
| Peran | Tidak | Tidak |

Keterbatasan dengan yang diekspor DDLs

- Gunakan escape hatch sebelum membuat ulang objek dengan yang diekspor DDLs — Babelfish tidak mendukung semua perintah dalam skrip DDL yang diekspor. Gunakan escape hatch untuk menghindari kesalahan yang disebabkan saat membuat ulang objek dari perintah DDL di Babelfish. Untuk informasi selengkapnya tentang escape hatch, lihat Mengelola penanganan kesalahan Babelfish dengan escape hatch

- Objek yang berisi batasan CHECK dengan klausa COLLATE eksplisit – Skrip dengan objek ini yang dihasilkan dari basis data SQL Server memiliki kolasi yang berbeda tetapi setara seperti dalam basis data Babelfish. Misalnya, beberapa kolasi, seperti sql_latin1_general_cp1_cs_as, sql_latin1_general_cp1251_cs_as, dan latin1_general_cs_as dihasilkan seperti latin1_general_cs_as, yang merupakan kolasi Windows terdekat.

## Mengelola izin dan kontrol akses di Babelfish untuk Aurora PostgreSQL

Di Babelfish for Aurora PostgreSQL, Anda dapat mengelola izin dan kontrol akses untuk database, skema, dan objek. Tabel berikut akan menguraikan perintah SQL tertentu untuk memberikan izin di Babelfish untuk mencapai berbagai skenario kontrol akses. Ini akan mencakup kasus penggunaan yang didukung yang dapat diimplementasikan serta solusi untuk kasus yang saat ini tidak didukung. Ini akan memungkinkan Anda mengonfigurasi izin yang sesuai untuk memenuhi persyaratan keamanan dan kepatuhan Anda saat bekerja dengan database Babelfish.

Kasus penggunaan yang didukung

Tabel berikut menjelaskan kasus penggunaan yang didukung di Babelfish. Untuk setiap kasus penggunaan, tabel menunjukkan tindakan yang diperlukan untuk mencapainya dan contoh perintah SQL.

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish |
|---|---|---|---|---|
| Izinkan login untuk melakukan SELECTs/DMLs/DDLs dalam database apa pun | Tambahkan login ke peran server sysadmin | UBAH PERAN SERVER sysadmin TAMBAHKAN ANGGOTA `login` | Tidak ada | Semua versi |
| Izinkan login untuk melakukan SELECTsDMLs//DDLs dalam database | Jadikan login sebagai pemilik database | UBAH OTORISASI PADA DATABASE:: KE `database login` | Database hanya dapat memiliki satu pemilik. | Versi 3.4 dan lebih tinggi |

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish |
|---|---|---|---|---|
| Izinkan pengguna database untuk melakukannya SELECTs/DMLs pada skema | Berikan izin kepada pengguna database pada skema | HIBAH SELECT/EXECUTE/INSERT/UPDATE/DELETE PADA SKEMA:: `schema` UNTUK `user` | Tidak ada | Versi 3.6 dan lebih tinggi, 4.2 dan lebih tinggi |
| Izinkan pengguna database untuk melakukannya SELECTs/DMLs pada skema | Jadikan pemilik basis data pengguna skema pada waktu pembuatan skema | BUAT OTORISASI SKEMA `schema` `user` | Mengubah kepemilikan skema setelah pembuatan saat ini tidak didukung. | Versi 1.2 dan lebih tinggi |
| Izinkan pengguna database untuk melakukan SELECTs/DMLs pada objek | Memberikan izin kepada pengguna database pada objek | HIBAH SELECT/EXECUTE/INSERT/UPDATE/DELETE PADA OBJEK:: `object` UNTUK `user` | Tidak ada | Semua versi |
| Izinkan pengguna database untuk melakukan SELECTsDMLs//DDLs dalam database termasuk menjatuhkan database | Tambahkan pengguna ke peran database tetap db_owner | UBAH PERAN db_owner TAMBAHKAN ANGGOTA `user` | Hanya pengguna yang dapat ditambahkan peran database tetap db_owner. Menambahkan peran ke peran db_owner belum didukung. | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish |
|---|---|---|---|---|
| Izinkan pengguna atau anggota peran database kustom untuk melakukan hanya SELECTs dalam database | Tambahkan pengguna atau peran ke peran database tetap db_datareader | UBAH PERAN db_datareader TAMBAHKAN ANGGOTA/userr | Tidak ada | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |
| Izinkan pengguna atau anggota peran database kustom untuk melakukan hanya DMLs dalam database | Tambahkan pengguna atau peran ke peran database tetap db_datawriter | UBAH PERAN db_datawriter TAMBAHKAN ANGGOTA/userr | Tidak ada | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |
| Izinkan pengguna atau anggota peran database kustom untuk melakukan hanya DDLs dalam database | Tambahkan pengguna atau peran ke peran database tetap db_accessadmin | UBAH PERAN db_accessadmin TAMBAHKAN ANGGOTA/userr | Tidak ada | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish |
|---|---|---|---|---|
| Izinkan pengguna atau anggota peran database kustom untuk hanya CREATE/ALTER/DROP custom roles, GRANT/REVOKE permissions on objects in a database and/or MEMBUAT SKEMA dalam database | Tambahkan pengguna atau peran ke peran database tetap db_securityadmin | UBAH PERAN db_securityadmin TAMBAHKAN ANGGOTA/userr | Tidak ada | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |
| Izinkan pengguna atau anggota peran database kustom untuk hanya CREATE/ALTER/DROP any user, grant and revoke database access and map user accounts to logins and/or MEMBUAT SKEMA dalam database | Tambahkan pengguna atau peran ke peran database tetap db_accessadmin | UBAH PERAN db_accessadmin TAMBAHKAN ANGGOTA/userr | Tidak ada | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish |
|---|---|---|---|---|
| Izinkan login hanya ke database CREATE/DROP/ALTER apa pun | Tambahkan login ke peran server tetap dbcreator | UBAH PERAN SERVER dbcreator TAMBAHKAN ANGGOTA `login` | Hanya database tersebut yang dapat diubah di mana login dbcreator memiliki akses. | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |
| Izinkan login hanya untuk login CREATE/ALTER/DROP apa pun | Tambahkan login ke peran server tetap securityadmin | UBAH KEAMANAN PERAN SERVERADMIN TAMBAHKAN ANGGOTA `login` | Tidak ada | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |

Kasus penggunaan yang tidak didukung dengan solusi

Tabel berikut menjelaskan kasus penggunaan yang tidak didukung di Babelfish, tetapi dapat dicapai dengan menggunakan solusi.

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish untuk solusi |
|---|---|---|---|---|
| Izinkan pengguna untuk melakukan SELECTs/DMLs opsi object/schema bersama dengan untuk MEMBERIKAN | BERIKAN izin kepada semua pengguna lain secara langsung | HIBAH SELECT/EXECUTE/INSERT/UPDATE/DELETE ON OBJECT/SCHEMA::`object/sc` `user` | HIBAH... DENGAN OPSI GRANT saat ini tidak didukung. | Versi 3.6 dan lebih tinggi, 4.2 dan lebih tinggi |

| Kasus penggunaan | Tindakan | Perintah SQL | Komentar | Kompatibilitas versi Babelfish untuk solusi |
|---|---|---|---|---|
| izin ini kepada pengguna lain | | | | |
| Izinkan pengguna database untuk melakukan SELECTsDM Ls//DDLs dalam database termasuk menjatuhkan database | Tambahkan anggota peran ke peran database tetap db_owner | UBAH PERAN db_owner TAMBAHKAN ANGGOTA `user` | Menambahk an peran ke peran db_owner saat ini tidak didukung. | Versi 4.5 dan lebih tinggi, 5.1 dan lebih tinggi |

Kasus penggunaan yang tidak didukung

Tabel berikut menjelaskan kasus penggunaan yang tidak didukung di Babelfish.

| Kasus penggunaan | Komentar |
|---|---|
| Tolak pengguna atau anggota peran database kustom untuk dilakukan SELECTs dalam database | peran database tetap db_denydatareader belum didukung |
| Tolak pengguna atau anggota peran database kustom untuk dilakukan DMLs dalam database | peran database tetap db_denydatawriter saat ini tidak didukung. |
| Izinkan login untuk hanya MEMBUNUH koneksi database apa pun | Peran server tetap processadmin saat ini tidak didukung. |
| Izinkan login untuk hanya menambah atau menghapus server yang ditautkan | peran server tetap setupadmin saat ini tidak didukung. |

### Mengatasi perbedaan kepemilikan objek setelah peningkatan

Babelfish versi 4.6 dan yang lebih baru, dan 5.2 dan yang lebih baru menyertakan perubahan pada penanganan kepemilikan objek melalui titik akhir TDS. Saat Anda membuat objek baru melalui titik akhir TDS, objek ini sekarang dimiliki oleh pemilik skema, bukan pengguna saat ini. Perubahan kepemilikan ini dapat memengaruhi perilaku izin untuk objek baru dibandingkan dengan objek yang ada saat Anda memutakhirkan dari versi yang lebih awal dari 4.6 atau 5.2.

Untuk mengatasi perbedaan kepemilikan ini, Babelfish menyediakan fungsinya. `sys.generate_alter_ownership_statements()` Fungsi ini menghasilkan pernyataan SQL yang menyelaraskan kepemilikan objek dengan kepemilikan skema.

Waspadai batasan berikut saat menangani kepemilikan objek:

- Pengguna dengan izin CREATE yang diberikan melalui titik akhir PostgreSQL tidak dapat membuat objek melalui titik akhir TDS dalam skema tersebut.
- Memodifikasi izin pada objek T-SQL melalui titik akhir PostgreSQL tidak disarankan dan dapat menyebabkan perilaku T-SQL yang salah.
- Izin akses mungkin berbeda antara objek lama dan baru karena ketidakcocokan kepemilikannya. Misalnya, pertimbangkan skema yang dimiliki oleh `sch_own` yang mencakup objek yang dimiliki oleh`dbo`. Dalam hal ini, objek yang dimiliki oleh `dbo` yang dibuat sebelum upgrade mungkin memiliki izin akses yang berbeda dibandingkan dengan objek yang dimiliki oleh `sch_own` yang dibuat setelah upgrade. Ini dapat mempengaruhi operasi seperti SELECT dan INSERT.

Jika cluster DB Anda menyertakan objek yang dibuat dalam versi Babelfish lebih awal dari 4.6 atau 5.2, pertimbangkan untuk menyelaraskan kepemilikannya.

Untuk mengatasi perbedaan kepemilikan objek

1. Connect ke `babelfish_db` database di cluster DB Anda menggunakan titik akhir PostgreSQL.
2. Jalankan perintah berikut:

   ```
   SELECT * from sys.generate_alter_ownership_statements();
   ```

   Ini menghasilkan daftar pernyataan SQL dimaksudkan untuk membakukan kepemilikan di antara objek.
3. Jalankan pernyataan yang dihasilkan pada lingkungan pengujian terlebih dahulu untuk memvalidasi efeknya sebelum diterapkan pada lingkungan produksi Anda.

Kami menyarankan Anda mengeksekusi pernyataan ini untuk mencapai model kepemilikan objek yang konsisten di seluruh database Anda.

## Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server

Babelfish adalah fitur Aurora PostgreSQL yang berkembang, dengan fungsionalitas baru yang ditambahkan di setiap rilis sejak penawaran awal di Aurora PostgreSQL 13.4. Fitur ini dirancang untuk menyediakan semantik T-SQL di atas PostgreSQL melalui dialek T-SQL menggunakan port TDS. Setiap versi baru Babelfish menambahkan fitur dan fungsi yang lebih selaras dengan fungsionalitas dan perilaku T-SQL, seperti yang ditunjukkan pada tabel Fungsionalitas yang didukung di Babelfish berdasarkan versi. Untuk hasil terbaik saat bekerja dengan Babelfish, sebaiknya pahami perbedaan yang ada saat ini antara T-SQL yang didukung oleh SQL Server dan Babelfish untuk versi terbaru. Untuk mempelajari selengkapnya, lihat Perbedaan T-SQL di Babelfish.

Selain perbedaan antara T-SQL yang didukung oleh Babelfish dan SQL Server, Anda mungkin juga perlu mempertimbangkan masalah interoperabilitas antara Babelfish dan PostgreSQL dalam konteks klaster DB Aurora PostgreSQL. Seperti disebutkan sebelumnya, Babelfish mendukung semantik T-SQL di atas PostgreSQL melalui dialek T-SQL menggunakan port TDS. Pada saat yang sama, basis data Babelfish juga dapat diakses melalui port PostgreSQL standar dengan pernyataan SQL PostgreSQL. Jika Anda mempertimbangkan untuk menggunakan fungsionalitas PostgreSQL dan Babelfish dalam deployment produksi, Anda perlu mengetahui potensi masalah interoperabilitas antara nama skema, pengidentifikasi, izin, semantik transaksional, beberapa set hasil, kolasi default, dan sebagainya. Secara sederhana, ketika pernyataan PostgreSQL atau akses PostgreSQL terjadi dalam konteks Babelfish, interferensi antara PostgreSQL dan Babelfish dapat terjadi dan berpotensi memengaruhi sintaks, semantik, dan kompatibilitas ketika versi baru Babelfish dirilis. Untuk informasi lengkap dan panduan tentang semua pertimbangan, lihat Guidance on Babelfish Interoperability dalam dokumentasi Babelfish for PostgreSQL.

> **Note**
>
> Sebelum menggunakan fungsionalitas asli PostgreSQL dan fungsionalitas Babelfish dalam konteks aplikasi yang sama, kami sangat menyarankan Anda mempertimbangkan masalah yang dibahas dalam Guidance on Babelfish Interoperability dalam dokumentasi Babelfish for PostgreSQL. Masalah interoperabilitas ini (Aurora PostgreSQL dan Babelfish) hanya relevan jika Anda berencana untuk menggunakan instans basis data PostgreSQL dalam konteks aplikasi yang sama dengan Babelfish.

Topik


- Babelfish membuang dan memulihkan
- Perbedaan T-SQL di Babelfish
- Tingkat isolasi transaksi di Babelfish


## Babelfish membuang dan memulihkan

Dimulai dengan versi 4.0.0 dan 3.4.0, pengguna Babelfish sekarang dapat memanfaatkan utilitas dump dan restore untuk mencadangkan dan memulihkan database mereka. Untuk informasi lebih lanjut, lihat Babelfish dump and restore. Fitur ini dibangun di atas utilitas dump dan restore PostgreSQL. Untuk informasi selengkapnya, lihat pg_dump dan lihat pg_restore. Untuk menggunakan fitur ini secara efektif di Babelfish, Anda perlu menggunakan alat berbasis PostgreSQL yang secara khusus disesuaikan untuk Babelfish. Fitur backup dan restore untuk Babelfish berbeda secara signifikan dari SQL Server. Untuk informasi selengkapnya tentang perbedaan ini, lihat Membuang dan memulihkan perbedaan fungsionalitas: Babelfish dan SQL Server. Babelfish untuk Aurora PostgreSQL menyediakan kemampuan tambahan untuk mencadangkan dan memulihkan cluster Amazon Aurora PostgreSQL DB. Lihat informasi yang lebih lengkap di Mencadangkan dan memulihkan klaster DB Amazon Aurora.

## Perbedaan T-SQL di Babelfish

Berikut cara Anda dapat menemukan tabel fungsionalitas T-SQL seperti yang didukung dalam rilis Babelfish saat ini dengan beberapa catatan tentang perbedaan perilaku dari SQL Server.

Untuk informasi selengkapnya tentang dukungan di berbagai versi, lihat Fungsionalitas yang didukung di Babelfish berdasarkan versi. Untuk informasi tentang fitur yang saat ini tidak didukung, lihat Fungsionalitas yang tidak didukung di Babelfish.

Babelfish tersedia dengan Aurora Edisi Kompatibel PostgreSQL. Untuk informasi lebih lanjut tentang rilis Babelfish, lihat Catatan Rilis untuk Aurora PostgreSQL.

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| \ (karakter kelanjutan baris) | Karakter kelanjutan baris (garis miring terbalik sebelum baris baru) untuk karakter dan string heksadesimal saat ini tidak didukung. Untuk string karakter, backslash-newline ditafsirkan sebagai karakter dalam string. Untuk string heksadesimal, garis miring terbalik menghasilkan kesalahan sintaks. |

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| @@version | Format nilai yang dikembalikan oleh `@@version` sedikit berbeda dari nilai yang dikembalikan oleh SQL Server. Kode Anda mungkin tidak berfungsi dengan benar jika tergantung pada pemformatan `@@version` . |
| Fungsi agregat | Fungsi agregat didukung sebagian (AVG, COUNT, COUNT_BIG , GROUPING, MAX, MIN, STRING_AGG, dan SUM didukung) . Untuk daftar fungsi agregat yang tidak didukung, lihat. Fungsi yang tidak didukung |
| ALTER TABLE | Mendukung penambahan atau penjatuhan satu kolom atau batasan saja. |
| ALTER TABLE..ALTER COLUMN | NULL dan NOT NULL saat ini tidak dapat ditentukan. Untuk mengubah nullability kolom, gunakan pernyataan PostgreSQL ALTER TABLE.. {SET\|DROP} NOT NULL. |
| DI ZONA WAKTU | Selama transisi dari Daylight Saving Time (DST) ke Waktu Standar, periode tumpang tindih ditampilkan menggunakan offset Waktu Standar. Untuk memperjelas, pertimbangkan contoh berikut:<br>`SELECT CONVERT(DATETIME2(0), '2022-10-30T02:00:00', 126) AT TIME ZONE 'Central European Standard Time';`<br>`GO;`<br>`Result: 2022-10-30 02:00:00 +01:00` |
| Nama kolom kosong tanpa alias kolom | Utilitas `sqlcmd` dan `psql` menangani kolom dengan nama kosong secara berbeda:<br>• SQL Server `sqlcmd` mengembalikan nama kolom kosong.<br>• PostgreSQL `psql` mengembalikan nama kolom yang dihasilkan. |

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| Fungsi CHECKSUM | Babelfish dan SQL Server menggunakan algoritma hashing yang berbeda untuk fungsi CHECKSUM. Akibatnya, nilai hash yang dihasilkan oleh fungsi CHECKSUM di Babelfish mungkin berbeda dari yang dihasilkan oleh fungsi CHECKSUM di SQL Server. |
| Kolom default | Saat membuat kolom default, nama batasan diabaikan. Untuk menjatuhkan kolom default, gunakan sintaks berikut: `ALTER TABLE...ALTER COLUMN..DROP DEFAULT...` |
| Constraint_name | Di SQL Server, nama kendala harus unik dalam skema yang menjadi milik tabel. Namun, di Babelfish, ini hanya berlaku untuk kendala KUNCI PRIMER dan UNIK. Jenis kendala lainnya tidak tunduk pada batasan ini. |
| Kekurangan | PostgreSQL tidak mendukung pengaktifan dan penonaktifan batasan individu. Pernyataan itu diabaikan dan peringatan diajukan. |
| Batasan dengan IGNORE_DUP_KEY | Batasan dibuat tanpa properti ini. |
| CREATE, ALTER, DROP SERVER ROLE | ALTER SERVER ROLE hanya didukung untuk `sysadmin`. Semua sintaks lainnya tidak didukung.<br><br>Pengguna T-SQL di Babelfish memiliki pengalaman yang mirip dengan SQL Server untuk konsep login (pengguna utama server), basis data, dan pengguna basis data user (pengguna utama basis data). |
| Klausa CREATE, ALTER LOGIN didukung dengan sintaks terbatas | CREATE LOGIN... Klausa PASSWORD, ...klausa DEFAULT_DATABASE, dan... klausa DEFAULT_LANGUAGE didukung. ALTER LOGIN... Klausa PASSWORD didukung, tetapi ALTER LOGIN... Klausa OLD_PASSWORD tidak didukung. Hanya login yang merupakan anggota sysadmin yang dapat memodifikasi kata sandi. |

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| Kolasi CREATE DATABASE peka huruf besar/kecil | Kolasi peka huruf besar/kecil tidak didukung dengan pernyataan CREATE DATABASE. |
| Kata kunci dan klausa CREATE DATABASE | Opsi kecuali COLLATE dan CONTAINMENT=NONE tidak didukung. Klausa COLLATE diterima dan selalu diatur ke nilai `babelfishpg_tsql.server_collation_name` . |
| CREATE SCHEMA... klausa pendukung | Anda dapat menggunakan perintah CREATE SCHEMA untuk membuat skema kosong. Gunakan perintah tambahan untuk membuat objek skema. |
| Nilai ID basis data berbeda pada Babelfish | Database master dan tempdb tidak akan menjadi database IDs 1 dan 2. |
| Format fungsi tipe tanggal didukung dengan batasan berikut | Meridian karakter tunggal tidak didukung.<br><br>Format "yyy" di server SQL mengembalikan 4 digit untuk tahun di atas 1000, tetapi hanya 3 digit untuk yang lain.<br><br>Format "g" dan "R" tidak didukung<br><br>Terjemahan lokal "Vi-VN" sedikit berbeda. |
| Pengidentifikasi melebihi 63 karakter | PostgreSQL mendukung maksimal 63 karakter untuk pengident ifikasi. Babelfish mengonversi pengenal lebih dari 63 karakter ke nama yang menyertakan hash dari nama aslinya. Misalnya, tabel yang dibuat sebagai “AB (ABC12345678901234 5678901234567890123456789012345678901234567890123456789 01234567890" dapat dikonversi ke" 012345678901234567 8901234567890123456789012345678901234567890". ABC123456789 |

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| Dukungan kolom IDENTITY | Kolom `IDENTITY` didukung untuk tipe data `tinyint`, `smallint`, `int`, `bigint`. `numeric`, dan `decimal`.<br><br>SQL Server mendukung presisi ke 38 tempat untuk tipe data `numeric` dan `decimal` di kolom IDENTITY.<br><br>PostgreSQL mendukung presisi ke 19 tempat untuk tipe data `numeric` dan `decimal` di kolom IDENTITY. |
| Indeks dengan IGNORE_DUP_KEY | Sintaks yang membuat indeks yang menyertakan IGNORE_DUP_KEY membuat indeks seolah-olah properti ini dihilangkan. |
| Indeks dengan lebih dari 32 kolom | Indeks tidak dapat menyertakan lebih dari 32 kolom. Kolom indeks yang disertakan dihitung menuju maksimum di PostgreSQL tetapi tidak di SQL Server. |
| Indeks (berklaster) | Indeks berklaster dibuat seolah-olah NONCLUSTERED ditentukan. |
| Klausa indeks | Klausul berikut diabaikan: FILLFACTOR, ALLOW_PAGE_LOCKS, ALLOW_ROW_LOCKS, PAD_INDEX, STATISTICS_NORECOMPUTE, OPTIMIZE_FOR_SEQUENTIAL_KEY, SORT_IN_TEMPDB, DROP_EXISTING, ONLINE, COMPRESSION_DELAY, MAXDOP, dan DATA_COMPRESSION |
| Dukungan JSON | Urutan pasangan nama-nilai tidak dijamin. Tetapi tipe array tetap tidak terpengaruh. |
| Objek LOGIN | Semua opsi untuk objek LOGIN tidak didukung kecuali PASSWORD, DEFAULT_DATABASE, DEFAULT_LANGUAGE, ENABLE, DISABLE. |
| Fungsi NEWSEQUENTIALID | Diimplementasikan sebagai NEWID; perilaku berurutan tidak dijamin. Saat memanggil `NEWSEQUENTIALID` , PostgreSQL menghasilkan nilai GUID baru. |

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| Klausa OUTPUT didukung dengan batasan berikut | OUTPUT dan OUTPUT INTO tidak didukung dalam kueri DML yang sama. Referensi ke tabel non-target operasi UPDATE atau DELETE dalam klausa OUTPUT tidak didukung. OUTPUT... DELETED *, INSERTED * tidak didukung dalam kueri yang sama. |
| Batas parameter prosedur atau fungsi | Babelfish mendukung maksimum 100 parameter untuk prosedur atau fungsi. |
| ROWGUIDCOL | Klausa ini diabaikan untuk sekarang. Kueri referensi `$GUIDGOL` menyebabkan kesalahan sintaks. |
| Dukungan objek SEQUENCE | Objek SEQUENCE didukung untuk tipe data tinyint, smallint, int, bigint, numerik, dan desimal.<br><br>Aurora PostgreSQL mendukung presisi ke 19 tempat untuk tipe data numerik dan desimal dalam SEQUENCE. |
| Peran tingkat server | Peran tingkat server `sysadmin` didukung. Peran tingkat server lainnya (selain `sysadmin`) tidak didukung. |
| Peran tingkat basis data selain `db_owner` | Peran `db_owner` tingkat basis data dan tingkat basis data yang ditentukan pengguna didukung. Peran tingkat basis data lainnya (selain db_owner) tidak didukung. |
| Kata kunci `SPARSE` SQL | Kata kunci SPARSE diterima dan diabaikan. |
| Klausa kata kunci SQL `ON filegroup` | Klausa ini diabaikan untuk sekarang. |
| Kata kunci SQL `CLUSTERED` dan `NONCLUSTERED` untuk indeks dan batasan | Babelfish menerima dan mengabaikan kata kunci `CLUSTERED` dan `NONCLUSTERED` . |
| `sysdatabases.cmptlevel` | `sysdatabases.cmptlevel` selalu diatur ke 120. |

| Fungsionalitas atau sintaks | Deskripsi perilaku atau perbedaan |
| --- | --- |
| tempdb tidak diinisialisasi ulang saat restart | Objek permanen (seperti tabel dan prosedur) yang dibuat di tempdb tidak dihapus saat basis data dimulai ulang. |
| Kumpulan file TEXTIMAGE_ON | Babelfish mengabaikan klausa `TEXTIMAGE_ON` *`filegroup`* . |
| Ketepatan waktu | Babelfish mendukung presisi 6 digit untuk detik pecahan. Tidak ada efek samping yang diantisipasi dengan perilaku ini. |
| Tingkat isolasi transaksi | READUNCOMMITTED diperlakukan sama dengan READCOMMITTED. |
| Kolom komputasi virtual (nonpersisten) | Kolom komputasi virtual dibuat sebagai persisten. |
| Tanpa klausa SCHEMABINDING | Klausa ini tidak didukung dalam fungsi, prosedur, pemicu, atau tampilan. Objek dibuat, tetapi seolah-olah WITH SCHEMABINDING ditentukan. |

Tingkat isolasi transaksi di Babelfish

Babelfish mendukung tingkat isolasi transaksi`READ UNCOMMITTED`, `READ COMMITTED` dan. `SNAPSHOT` Mulai dari versi Babelfish 3.4 tingkat isolasi tambahan `REPEATABLE READ` dan `SERIALIZABLE` didukung. Semua tingkat isolasi di Babelfish didukung dengan perilaku tingkat isolasi yang sesuai di Postgre. SQL SQL Server dan Babelfish menggunakan mekanisme dasar yang berbeda untuk menerapkan tingkat isolasi transaksi (memblokir untuk akses bersamaan, kunci yang dipegang oleh transaksi, penanganan kesalahan, dll). Dan, ada beberapa perbedaan halus dalam cara akses bersamaan dapat bekerja untuk beban kerja yang berbeda. Untuk informasi selengkapnya tentang SQL perilaku Postgre ini, lihat Isolasi Transaksi.

Topik

- Ikhtisar tingkat isolasi transaksi
- Menyiapkan tingkat isolasi transaksi
- Mengaktifkan atau menonaktifkan tingkat isolasi transaksi
- Membandingkan tingkat isolasi Babelfish dan SQL Server

Ikhtisar tingkat isolasi transaksi

Tingkat isolasi transaksi SQL Server asli didasarkan pada penguncian pesimis di mana hanya satu salinan data yang ada dan kueri harus mengunci sumber daya seperti baris sebelum mengaksesnya. Kemudian, variasi tingkat `READ COMMITTED` isolasi diperkenalkan. Ini memungkinkan penggunaan versi baris untuk memberikan konkurensi yang lebih baik antara pembaca dan penulis menggunakan akses non-pemblokiran. Selain itu, tingkat isolasi baru yang `SNAPSHOT` disebut tersedia. Ini juga menggunakan versi baris untuk memberikan konkurensi yang lebih baik daripada tingkat `REPEATABLE READ` isolasi dengan menghindari kunci bersama pada data baca yang disimpan hingga akhir transaksi.

Tidak seperti SQL Server, semua tingkat isolasi transaksi di Babelfish didasarkan pada Optimistic Locking (). MVCC Setiap transaksi melihat snapshot data baik di awal pernyataan (`READ COMMITTED`) atau di awal transaksi (`REPEATABLE READ`,`SERIALIZABLE`), terlepas dari keadaan saat ini dari data yang mendasarinya. Oleh karena itu, perilaku eksekusi transaksi bersamaan di Babelfish mungkin berbeda dari Server. SQL

Misalnya, pertimbangkan transaksi dengan tingkat isolasi `SERIALIZABLE` yang awalnya diblokir di SQL Server tetapi berhasil kemudian. Ini mungkin berakhir gagal di Babelfish karena konflik serialisasi dengan transaksi bersamaan yang membaca atau memperbarui baris yang sama.

Mungkin juga ada kasus di mana mengeksekusi beberapa transaksi bersamaan menghasilkan hasil akhir yang berbeda di Babelfish dibandingkan dengan Server. SQL Aplikasi yang menggunakan tingkat isolasi, harus diuji secara menyeluruh untuk skenario konkurensi.

| Tingkat isolasi di SQL Server | Tingkat isolasi babelfish | Tingkat isolasi postgre SQL | Komentar |
|---|---|---|---|
| `READ UNCOMMITTED` | `READ UNCOMMITTED` | `READ UNCOMMITTED` | `READ UNCOMMITTED` sama seperti `READ COMMITTED` di Babelfish atau Postgre SQL |
| `READ COMMITTED` | `READ COMMITTED` | `READ COMMITTED` | SQLServer `READ COMMITTED` berbasis penguncian pesimis, Babelfish `READ COMMITTED` berbasis snapshot (). MVCC |
| `READ COMMITTED SNAPSHOT` | `READ COMMITTED` | `READ COMMITTED` | Keduanya berbasis snapshot (MVCC) tetapi tidak persis sama. |
| `SNAPSHOT` | `SNAPSHOT` | `REPEATABLE READ` | Persis sama. |
| `REPEATABLE READ` | `REPEATABLE READ` | `REPEATABLE READ` | SQLServer `REPEATABLE READ` berbasis penguncian pesimis, Babelfish `REPEATABLE READ` berbasis snapshot (). MVCC |
| `SERIALIZABLE` | `SERIALIZABLE` | `SERIALIZABLE` | SQLServer `SERIALIZABLE` |

| Tingkat isolasi di SQL Server | Tingkat isolasi babelfish | Tingkat isolasi postgre SQL | Komentar |
|---|---|---|---|
| | | | adalah isolasi pesimis, Babelfish `SERIALIZABLE` berbasis snapshot (). MVCC |

> **Note**
>
> Petunjuk tabel saat ini tidak didukung dan perilakunya dikendalikan dengan menggunakan palka pelarian Babelfish yang telah ditentukan sebelumnya. `escape_hatch_table_hints`

Menyiapkan tingkat isolasi transaksi

Gunakan perintah berikut untuk mengatur tingkat isolasi transaksi:

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

Mengaktifkan atau menonaktifkan tingkat isolasi transaksi

Tingkat isolasi transaksi `REPEATABLE READ` dan dinonaktifkan `SERIALIZABLE` secara default di Babelfish dan Anda harus mengaktifkannya secara eksplisit dengan menyetel `babelfishpg_tsql.isolation_level_serializable` atau `babelfishpg_tsql.isolation_level_repeatable_read` escape hatch untuk digunakan. `pg_isolation` `sp_babelfish_configure` Untuk informasi selengkapnya, lihat Mengelola penanganan kesalahan Babelfish dengan escape hatch.

Di bawah ini adalah contoh untuk mengaktifkan atau menonaktifkan penggunaan `REPEATABLE READ` dan `SERIALIZABLE` dalam sesi saat ini dengan mengatur lubang keluar masing-masing. Secara opsional sertakan `server` parameter untuk mengatur escape hatch untuk sesi saat ini serta untuk semua sesi baru berikutnya.

Untuk mengaktifkan penggunaan `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` dalam sesi saat ini saja.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

Untuk mengaktifkan penggunaan `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` dalam sesi saat ini dan semua sesi baru yang diakibatkannya.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

Untuk menonaktifkan penggunaan `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` dalam sesi saat ini dan konsekuen sesi baru.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

Untuk mengaktifkan penggunaan `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` dalam sesi saat ini saja.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

Untuk mengaktifkan penggunaan `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` dalam sesi saat ini dan semua sesi baru yang diakibatkannya.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

Untuk menonaktifkan penggunaan `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` dalam sesi saat ini dan konsekuen sesi baru.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

Membandingkan tingkat isolasi Babelfish dan SQL Server

Di bawah ini adalah beberapa contoh tentang nuansa bagaimana SQL Server dan Babelfish menerapkan tingkat ANSI isolasi.

> **Note**
>
> - Tingkat isolasi `REPEATABLE READ` dan `SNAPSHOT` sama di Babelfish.
> - Tingkat isolasi `READ UNCOMMITTED` dan `READ COMMITTED` sama di Babelfish.

Contoh berikut menunjukkan cara membuat tabel dasar untuk semua contoh yang disebutkan di bawah ini:

```
CREATE TABLE employee (
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

Topik

- Babelfish READ UNCOMMITTED dibandingkan dengan tingkat isolasi SQL Server READ UNCOMMITTED
- Babelfish READ COMMITTED dibandingkan dengan tingkat isolasi SQL Server READ COMMITTED
- Babelfish READ COMMITTED dibandingkan dengan tingkat isolasi SQL Server READ COMMITTED SNAPSHOT

- Babelfish REPEATABLE READ dibandingkan dengan tingkat isolasi SQL Server REPEATABLE READ
- Babelfish SERIALIZABLE dibandingkan dengan tingkat isolasi SQL Server SERIALIZABLE

### Babelfish **READ UNCOMMITTED** dibandingkan dengan tingkat isolasi SQL Server **READ UNCOMMITTED**

Tabel berikut memberikan rincian tentang pembacaan kotor ketika transaksi bersamaan dieksekusi. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `READ UNCOMMITTED` isolasi di SQL Server dibandingkan dengan implementasi Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **READ UNCOMMITTED** | Ikan babelfish **READ UNCOMMITTED** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;` | `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;` | Tidak ada | Tidak ada |
| Idle pada transaksi | `UPDATE employee SET age=0;` | Pembaruan berhasil. | Pembaruan berhasil. |
| Idle pada transaksi | `INSERT INTO employee VALUES (4, 'D', 40);` | Masukkan berhasil. | Masukkan berhasil. |
| `SELECT * FROM employee;` | Idle pada transaksi | Transaksi 1 dapat melihat perubahan tanpa komitmen dari transaksi 2. | Sama seperti `READ COMMITTED` di Babelfish. Perubahan tanpa komitmen dari transaksi 2 tidak terlihat oleh transaksi 1. |
| Idle pada transaksi | `COMMIT` | Tidak ada | Tidak ada |

| Transaksi 1 | Transaksi 2 | SQLServer **READ UNCOMMITTED** | Ikan babelfish **READ UNCOMMITTED** |
|---|---|---|---|
| `SELECT * FROM employee;` | Idle pada transaksi | Melihat perubahan yang dilakukan oleh transaksi 2. | Melihat perubahan yang dilakukan oleh transaksi 2. |

Babelfish **`READ COMMITTED`** dibandingkan dengan tingkat isolasi SQL Server **`READ COMMITTED`**

Tabel berikut memberikan rincian tentang perilaku pemblokiran baca-tulis ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `READ COMMITTED` isolasi di SQL Server dibandingkan dengan implementasi Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **READ COMMITTED** | Ikan babelfish **READ COMMITTED** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;` | `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `UPDATE employee SET age=100 WHERE id = 1;` | Pembaruan berhasil. | Pembaruan berhasil. |
| `UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);` | Idle pada transaksi | Langkah diblokir hingga transaksi 2 dilakukan. | Transaksi 2 perubahan belum terlihat. Update baris dengan id = 3. |
| Idle pada transaksi | `COMMIT` | Transaksi 2 berhasil dilakukan. Transaksi | Transaksi 2 berhasil dilakukan. |

| Transaksi 1 | Transaksi 2 | SQLServer **READ COMMITTED** | Ikan babelfish **READ COMMITTED** |
|---|---|---|---|
| | | 1 sekarang tidak diblokir dan melihat pembaruan dari transaksi 2. | |
| `SELECT * FROM employee;` | Idle pada transaksi | Transaksi 1 memperbarui baris dengan id = 1. | Transaksi 1 memperbarui baris dengan id = 3. |

Babelfish **READ COMMITTED** dibandingkan dengan tingkat isolasi SQL Server **READ COMMITTED SNAPSHOT**

Tabel berikut memberikan rincian tentang perilaku pemblokiran baris yang baru dimasukkan ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `READ COMMITTED SNAPSHOT` isolasi di SQL Server dibandingkan dengan implementasi `READ COMMITTED` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **READ COMMITTED SNAPSHOT** | Ikan babelfish **READ COMMITTED** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;` | `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;` | Tidak ada | Tidak ada |
| `INSERT INTO employee VALUES (4, 'D', 40);` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `UPDATE employee SET age = 99;` | Langkah diblokir sampai transaksi 1 dilakukan. Baris yang | Diperbarui tiga baris. Baris yang baru |

| Transaksi 1 | Transaksi 2 | SQLServer **READ COMMITTED SNAPSHOT** | Ikan babelfish **READ COMMITTED** |
|---|---|---|---|
| | | dimasukkan dikunci oleh transaksi 1. | dimasukkan belum terlihat. |
| `COMMIT` | Idle pada transaksi | Berkomitmen sukses. Transaksi 2 sekarang tidak diblokir. | Berkomitmen sukses. |
| Idle pada transaksi | `SELECT * FROM employee;` | Semua 4 baris memiliki usia = 99. | Baris dengan id = 4 memiliki nilai usia 40 karena tidak terlihat oleh transaksi 2 selama kueri pembaruan. Baris lainnya diperbarui ke age=99. |

Babelfish **`REPEATABLE READ`** dibandingkan dengan tingkat isolasi SQL Server **`REPEATABLE READ`**

Tabel berikut memberikan rincian tentang perilaku pemblokiran baca-tulis ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `REPEATABLE READ` isolasi di SQL Server dibandingkan dengan implementasi `REPEATABLE READ` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **`REPEATABLE READ`** | Ikan babelfish **`REPEATABLE READ`** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | Tidak ada | Tidak ada |

| Transaksi 1 | Transaksi 2 | SQLServer **REPEATABLE READ** | Ikan babelfish **REPEATABLE READ** |
|---|---|---|---|
| `SELECT * FROM employee;` | Idle pada transaksi | Tidak ada | Tidak ada |
| `UPDATE employee SET name='A_TXN1' WHERE id=1;` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `SELECT * FROM employee WHERE id != 1;` | Tidak ada | Tidak ada |
| Idle pada transaksi | `SELECT * FROM employee;` | Transaksi 2 diblokir sampai transaksi 1 dilakukan. | Transaksi 2 berlangsung secara normal. |
| `COMMIT` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `SELECT * FROM employee;` | Pembaruan dari transaksi 1 terlihat. | Pembaruan dari transaksi 1 tidak terlihat. |
| `COMMIT` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `SELECT * FROM employee;` | melihat pembaruan dari transaksi 1. | melihat pembaruan dari transaksi 1. |

Tabel berikut memberikan rincian tentang perilaku pemblokiran tulis-tulis ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `REPEATABLE READ` isolasi di SQL Server dibandingkan dengan implementasi `REPEATABLE READ` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **REPEATABLE READ** | Ikan babelfish **REPEATABLE READ** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | Tidak ada | Tidak ada |
| `UPDATE employee SET name='A_TXN1' WHERE id=1;` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `UPDATE employee SET name='A_TXN2' WHERE id=1;` | Transaksi 2 diblokir. | Transaksi 2 diblokir. |
| `COMMIT` | Idle pada transaksi | Komit berhasil dan transaksi 2 telah diblokir. | Komit berhasil dan transaksi 2 gagal dengan kesalahan tidak dapat membuat serial akses karena pembaruan bersamaan. |
| Idle pada transaksi | `COMMIT` | Berkomitmen sukses. | Transaksi 2 telah dibatalkan. |
| Idle pada transaksi | `SELECT * FROM employee;` | Baris dengan id=1 memiliki TX2 name='a_ '. | Baris dengan id=1 memiliki TX1 name='a_ '. |

Tabel berikut memberikan rincian tentang perilaku membaca hantu ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `REPEATABLE READ` isolasi di SQL Server dibandingkan dengan implementasi `REPEATABLE READ` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **`REPEATABLE READ`** | Ikan babelfish **`REPEATABLE READ`** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `INSERT INTO employee VALUES (4, 'NewRowName', 20);` | Transaksi 2 berlangsung tanpa pemblokiran. | Transaksi 2 berlangsung tanpa pemblokiran. |
| Idle pada transaksi | `SELECT * FROM employee;` | Baris yang baru dimasukkan terlihat. | Baris yang baru dimasukkan terlihat. |
| Idle pada transaksi | `COMMIT` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Baris baru yang dimasukkan oleh transaksi 2 terlihat. | Baris baru yang dimasukkan oleh transaksi 2 tidak terlihat. |
| `COMMIT` | Idle pada transaksi | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Baris yang baru dimasukkan terlihat. | Baris yang baru dimasukkan terlihat. |

Tabel berikut memberikan rincian kapan transaksi bersamaan dijalankan dan hasil akhir yang berbeda saat menggunakan tingkat `REPEATABLE READ` isolasi di SQL Server dibandingkan dengan implementasi `REPEATABLE READ` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **`REPEATABLE READ`** | Ikan babelfish **`REPEATABLE READ`** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | Tidak ada | Tidak ada |
| `UPDATE employee SET age = 100 WHERE age IN (SELECT MIN(age) FROM employee);` | Idle pada transaksi | Transaksi 1 memperbarui baris dengan id 1. | Transaksi 1 memperbarui baris dengan id 1. |
| Idle pada transaksi | `UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);` | Transaksi 2 diblokir karena SELECT pernyataan mencoba membaca baris yang dikunci oleh UPDATE kueri dalam transaksi 1. | Transaksi 2 berlangsung tanpa pemblokiran karena baca tidak pernah diblokir, SELECT pernyataan dijalankan dan akhirnya baris dengan id = 3 diperbarui karena perubahan transaksi 1 belum terlihat. |
| Idle pada transaksi | `SELECT * FROM employee;` | Langkah ini dilakukan setelah transaksi 1 dilakukan. Baris dengan id = 1 diperbarui oleh | Baris dengan id = 3 diperbarui oleh Transaksi 2. |

| Transaksi 1 | Transaksi 2 | SQLServer **REPEATABLE READ** | Ikan babelfish **REPEATABLE READ** |
|---|---|---|---|
| | | transaksi 2 pada langkah sebelumnya dan terlihat di sini. | |
| `COMMIT` | Idle pada transaksi | Transaksi 2 sekarang tidak diblokir. | Berkomitmen sukses. |
| Idle pada transaksi | `COMMIT` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Kedua transaksi mengeksekusi pembaruan pada baris dengan id = 1. | Baris yang berbeda diperbarui dengan transaksi 1 dan 2. |

Babelfish **SERIALIZABLE** dibandingkan dengan tingkat isolasi SQL Server **SERIALIZABLE**

Tabel berikut memberikan rincian tentang kunci rentang ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `SERIALIZABLE` isolasi di SQL Server dibandingkan dengan implementasi `SERIALIZABLE` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Tidak ada | Tidak ada |

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| Idle pada transaksi | `INSERT INTO employee VALUES (4, 'D', 35);` | Transaksi 2 diblokir sampai transaksi 1 dilakukan. | Transaksi 2 berlangsung tanpa pemblokiran. |
| Idle pada transaksi | `SELECT * FROM employee;` | Tidak ada | Tidak ada |
| `COMMIT` | Idle pada transaksi | Transaksi 1 berhasil dilakukan. Transaksi 2 sekarang tidak diblokir. | Transaksi 1 berhasil dilakukan. |
| Idle pada transaksi | `COMMIT` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Baris yang baru dimasukkan terlihat. | Baris yang baru dimasukkan terlihat. |

Tabel berikut memberikan rincian kapan transaksi bersamaan dijalankan dan hasil akhir yang berbeda saat menggunakan tingkat `SERIALIZABLE` isolasi di SQL Server dibandingkan dengan implementasi `SERIALIZABLE` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | Tidak ada | Tidak ada |
| Idle pada transaksi | `INSERT INTO employee VALUES (4, 'D', 40);` | Tidak ada | Tidak ada |

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `UPDATE employee SET age =99 WHERE id = 4;` | Idle pada transaksi | Transaksi 1 diblokir sampai transaksi 2 dilakukan. | Transaksi 1 berlangsung tanpa pemblokiran. |
| Idle pada transaksi | `COMMIT` | Transaksi 2 berhasil dilakukan. Transaksi 1 sekarang tidak diblokir. | Transaksi 2 berhasil dilakukan. |
| `COMMIT` | Idle pada transaksi | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Baris yang baru dimasukkan terlihat dengan nilai usia = 99. | Baris yang baru dimasukkan terlihat dengan nilai usia = 40. |

Tabel berikut memberikan rincian ketika Anda `INSERT` ke dalam tabel dengan kendala yang unik. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `SERIALIZABLE` isolasi di SQL Server dibandingkan dengan implementasi `SERIALIZABLE` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | Tidak ada | Tidak ada |
| Idle pada transaksi | `INSERT INTO employee VALUES (4, 'D', 40);` | Tidak ada | Tidak ada |

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `INSERT INTO employee VALUES ((SELECT MAX(id)+1 FROM employee), 'E', 50);` | Idle pada transaksi | Transaksi 1 diblokir sampai transaksi 2 dilakukan. | Transaksi 1 diblokir sampai transaksi 2 dilakukan. |
| Idle pada transaksi | `COMMIT` | Transaksi 2 berhasil dilakukan. Transaksi 1 sekarang tidak diblokir. | Transaksi 2 berhasil dilakukan. Transaksi 1 dibatalkan dengan kesalahan nilai kunci duplikat melanggar batasan unik. |
| `COMMIT` | Idle pada transaksi | Transaksi 1 berhasil dilakukan. | Transaksi 1 komit gagal dengan tidak dapat membuat serial akses karena ketergantungan baca atau tulis di antara transaksi. |
| `SELECT * FROM employee;` | Idle pada transaksi | baris (5, 'E', 50) dimasukkan. | Hanya ada 4 baris. |

Di Babelfish, transaksi bersamaan yang berjalan dengan Isolation Level serializable akan gagal dengan kesalahan anomali serialisasi jika eksekusi transaksi ini tidak konsisten dengan semua kemungkinan eksekusi serial (satu per satu) dari transaksi tersebut.

Tabel berikut memberikan rincian tentang anomali serialisasi ketika transaksi bersamaan dijalankan. Ini menunjukkan hasil yang diamati saat menggunakan tingkat `SERIALIZABLE` isolasi di SQL Server dibandingkan dengan implementasi `SERIALIZABLE` Babelfish.

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Tidak ada | Tidak ada |
| `UPDATE employee SET age=5 WHERE age=10;` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `SELECT * FROM employee;` | Transaksi 2 diblokir sampai transaksi 1 dilakukan. | Transaksi 2 berlangsung tanpa pemblokiran. |
| Idle pada transaksi | `UPDATE employee SET age=35 WHERE age=30;` | Tidak ada | Tidak ada |
| `COMMIT` | Idle pada transaksi | Transaksi 1 berhasil dilakukan. | Transaksi 1 dilakukan terlebih dahulu dan mampu melakukan komit dengan sukses. |
| Idle pada transaksi | `COMMIT` | Transaksi 2 berhasil dilakukan. | Komit transaksi 2 gagal dengan kesalahan serialisasi, seluruh transaksi telah dibatalkan. Coba lagi transaksi 2. |

| Transaksi 1 | Transaksi 2 | SQLServer **SERIALIZABLE** | Ikan babelfish **SERIALIZABLE** |
|---|---|---|---|
| `SELECT * FROM employee;` | Idle pada transaksi | Perubahan dari kedua transaksi terlihat. | Transaksi 2 digulirkan kembali. Hanya transaksi 1 perubahan yang terlihat. |

Di Babelfish, anomali serialisasi hanya mungkin terjadi jika semua transaksi bersamaan dijalankan pada tingkat isolasi. `SERIALIZABLE` Dalam tabel berikut, mari kita ambil contoh di atas tetapi atur transaksi 2 ke tingkat isolasi `REPEATABLE READ` sebagai gantinya.

| Transaksi 1 | Transaksi 2 | SQLTingkat isolasi server | Tingkat isolasi babelfish |
|---|---|---|---|
| `BEGIN TRANSACTION` | `BEGIN TRANSACTION` | Tidak ada | Tidak ada |
| `SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;` | `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;` | Tidak ada | Tidak ada |
| `SELECT * FROM employee;` | Idle pada transaksi | Tidak ada | Tidak ada |
| `UPDATE employee SET age=5 WHERE age=10;` | Idle pada transaksi | Tidak ada | Tidak ada |
| Idle pada transaksi | `SELECT * FROM employee;` | Transaksi 2 diblokir sampai transaksi 1 dilakukan. | Transaksi 2 berlangsung tanpa pemblokiran. |
| Idle pada transaksi | `UPDATE employee SET age=35 WHERE age=30;` | Tidak ada | Tidak ada |

| Transaksi 1 | Transaksi 2 | SQLTingkat isolasi server | Tingkat isolasi babelfish |
|---|---|---|---|
| `COMMIT` | Idle pada transaksi | Transaksi 1 berhasil dilakukan. | Transaksi 1 berhasil dilakukan. |
| Idle pada transaksi | `COMMIT` | Transaksi 2 berhasil dilakukan. | Transaksi 2 berhasil dilakukan. |
| `SELECT * FROM employee;` | Idle pada transaksi | Perubahan dari kedua transaksi terlihat. | Perubahan dari kedua transaksi terlihat. |

## Menggunakan fitur Babelfish dengan implementasi terbatas

Setiap versi baru Babelfish menambahkan dukungan untuk fitur yang lebih selaras dengan fungsionalitas dan perilaku T-SQL. Namun, ada beberapa fitur yang tidak didukung dan perbedaan dalam implementasi saat ini. Berikut ini, Anda dapat menemukan informasi tentang perbedaan fungsional antara Babelfish dan T-SQL, dengan beberapa solusi atau catatan penggunaan.

Pada versi 1.2.0 dari Babelfish, fitur berikut saat ini memiliki implementasi terbatas:

- Katalog SQL Server (tampilan sistem) – Katalog `sys.sysconfigures`, `sys.syscurconfigs`, dan `sys.configurations` mendukung konfigurasi hanya-baca tunggal saja. Saat ini, `sp_configure` belum didukung. Untuk informasi selengkapnya tentang tampilan SQL Server lainnya yang diterapkan oleh Babelfish, lihat Mendapatkan informasi dari katalog sistem Babelfish.
- Izin GRANT – GRANT…TO PUBLIC didukung, tetapi GRANT..TO PUBLIC WITH GRANT OPTION saat ini tidak didukung.
- Rantai kepemilikan dan batasan mekanisme izin SQL Server – Di Babelfish, rantai kepemilikan SQL Server berfungsi untuk tampilan tetapi tidak untuk prosedur yang disimpan. Ini berarti bahwa prosedur harus diberikan akses eksplisit ke objek lain yang dimiliki oleh pemilik yang sama dengan prosedur panggilan. Di SQL Server, memberikan izin EXECUTE pemanggil pada prosedur cukup untuk memanggil objek lain yang dimiliki oleh pemilik yang sama. Di Babelfish, pemanggil juga harus diberikan izin pada objek yang diakses oleh prosedur.
- Resolusi referensi objek yang tidak memenuhi syarat (tanpa nama skema) – Ketika objek SQL (prosedur, tampilan, fungsi atau pemicu) mereferensikan objek tanpa memenuhi syarat dengan nama skema, SQL Server menyelesaikan nama skema objek dengan menggunakan nama skema

objek SQL tempat referensi terjadi. Saat ini, Babelfish memiliki penyelesaian yang berbeda, dengan menggunakan skema default dari pengguna basis data yang menjalankan prosedur.

- Perubahan, sesi, dan koneksi skema default – Jika pengguna mengubah skema default-nya dengan `ALTER USER...WITH DEFAULT SCHEMA`, perubahan akan segera berlaku di sesi tersebut. Namun, untuk sesi lain yang saat ini terhubung milik pengguna yang sama, waktunya berbeda, sebagai berikut:
  - Untuk SQL Server: – Perubahan akan langsung berlaku di semua koneksi lain untuk pengguna ini.
  - Untuk Babelfish: – Perubahan hanya berlaku di koneksi baru untuk pengguna ini.
- Implementasi tipe data ROWVERSION dan TIMESTAMP dan pengaturan escape hatch – Tipe data ROWVERSION dan TIMESTAMP sekarang didukung di Babelfish. Untuk menggunakan ROWVERSION atau TIMESTAMP di Babelfish, Anda harus mengubah pengaturan untuk escape hatch `babelfishpg_tsql.escape_hatch_rowversion` dari default (strict) menjadi `ignore`. Implementasi Babelfish dari tipe data ROWVERSION dan TIMESTAMP sebagian besar identik secara semantik dengan SQL Server, dengan pengecualian berikut:
  - Fungsi @@DBTS default berperilaku mirip dengan SQL Server, tetapi dengan perbedaan kecil. Daripada mengembalikan nilai yang terakhir digunakan untuk `SELECT @@DBTS`, Babelfish menghasilkan stempel waktu baru, karena mesin basis data PostgreSQL yang mendasarinya dan implementasi kontrol konkurensi multi-versi (MVCC).
  - Di SQL Server, setiap baris yang disisipkan atau diperbarui mendapat ROWVERSION/TIMESTAMP value. In Babelfish, every inserted row updated by the same statement is assigned the same ROWVERSION/TIMESTAMP nilai unik.

    Misalnya, ketika pernyataan UPDATE atau pernyataan INSERT-SELECT mempengaruhi beberapa baris, di SQL Server, semua baris yang terpengaruh memiliki nilai yang berbeda di kolom mereka. ROWVERSION/TIMESTAMP Dalam Babelfish (PostgreSQL), baris memiliki nilai yang sama.
  - Di SQL Server, saat Anda membuat tabel baru dengan SELECT-INTO, Anda dapat memberikan nilai eksplisit (seperti NULL) ke to-be-created ROWVERSION/TIMESTAMP column. When you do the same thing in Babelfish, an actual ROWVERSION/TIMESTAMP nilai yang ditetapkan ke setiap baris dalam tabel baru untuk Anda, oleh Babelfish.

  Perbedaan kecil dalam ROWVERSION/TIMESTAMP tipe data ini seharusnya tidak berdampak buruk pada aplikasi yang berjalan di Babelfish.
- Pembuatan skema, kepemilikan, dan izin – Izin untuk membuat dan mengakses objek dalam skema yang dimiliki oleh pengguna non-DBO (menggunakan `CREATE SCHEMA` *`schema name`*

`AUTHORIZATION` *`user name`*) berbeda untuk pengguna SQL Server dan Babelfish non-DBO, seperti yang ditunjukkan pada tabel berikut:

| Pengguna basis data (non-DBO) yang memiliki skema dapat melakukan hal berikut: | SQL Server | Babelfish |
|---|---|---|
| Buat objek dalam skema tanpa pemberian izin tambahan oleh DBO? | Tidak | Ya |
| Akses objek yang dibuat oleh DBO dalam skema tanpa pemberian izin tambahan? | Ya | Tidak |

- CREATE OR ALTER VIEW /ALTER VIEW sintaks - Dukungan untuk sintaks ini di Babelfish memiliki batasan sebagai berikut:
  - Pernyataan ini tidak dapat digunakan pada tampilan yang memiliki pemicu INSTEAD-OF terpasang.
  - Pernyataan ini tidak dapat digunakan pada pandangan yang memiliki pandangan lain berdasarkan pandangan ini.

## Meningkatkan performa kueri Babelfish

Anda dapat mencapai pemrosesan kueri yang lebih cepat di Babelfish menggunakan petunjuk kueri dan pengoptimal PostgreSQL.

Topik

- Menggunakan rencana explain untuk meningkatkan performa kueri Babelfish
- Menggunakan petunjuk kueri T-SQL untuk meningkatkan kinerja kueri Babelfish

Anda juga dapat meningkatkan performa kueri menggunakan prosedur `sp_babelfish_volatility`. Untuk informasi selengkapnya, lihat sp_babelfish_volatility.

Anda juga dapat meningkatkan kinerja kueri menggunakan transformasi subquery dan cache subquery. Lihat informasi yang lebih lengkap di Mengoptimalkan subkueri berkorelasi di Aurora PostgreSQL.

Menggunakan rencana explain untuk meningkatkan performa kueri Babelfish

Mulai versi 2.1.0, Babelfish menyertakan dua fungsi yang secara transparan menggunakan pengoptimal PostgreSQL guna menghasilkan perkiraan dan rencana kueri aktual untuk kueri T-SQL di port TDS. Fungsi tersebut mirip dengan penggunaan SET STATISTICS PROFILE atau SET SHOWPLAN_ALL dengan basis data SQL Server untuk mengidentifikasi dan meningkatkan kueri yang berjalan lambat.

> **Note**
>
> Mendapatkan paket kueri dari fungsi, alur kontrol, dan kursor saat ini tidak didukung.

Di tabel, Anda dapat menemukan perbandingan fungsi explain rencana kueri di SQL Server, Babelfish, dan PostgreSQL.

| SQL Server | Babelfish | PostgreSQL |
|---|---|---|
| SHOWPLAN_ALL | BABELFISH_SHOWPLAN_ALL | EXPLAIN |
| STATISTICS PROFILE | BABELFISH_STATISTICS PROFILE | EXPLAIN ANALYZE |
| Menggunakan pengoptimal SQL Server | Menggunakan pengoptimal PostgreSQL | Menggunakan pengoptimal PostgreSQL |
| Format input dan output SQL Server | Format input SQL Server dan output PostgreSQL | Format input dan output PostgreSQL |
| Ditetapkan untuk sesi | Ditetapkan untuk sesi | Berlaku untuk pernyataan tertentu |
| Mendukung hal berikut:<br>• SELECT<br>• INSERT<br>• UPDATE | Mendukung hal berikut:<br>• SELECT<br>• INSERT<br>• UPDATE | Mendukung hal berikut:<br>• SELECT<br>• INSERT<br>• UPDATE |

| SQL Server | Babelfish | PostgreSQL |
| --- | --- | --- |
| • DELETE<br>• CURSOR<br>• CREATE<br>• EXECUTE<br>• EXEC dan fungsi, termasuk aliran kontrol (CASE, WHILE-BREAK-CONTINUE, WAITFOR, BEGIN-END, IF-ELSE, dan sebagainya) | • DELETE<br>• CREATE<br>• EXECUTE<br>• EXEC<br>• RAISEERROR<br>• THROW<br>• PRINT<br>• USE | • DELETE<br>• CURSOR<br>• CREATE<br>• EXECUTE |

Gunakan fungsi Babelfish sebagai berikut:

- SET BABELFISH_SHOWPLAN_ALL [ON|OFF] – Tetapkan ke ON untuk menghasilkan perkiraan rencana eksekusi kueri. Fungsi ini mengimplementasikan perilaku perintah `EXPLAIN` PostgreSQL. Gunakan perintah ini untuk mendapatkan rencana explain untuk kueri tertentu.
- SET BABELFISH_STATISTICS PROFILE [ON|OFF] – Tetapkan ke ON untuk rencana eksekusi kueri aktual. Fungsi ini mengimplementasikan perilaku perintah `EXPLAIN ANALYZE` PostgreSQL.

Untuk informasi selengkapnya tentang `EXPLAIN` dan `EXPLAIN ANALYZE` PostgreSQL, lihat EXPLAIN di dokumentasi PostgreSQL.

> **Note**
>
> Mulai versi 2.2.0, Anda dapat menetapkan parameter `escape_hatch_showplan_all` ke ignore untuk menghindari penggunaan awalan BABELFISH_ dalam sintaks SQL Server untuk perintah SET `SHOWPLAN_ALL` dan `STATISTICS PROFILE`.

Misalnya, urutan perintah berikut mengaktifkan perencanaan kueri, lalu mengembalikan perkiraan rencana eksekusi kueri untuk pernyataan SELECT tanpa menjalankan kueri. Contoh ini menggunakan contoh basis data `northwind` SQL Server menggunakan alat baris perintah `sqlcmd` untuk mengkueri port TDS:

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN


-------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

Ketika Anda selesai meninjau dan menyesuaikan kueri, Anda dapat menonaktifkan fungsi seperti yang ditunjukkan berikut:

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

Dengan menetapkan BABELFISH_STATISTICS PROFILE ke ON, setiap kueri yang dieksekusi akan mengembalikan set hasil regulernya, yang diikuti oleh kumpulan hasil tambahan yang menunjukkan rencana eksekusi kueri aktual. Babelfish menghasilkan rencana kueri yang menyediakan set hasil tercepat saat menginvokasi pernyataan SELECT.

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

Set hasil dan rencana kueri dikembalikan (contoh ini hanya menunjukkan rencana kueri).

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

Untuk mempelajari cara menganalisis kueri dan hasil yang dikembalikan oleh pengoptimal PostgreSQL, lihat explain.depesz.com. Untuk informasi lebih lanjut tentang EXPLAIN dan EXPLAIN ANALYZE PostgreSQL, lihat EXPLAIN dalam dokumentasi PostgreSQL.

Parameter yang mengontrol opsi explain Babelfish

Anda dapat menggunakan parameter yang ditunjukkan dalam tabel berikut untuk mengontrol jenis informasi yang ditampilkan oleh rencana kueri Anda.

| Parameter | Deskripsi |
| --- | --- |
| babelfishpg_tsql.explain_buffers | Boolean yang mengaktifkan (dan menonakti fkan) informasi penggunaan buffer untuk pengoptimal. (Default: off) (Diizinkan: off, on) |
| babelfishpg_tsql.explain_costs | Boolean yang mengaktifkan (dan menonakti fkan) perkiraan startup dan informasi total biaya untuk pengoptimal. (Default: on) (Diizinkan: off, on) |

| Parameter | Deskripsi |
| --- | --- |
| babelfishpg_tsql.explain_format | Menentukan format output untuk rencana `EXPLAIN`. (Default: text) (Diizinkan: text, xml/ json, yaml) |
| babelfishpg_tsql.explain_settings | Boolean yang mengaktifkan (atau menonakti fkan) penyertaan informasi tentang parameter konfigurasi dalam output rencana EXPLAIN. (Default: off) (Diizinkan: off, on) |
| babelfishpg_tsql.explain_summary | Boolean yang mengaktifkan (atau menonakti fkan) informasi ringkasan seperti total waktu setelah rencana kueri. (Default: on) (Diizinkan: off, on) |
| babelfishpg_tsql.explain_timing | Boolean yang mengaktifkan (atau menonakti fkan) waktu startup aktual dan waktu yang dihabiskan di setiap simpul dalam output. (Default: on) (Diizinkan: off, on) |
| babelfishpg_tsql.explain_verbose | Boolean yang mengaktifkan (atau menonakti fkan) versi rencana explain paling detail. (Default: off) (Diizinkan: off, on) |
| babelfishpg_tsql.explain_wal | Boolean yang mengaktifkan (atau menonakti fkan) pembuatan informasi catatan WAL sebagai bagian dari rencana explain. (Default: off) (Diizinkan: off, on) |

Anda dapat memeriksa nilai parameter terkait Babelfish pada sistem Anda dengan menggunakan klien PostgreSQL atau klien SQL Server. Jalankan perintah berikut untuk mendapatkan nilai parameter Anda saat ini:

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

Di output berikut, Anda dapat melihat bahwa semua pengaturan di klaster Babelfish DB khusus ini berada pada nilai defaultnya. Tidak semua output ditampilkan dalam contoh ini.

```
             name                    setting                      short_desc
--------------------------------- --------
 ---------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation

(8 rows affected)
```

Anda dapat mengubah pengaturan untuk parameter ini menggunakan `sp_babelfish_configure`, seperti yang ditunjukkan pada contoh berikut.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

Jika Anda ingin membuat pengaturan permanen di tingkat seluruh klaster, sertakan kata kunci server, seperti yang ditunjukkan di contoh berikut.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

Menggunakan petunjuk kueri T-SQL untuk meningkatkan kinerja kueri Babelfish

Dimulai dengan versi 2.3.0, Babelfish mendukung penggunaan petunjuk kueri. menggunakan `pg_hint_plan`. Di Aurora PostgreSQL, `pg_hint_plan` diinstal secara default. Untuk informasi

selengkapnya tentang ekstensi PostgreSQL `pg_hint_plan`, lihat https://github.com/ossc-db/pg_hint_plan. Untuk detail tentang versi ekstensi ini yang didukung oleh Aurora PostgreSQL, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

Pengoptimal kueri dirancang dengan baik untuk menemukan rencana eksekusi yang optimal untuk pernyataan SQL. Saat memilih paket, pengoptimal kueri mempertimbangkan model biaya mesin, serta statistik kolom dan tabel. Namun, rencana yang disarankan mungkin tidak memenuhi kebutuhan set data Anda. Dengan demikian, petunjuk kueri mengatasi masalah kinerja untuk meningkatkan rencana eksekusi. `query hint` adalah sintaks yang ditambahkan ke standar SQL yang menginstruksikan mesin basis data tentang cara mengeksekusi kueri. Misalnya, petunjuk dapat menginstruksikan mesin untuk mengikuti pemindaian berurutan dan mengganti rencana apa pun yang telah dipilih oleh pengoptimal kueri.

Mengaktifkan petunjuk kueri T-SQL di Babelfish

Saat ini, Babelfish mengabaikan semua petunjuk T-SQL secara default. Untuk menerapkan petunjuk T-SQL, jalankan perintah `sp_babelfish_configure` dengan nilai enable_pg_hint dalam kondisi ON.

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

Anda dapat membuat pengaturan aktif secara permanen pada tingkat klaster dengan memasukkan kata kunci server. Untuk mengonfigurasi pengaturan hanya untuk sesi saat ini, jangan gunakan server.

Setelah `enable_pg_hint` ON, Babelfish menerapkan petunjuk T-SQL berikut.

- Petunjuk INDEX
- Petunjuk JOIN
- Petunjuk FORCE ORDER
- Petunjuk MAXDOP

Misalnya, urutan perintah berikut mengaktifkan `pg_hint_plan`.

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
```

```
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

Tidak ada petunjuk yang diterapkan pada pernyataan SELECT. Rencana kueri tanpa petunjuk dikembalikan.

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

Tidak ada petunjuk yang diterapkan pada pernyataan SELECT. Output berikut menunjukkan bahwa rencana kueri dengan penggabungan yang digabung dikembalikan.

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

Batasan

Saat menggunakan petunjuk kueri, pertimbangkan batasan berikut:

- Jika rencana kueri di-cache sebelum `enable_pg_hint` diaktifkan, petunjuk tidak akan diterapkan di sesi yang sama. Ini akan diterapkan di sesi baru.
- Jika nama skema diberikan secara eksplisit, petunjuk tidak dapat diterapkan. Anda dapat menggunakan alias tabel sebagai solusi.
- Petunjuk kueri tidak dapat diterapkan ke tampilan dan sub-kueri.
- Petunjuk tidak berfungsi untuk pernyataan UPDATE/DELETE dengan JOIN.
- Petunjuk indeks untuk indeks atau tabel yang tidak ada diabaikan.
- Petunjuk FORCE ORDER tidak berfungsi untuk HASH JOIN dan yang bukan ANSI JOIN.

## Menggunakan ekstensi Aurora PostgreSQL dengan Babelfish

Aurora PostgreSQL menyediakan ekstensi untuk bekerja dengan layanan lain. AWS Ini adalah ekstensi opsional yang mendukung berbagai kasus penggunaan, seperti menggunakan Amazon S3 dengan klaster DB Anda untuk mengimpor atau mengekspor data.

- Untuk mengimpor data dari bucket Amazon S3 ke klaster DB Babelfish, siapkan ekstensi Aurora PostgreSQL `aws_s3`. Ekstensi ini juga memungkinkan Anda mengekspor data dari klaster DB Aurora PostgreSQL ke bucket Amazon S3.
- AWS Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Anda dapat menggunakan fungsi Lambda untuk melakukan hal-hal seperti memproses notifikasi peristiwa dari instans DB Anda. Untuk mempelajari lebih lanjut tentang Lambda, lihat Apa itu AWS Lambda? di Panduan Developer AWS Lambda . Untuk menginvokasi fungsi Lambda dari klaster DB Babelfish Anda, siapkan ekstensi Aurora PostgreSQL `aws_lambda`.

Untuk menyiapkan ekstensi ini untuk klaster Babelfish Anda, Anda harus terlebih dahulu memberikan izin kepada pengguna internal Babelfish untuk memuat ekstensi. Setelah memberikan izin, Anda kemudian dapat memuat ekstensi Aurora PostgreSQL.

Mengaktifkan ekstensi Aurora PostgreSQL di klaster DB Babelfish Anda

Sebelum Anda dapat memuat ekstensi `aws_s3` atau `aws_lambda`, Anda memberikan hak istimewa yang diperlukan untuk klaster DB Babelfish Anda.

Prosedur berikut menggunakan alat baris perintah PostgreSQL `psql` untuk terhubung ke klaster DB. Untuk informasi selengkapnya, lihat Menggunakan psql untuk terhubung ke klaster DB. Anda juga dapat menggunakan pgAdmin. Lihat perinciannya di Menggunakan pgAdmin untuk terhubung ke klaster DB.

Prosedur ini memuat keduanya `aws_s3` dan `aws_lambda`, satu demi satu. Anda tidak perlu memuat keduanya jika hanya ingin menggunakan salah satu dari ekstensi ini. Ekstensi `aws_commons` diperlukan oleh masing-masing, dan itu dimuat secara default seperti yang ditunjukkan pada output.

Untuk mengatur klaster DB Babelfish Anda dengan hak istimewa untuk ekstensi Aurora PostgreSQL

1. Hubungkan ke klaster DB Babelfish Anda. Gunakan nama untuk pengguna “master” (-U) yang Anda tentukan saat Anda membuat klaster DB Babelfish. Default (`postgres`) ditampilkan dalam contoh.

   Untuk Linux, macOS, atau Unix:

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
   -U postgres \
   -d babelfish_db \
   -p 5432
   ```

   Untuk Windows:

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
   -U postgres ^
   -d babelfish_db ^
   -p 5432
   ```

   Perintah merespons dengan prompt untuk memasukkan kata sandi untuk nama pengguna (-U).

   ```
   Password:
   ```

   Anda perlu memasukkan kata sandi untuk nama pengguna (-U) untuk klaster DB. Ketika Anda berhasil terhubung, Anda melihat output yang serupa dengan berikut ini.

```
psql (13.4)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

2. Berikan hak istimewa kepada pengguna Babelfish internal untuk membuat dan memuat ekstensi.

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
GRANT ROLE
```

3. Buat dan muat ekstensi `aws_s3`. Ekstensi `aws_commons` diperlukan, dan diinstal secara otomatis ketika `aws_s3` diinstal.

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. Buat dan muat ekstensi `aws_lambda`.

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

### Menggunakan Babelfish dengan Amazon S3

Jika Anda belum memiliki bucket Amazon S3 untuk digunakan dengan klaster DB Babelfish Anda, Anda dapat membuatnya. Untuk bucket Amazon S3 yang ingin Anda gunakan, Anda menyediakan akses.

Sebelum mencoba mengimpor atau mengekspor data menggunakan bucket Amazon S3, selesaikan langkah-langkah satu kali berikut.

#### Cara menyiapkan akses instans DB Babelfish ke bucket Amazon S3

1. Buat bucket Amazon S3 untuk instans Babelfish Anda, jika diperlukan. Untuk melakukannya, ikuti petunjuk di Buat ember di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon Bisa-S3-user;.

2. Unggah file ke bucket Amazon S3. Untuk melakukannya, ikuti langkah-langkah dalam Menambahkan objek ke bucket di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon Sederhana -S3-User;.
3. Siapkan izin sesuai kebutuhan:

   - Untuk mengimpor data dari Amazon S3, klaster DB Babelfish memerlukan izin untuk mengakses bucket. Sebaiknya gunakan peran AWS Identity and Access Management (IAM) dan melampirkan kebijakan IAM ke peran tersebut untuk klaster Anda. Untuk melakukannya, ikuti langkah yang ada di Menggunakan peran IAM untuk mengakses bucket Amazon S3.
   - Untuk mengekspor data dari klaster DB Babelfish, klaster Anda harus diberikan akses ke bucket Amazon S3. Seperti halnya mengimpor, sebaiknya gunakan kebijakan dan peran IAM. Untuk melakukannya, ikuti langkah yang ada di Menyiapkan akses ke bucket Amazon S3.

Sekarang Anda dapat menggunakan Amazon S3 dengan ekstensi `aws_s3` dengan klaster DB Babelfish Anda.

Untuk mengimpor data dari Amazon S3 ke Babelfish dan untuk mengekspor data Babelfish ke Amazon S3

1. Gunakan ekstensi `aws_s3` dengan klaster DB Babelfish Anda.

   Ketika Anda melakukannya, pastikan untuk mereferensikan tabel seperti yang ada dalam konteks PostgreSQL. Artinya, jika Anda ingin mengimpor ke dalam tabel Babelfish bernama `[database].[schema].[tableA]`, lihat tabel itu seperti `database_schema_tableA` dalam fungsi `aws_s3`:

   - Untuk contoh penggunaan fungsi `aws_s3` untuk mengimpor data, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL.
   - Untuk contoh penggunaan fungsi `aws_s3` untuk mengekspor data, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.
2. Pastikan untuk mereferensikan tabel Babelfish menggunakan penamaan PostgreSQL saat menggunakan ekstensi `aws_s3` dan Amazon S3, seperti yang ditunjukkan pada tabel berikut.

| Tabel Babelfish | Tabel Aurora PostgreSQL |
| --- | --- |
| *database*.*schema*.*table* | *database*_*schema*_*table* |

Untuk mempelajari selengkapnya tentang menggunakan Amazon S3 dengan Aurora PostgreSQL, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL dan Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.

Menggunakan Babelfish dengan AWS Lambda

Setelah ekstensi `aws_lambda` dimuat di klaster DB Babelfish Anda tetapi sebelum Anda dapat menginvokasi fungsi Lambda, beri Lambda akses ke klaster DB Anda dengan mengikuti prosedur ini.

Untuk mengatur akses klaster DB Babelfish Anda agar berfungsi dengan Lambda

Prosedur ini menggunakan AWS CLI untuk membuat kebijakan dan peran IAM, dan mengaitkannya dengan cluster Babelfish DB.

1. Buat kebijakan IAM yang memungkinkan akses ke Lambda dari klaster DB Babelfish Anda.

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

2. Buat peran IAM yang dapat diambil oleh kebijakan saat runtime.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Lampirkan kebijakan pada peran tersebut.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. Melampirkan peran ke klaster DB Babelfish Anda

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

Setelah Anda menyelesaikan tugas-tugas ini, Anda dapat menginvokasi fungsi Lambda Anda. Untuk informasi lebih lanjut dan contoh pengaturan AWS Lambda untuk Aurora PostgreSQL DB cluster dengan, lihat. AWS LambdaAWS Lambda

Cara menginvokasi fungsi Lambda dari klaster DB Babelfish Anda

AWS Lambda mendukung fungsi yang ditulis dalam Java, Node.js, Python, Ruby, dan bahasa lainnya. Jika fungsi menampilkan teks saat diinvokasi, Anda dapat memanggilnya dari klaster DB Babelfish Anda. Contoh berikut adalah fungsi python placeholder yang menampilkan salam.

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

Saat ini, Babelfish tidak mendukung JSON. Jika fungsi Anda menampilkan JSON, gunakan pembungkus untuk menangani JSON. Misalnya, `lambda_function.py` yang ditunjukkan sebelumnya disimpan di Lambda sebagai `my-function`.

1. Hubungkan ke klaster DB Babelfish Anda menggunakan klien `psql` (atau klien pgAdmin). Untuk informasi selengkapnya, lihat Menggunakan psql untuk terhubung ke klaster DB.

2. Buat pembungkusnya. Contoh ini menggunakan bahasa prosedural PostgreSQL untuk SQL, PL/pgSQL. Untuk mempelajari selengkapnya, lihat PL/pgSQL–Bahasa Prosedural SQL.

```
create or replace function master_dbo.lambda_wrapper()
returns text
language plpgsql
as
$$
declare
   r_status_code integer;
   r_payload text;
begin
   SELECT payload INTO r_payload
     FROM aws_lambda.invoke(  aws_commons.create_lambda_function_arn('my-function',
 'us-east-1')
                            ,'{"body": "Hello from Postgres!"}'::json );
   return r_payload ;
end;
$$;
```

Fungsi ini sekarang dapat dijalankan dari port TDS Babelfish (1433) atau dari port PostgreSQL (5433).

a. Untuk menginvokasi (memanggil) fungsi ini dari port PostgreSQL Anda:

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

Output Anda akan serupa dengan yang berikut ini.

```
status_code |                       payload                        |
 executed_version | log_result
-------------+------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
(1 row)
```

b. Untuk menginvokasi (memanggil) fungsi ini dari port TDS, sambungkan ke port menggunakan klien baris perintah `sqlcmd` SQL Server. Lihat perinciannya di Menggunakan

klien SQL Server untuk terhubung ke klaster DB Anda. Saat terhubung, jalankan yang berikut ini:

```
1> select lambda_wrapper();
2> go
```

Perintah tersebut menghasilkan output yang serupa dengan berikut ini:

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

Untuk mempelajari selengkapnya tentang menggunakan Lambda dengan Aurora PostgreSQL, lihat . Untuk informasi selengkapnya tentang bekerja dengan fungsi Lambda, lihat Memulai dengan Lambda di Panduan Developer AWS Lambda .

Menggunakan pg_stat_statements di Babelfish

Babelfish for Aurora PostgreSQL mendukung ekstensi `pg_stat_statements` dari 3.3.0. Untuk mempelajari lebih lanjut, lihat pg_stat_statements.

Untuk detail tentang versi ekstensi ini didukung oleh Aurora PostgreSQL, lihat Versi ekstensi.

Membuat ekstensi pg_stat_statements

Untuk mengaktifkan `pg_stat_statements`, Anda harus mengaktifkan perhitungan pengidentifikasi Kueri. Hal ini dilakukan secara otomatis jika `compute_query_id` diatur ke on atau `auto` di grup parameter. Nilai default untuk parameter `compute_query_id` adalah `auto`. Anda juga perlu membuat ekstensi ini untuk mengaktifkan fitur ini. Gunakan perintah berikut untuk menginstal ekstensi dari titik akhir T-SQL:

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

Anda dapat mengakses statistik kueri menggunakan kueri berikut:

```
postgres=>select * from pg_stat_statements;
```

> **Note**
>
> Selama instalasi, jika Anda tidak memberikan nama skema untuk ekstensi, maka secara default akan membuatnya dalam skema publik. Untuk mengaksesnya, Anda harus menggunakan tanda kurung siku dengan kualifikasi skema seperti yang ditunjukkan di bawah ini:

```
postgres=>select * from [public].pg_stat_statements;
```

Anda juga dapat membuat ekstensi dari titik akhir PSQL.

Mengotorisasi ekstensi

Secara default, Anda dapat melihat statistik untuk kueri yang dilakukan dalam basis data T-SQL Anda tanpa memerlukan otorisasi apa pun.

Untuk mengakses statistik kueri yang dibuat oleh orang lain, Anda harus memiliki peran PostgreSQL `pg_read_all_stats`. Ikuti langkah-langkah yang disebutkan di bawah ini untuk membuat perintah GRANT pg_read_all_stats.

1. Di T-SQL, gunakan kueri berikut yang menampilkan nama peran PG internal.

   ```
   SELECT rolname FROM pg_roles WHERE oid = USER_ID();
   ```

2. Hubungkan ke basis data Babelfish for Aurora PostgreSQL dengan hak istimewa rds_superuser dan gunakan perintah berikut:

   ```
   GRANT pg_read_all_stats TO <rolname_from_above_query>
   ```

Contoh

Dari titik akhir T-SQL:

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```


rolname
```

```
-------
master_dbo
(1 rows affected)
```

Dari titik akhir PSQL:

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

Anda dapat mengakses statistik kueri menggunakan tampilan pg_stat_statements:

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

Mengatur ulang statistik kueri

Anda dapat menggunakan `pg_stat_statements_reset()` untuk mengatur ulang statistik yang dikumpulkan sejauh ini oleh pg_stat_statements. Untuk mempelajari lebih lanjut, lihat

pg_stat_statements. Saat ini didukung melalui titik akhir PSQL saja. Hubungkan ke Babelfish for Aurora PostgreSQL dengan hak istimewa `rds_superuser`, gunakan perintah berikut:

```
SELECT pg_stat_statements_reset();
```

Batasan

- Saat ini, `pg_stat_statements()` tidak didukung melalui titik akhir T-SQL. Tampilan `pg_stat_statements` adalah cara yang disarankan untuk mengumpulkan statistik.
- Beberapa kueri mungkin ditulis ulang oleh pengurai T-SQL yang diimplementasikan oleh mesin Aurora PostgreSQL, tampilan `pg_stat_statements` akan menampilkan kueri yang ditulis ulang dan bukan kueri asli.

  Contoh

  ```
  select next value for [dbo].[newCounter];
  ```

  Kueri di atas ditulis ulang sebagai berikut dalam tampilan pg_stat_statements.

  ```
  select nextval($1);
  ```
- Berdasarkan alur eksekusi pernyataan, beberapa kueri mungkin tidak dilacak oleh pg_stat_statements dan tidak akan terlihat dalam tampilan. Ini termasuk pernyataan berikut: `use dbname`, `goto`, `print`, `raise error`, `set`, `throw`, `declare cursor`.
- Untuk pernyataan CREATE LOGIN dan ALTER LOGIN, query dan queryid tidak akan ditampilkan. Ini akan menunjukkan hak istimewa yang tidak memadai.
- Tampilan `pg_stat_statements` selalu berisi dua entri di bawah ini, karena ini dijalankan secara internal oleh klien `sqlcmd`.
  - SET QUOTED_IDENTIFIER OFF
  - SET TEXTSIZE 4096

Menggunakan pgvector di Babelfish

pgvector, ekstensi sumber terbuka, memungkinkan Anda mencari data serupa secara langsung dalam database Postgres Anda. Babelfish sekarang mendukung ekstensi ini dimulai dengan versi 15.6 dan 16.2. Untuk informasi lebih lanjut, pgvector Open source Documentation.

### Prasyarat

Untuk mengaktifkan fungsionalitas pgvector, instal ekstensi dalam skema sys menggunakan salah satu metode berikut:

- Jalankan perintah berikut di klien sqlcmd:

```
exec sys.sp_execute_postgresql 'CREATE EXTENSION vector WITH SCHEMA sys';
```

- Connect to `babelfish_db` dan jalankan perintah berikut di psql client:

```
CREATE EXTENSION vector WITH SCHEMA sys;
```

> **Note**
>
> Setelah menginstal ekstensi pgvector, tipe data vektor hanya akan tersedia dalam koneksi database baru yang Anda buat. Koneksi yang ada tidak akan mengenali tipe data baru.

### Fungsionalitas yang Didukung

Babelfish memperluas fungsionalitas T-SQL untuk mendukung hal-hal berikut:

- Menyimpan

  Babelfish sekarang mendukung sintaks yang kompatibel dengan tipe data vektor, meningkatkan kompatibilitas T-SQL-nya. Untuk mempelajari selengkapnya tentang menyimpan data dengan pgvector, lihat Menyimpan.

- Meminta

  Babelfish memperluas dukungan ekspresi T-SQL untuk menyertakan operator kesamaan vektor. Namun, untuk semua kueri lainnya, sintaks T-SQL standar masih diperlukan.

> **Note**
>
> T-SQL tidak mendukung tipe Array, dan driver database tidak memiliki antarmuka untuk menanganinya. Sebagai solusinya, Babelfish menggunakan string teks (varchar/nvarchar) untuk menyimpan data vektor. Misalnya, ketika Anda meminta nilai vektor [1,2,3], Babelfish

akan mengembalikan string '[1,2,3]' sebagai respons. Anda dapat mengurai dan membagi string ini di tingkat aplikasi sesuai kebutuhan Anda.

Untuk mempelajari lebih lanjut tentang kueri data dengan pgvector, lihat Menanyakan.

- Pengindeksan

  T-SQL `Create Index` sekarang mendukung `USING INDEX_METHOD` sintaks. Sekarang Anda dapat menentukan operator pencarian kesamaan yang akan digunakan pada kolom tertentu saat membuat indeks.

  Tata bahasa juga diperluas untuk mendukung operasi kesamaan Vektor pada kolom yang diperlukan (Periksa tata bahasa column_name_list_with_order_for_vector).

```
CREATE [UNIQUE] [clustered] [COLUMNSTORE] INDEX <index_name> ON <table_name> [USING
 vector_index_method] (<column_name_list_with_order_for_vector>)
Where column_name_list_with_order_for_vector is:
    <column_name> [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS | VECTOR_L2_OPS]
 (COMMA simple_column_name [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS |
 VECTOR_L2_OPS])
```

  Untuk mempelajari lebih lanjut tentang pengindeksan data dengan pgvector, lihat Pengindeksan.

- Performa
  - Gunakan `SET BABELFISH_STATISTICS PROFILE ON` untuk men-debug Rencana Kueri dari titik akhir T-SQL.
  - Tingkatkan `max_parallel_workers_get_gather` menggunakan `set_config` fungsi yang didukung di T-SQL.
  - Gunakan `IVFFlat` untuk perkiraan pencarian. Untuk informasi selengkapnya, lihat IVFFlat.

  Untuk meningkatkan kinerja dengan pgvector, lihat Kinerja.

Batasan

- Babelfish tidak mendukung Pencarian Teks Lengkap untuk Pencarian Hibrida. Untuk informasi selengkapnya, lihat Pencarian Hybrid.

- Babelfish saat ini tidak mendukung fungsionalitas pengindeksan ulang. Namun, Anda masih dapat menggunakan PostgreSQL endpoint untuk mengindeks ulang. Untuk informasi lebih lanjut, lihat Menyedot debu.

### Menggunakan pembelajaran mesin Amazon Aurora dengan Babelfish

Anda dapat memperluas kemampuan klaster Babelfish untuk Aurora PostgreSQL DB Anda dengan mengintegrasikannya dengan pembelajaran mesin Amazon Aurora. Integrasi tanpa batas ini memberi Anda akses ke berbagai layanan canggih seperti Amazon Comprehend atau Amazon AI atau SageMaker Amazon Bedrock, masing-masing disesuaikan untuk memenuhi kebutuhan pembelajaran mesin yang berbeda.

Sebagai pengguna Babelfish, Anda dapat menggunakan pengetahuan yang ada tentang sintaks dan semantik T-SQL saat bekerja dengan pembelajaran mesin Aurora. Ikuti instruksi yang diberikan dalam AWS dokumentasi untuk Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora Postgre SQL.

#### Prasyarat

- Sebelum mencoba menyiapkan klaster Babelfish untuk Aurora PostgreSQL DB untuk menggunakan pembelajaran mesin Aurora, Anda harus memahami persyaratan dan prasyarat terkait. Untuk informasi selengkapnya, lihat Persyaratan untuk menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL.
- Pastikan Anda menginstal `aws_ml` ekstensi baik menggunakan titik akhir Postgres atau prosedur `sp_execute_postgresql` penyimpanan.

```
exec sys.sp_execute_postgresql 'Create Extension aws_ml'
```

> **Note**
>
> Saat ini Babelfish tidak mendukung operasi kaskade di Babelfish. `sp_execute_postgresql` Karena `aws_ml` bergantung pada`aws_commons`, Anda harus menginstalnya secara terpisah menggunakan titik akhir Postgres.

```
create extension aws_common;
```

Menangani sintaks dan semantik T-SQL dengan fungsi **aws_ml**

Contoh-contoh berikut menjelaskan bagaimana sintaks dan semantik T-SQL diterapkan ke layanan Amazon ML:

Example : aws_bedrock.invoke_model — Sebuah query sederhana menggunakan fungsi Amazon Bedrock

```
aws_bedrock.invoke_model(
   model_id      varchar,
   content_type  text,
   accept_type   text,
   model_input   text)
Returns Varchar(MAX)
```

Contoh berikut menunjukkan cara memanggil model Anthropic Claude 2 untuk Bedrock menggunakan invoke_model.

```
SELECT aws_bedrock.invoke_model (
    'anthropic.claude-v2', -- model_id
    'application/json', -- content_type
    'application/json', -- accept_type
    '{"prompt": "\n\nHuman:
    You are a helpful assistant that answers questions directly
    and only using the information provided in the context below.
    \nDescribe the answerin detail.\n\nContext: %s \n\nQuestion:
    %s \n\nAssistant:","max_tokens_to_sample":4096,"temperature"
    :0.5,"top_k":250,"top_p":0.5,"stop_sequences":[]}' -- model_input
);
```

Example : aws_comprehend.detect_sentiment — Kueri sederhana menggunakan fungsi Amazon Comprehend

```
aws_comprehend.detect_sentiment(
   input_text varchar,
   language_code varchar,
   max_rows_per_batch int)
Returns table (sentiment varchar, confidence real)
```

Contoh berikut menunjukkan cara memanggil layanan Amazon Comprehend.

```
select sentiment from aws_comprehend.detect_sentiment('This is great', 'en');
```

Example : aws_sagemaker.invoke_endpoint — Sebuah query sederhana menggunakan fungsi Amazon SageMaker

```
aws_sagemaker.invoke_endpoint(
  endpoint_name varchar,
  max_rows_per_batch int,
  VARIADIC model_input "any") -- Babelfish inherits PG's variadic parameter type
Rerurns Varchar(MAX)
```

Karena model_input ditandai sebagai VARIADIC dan bertipe “any”, pengguna dapat meneruskan daftar dengan panjang berapa pun dan tipe data apa pun ke fungsi yang akan bertindak sebagai input input ke model. Contoh berikut menunjukkan cara memanggil SageMaker layanan Amazon.

```
SELECT CAST (aws_sagemaker.invoke_endpoint(
    'sagemaker_model_endpoint_name',
    NULL,
    arg1, arg2 -- model inputs are separate arguments )
AS INT) -- cast the output to INT
```

Untuk informasi lebih rinci tentang penggunaan pembelajaran mesin Aurora dengan Aurora PostgreSQL, lihat. Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora Postgre SQL

Batasan

- Meskipun Babelfish tidak mengizinkan pembuatan array, Babelfish masih dapat menangani data yang mewakili array. Saat Anda menggunakan fungsi seperti array pengembalian `aws_bedrock.invoke_model_get_embeddings` itu, hasilnya dikirimkan sebagai string yang berisi elemen array.

## Babelfish mendukung server tertaut

Babelfish untuk Aurora Postgre SQL mendukung server tertaut dengan menggunakan ekstensi Postgre di versi 3.1.0. SQL `tds_fdw` Untuk bekerja dengan server yang terhubung, Anda harus menginstal ekstensi `tds_fdw`. Untuk informasi selengkapnya tentang ekstensi `tds_fdw`, lihat .

### Menginstal ekstensi **`tds_fdw`**

Anda dapat menginstal ekstensi `tds_fdw` menggunakan metode berikut ini.

#### Menggunakan CREATE EXTENSION dari titik akhir Postgre SQL

1. Connect ke instans Postgre SQL DB Anda pada database Babelfish di port Postgre. SQL Gunakan akun yang memiliki peran rds_superuser.

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=test --dbname=babelfish_db --password
   ```

2. Instal ekstensi `tds_fdw`. Ini adalah proses instalasi satu kali. Anda tidak perlu menginstal ulang saat klaster DB dimulai ulang.

   ```
   babelfish_db=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

#### Memanggil prosedur **`sp_execute_postgresql`** tersimpan dari titik TDS akhir

Babelfish mendukung instalasi ekstensi `tds_fdw` dengan memanggil prosedur `sp_execute_postgresql` dari versi 3.3.0. Anda dapat mengeksekusi SQL pernyataan Postgre dari T- SQL endpoint tanpa keluar dari T-port. SQL Untuk informasi selengkapnya, silakan lihat Bekerja dengan Babelfish untuk prosedur Aurora Postgre SQL

1. Connect ke instans Postgre SQL DB Anda pada database Babelfish di T-port. SQL

   ```
   sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
   ```

2. Instal ekstensi `tds_fdw`.

   ```
   1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
   2>go
   ```

### Fungsionalitas yang didukung

Babelfish mendukung penambahan remote RDS untuk SQL Server atau Babelfish untuk titik akhir Aurora SQL Postgre sebagai server tertaut. Anda juga dapat menambahkan instance SQL Server jarak jauh lainnya sebagai server tertaut. Kemudian, gunakan `OPENQUERY()` untuk mengambil data dari server tertaut ini. Mulai dari Babelfish versi 3.2.0, nama empat bagian juga didukung.

Prosedur tersimpan dan tampilan katalog berikut didukung untuk menggunakan server tertaut.

#### Prosedur tersimpan

- sp_addlinkedserver – Babelfish tidak mendukung parameter `@provstr`.
- sp_addlinkedsrvlogin
  - Anda harus memberikan nama pengguna dan kata sandi jarak jauh eksplisit untuk terhubung ke sumber data jarak jauh. Anda tidak dapat terhubung dengan kredensial mandiri pengguna. Babelfish hanya mendukung `@useself = false`.
  - Babelfish tidak mendukung parameter `@locallogin` karena mengonfigurasi akses server jarak jauh khusus untuk proses masuk lokal tidak didukung.
- sp_linkedservers
- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin – Babelfish tidak mendukung parameter `@locallogin` karena mengonfigurasi akses server jarak jauh khusus untuk proses masuk lokal tidak didukung.
- sp_serveroption – Babelfish mendukung opsi server berikut:
  - waktu habis kueri (dari Babelfish versi 3.2.0)
  - waktu habis koneksi (dari Babelfish versi 3.3.0)
- sp_testlinkedserver (dari Babelfish versi 3.3.0)
- sp_enum_oledb_providers (dari Babelfish versi 3.3.0)

#### Tampilan katalog

- sys.servers
- sys.linked_logins

### Menggunakan enkripsi dalam transit untuk koneksi

Koneksi dari sumber Babelfish untuk server Aurora Postgre ke SQL server jarak jauh target menggunakan enkripsi dalam transit (TLS/SSL), tergantung pada konfigurasi database server jarak jauh. Jika server jarak jauh tidak dikonfigurasi untuk enkripsi, server Babelfish yang membuat permintaan ke basis data jarak jauh kembali ke tidak terenkripsi.

Untuk menegakkan enkripsi koneksi

- Jika server yang ditautkan target adalah instance RDS untuk SQL Server, tetapkan `rds.force_ssl = on` untuk instance SQL Server target. Untuk informasi selengkapnya tentangSSL/TLSkonfigurasi RDS untuk SQL Server, lihat Menggunakan SSL dengan instans Microsoft SQL Server DB
- Jika server yang ditautkan target adalah Babelfish untuk SQL cluster Aurora Postgre, tetapkan `babelfishpg_tds.tds_ssl_encrypt = on` dan untuk server target. `ssl = on` Untuk informasi lebih lanjut tentangSSL/TLS, lihatPengaturan SSL Babelfish dan koneksi klien.

### Menambahkan Babelfish sebagai server tertaut dari Server SQL

Babelfish untuk Aurora Postgre SQL dapat ditambahkan sebagai server tertaut dari Server. SQL Pada database SQL Server, Anda dapat menambahkan Babelfish sebagai server tertaut menggunakan penyedia Microsoft OLE DB untukODBC:. MSDASQL

Ada dua cara untuk mengkonfigurasi Babelfish sebagai server tertaut dari SQL Server menggunakan MSDASQL penyedia:

- Menyediakan string ODBC koneksi sebagai string penyedia.
- Menyediakan Sistem DSN sumber ODBC data sambil menambahkan server yang ditautkan.

### Batasan

- OPENQUERY() hanya berfungsi untuk SELECT dan tidak berfungsi untukDML.
- Nama objek empat bagian hanya berfungsi untuk membaca dan tidak berfungsi untuk memodifikasi tabel jarak jauh. An UPDATE dapat mereferensikan tabel jarak jauh dalam FROM klausa tanpa memodifikasinya.
- Menjalankan prosedur tersimpan terhadap server tertaut Babelfish tidak didukung.
- Peningkatan versi mayor Babelfish mungkin tidak berfungsi jika ada objek yang bergantung pada `OPENQUERY()` atau objek yang direferensikan melalui nama empat bagian. Anda harus

memastikan bahwa objek apa pun yang mereferensikan `OPENQUERY()` atau nama empat bagian dihapus sebelum peningkatan versi mayor.

- Jenis data berikut tidak berfungsi seperti yang diharapkan terhadap server Babelfish jarak jauh: `nvarchar(max)`, `varchar(max)`, `varbinary(max)`, `binary(max)`, dan `time`. Sebaiknya gunakan CAST fungsi untuk mengonversinya ke tipe data yang didukung.

### Contoh

Dalam contoh berikut, instance Babelfish untuk Aurora Postgre terhubung ke SQL instance untuk Server di cloud. RDS SQL

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

Ketika server tertaut sudah terpasang, Anda kemudian dapat menggunakan T- SQL OPENQUERY () atau penamaan empat bagian standar untuk mereferensikan tabel, tampilan, atau objek lain yang didukung, di server jarak jauh:

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

Untuk menghapus server tertaut dan semua kredensial masuk terkait:

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

### Pemecahan Masalah

Anda dapat menggunakan grup keamanan yang sama untuk server sumber dan jarak jauh untuk memungkinkan keduanya berkomunikasi satu sama lain. Grup keamanan harus mengizinkan hanya lalu lintas masuk pada TDS port (1433 secara default) dan IP sumber dalam grup keamanan dapat

diatur sebagai ID grup keamanan itu sendiri. Untuk informasi selengkapnya tentang cara mengatur aturan untuk menghubungkan ke satu instans dari instans yang lain dengan grup keamanan yang sama, lihat Aturan untuk menghubungkan ke satu instans dari sebuah instans dengan grup keamanan yang sama.

Jika akses tidak dikonfigurasi dengan benar, pesan kesalahan yang mirip dengan contoh berikut akan muncul saat Anda mencoba menjalankan kueri server jarak jauh.

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

## Menggunakan Pencarian Teks Lengkap di Babelfish

Dimulai dengan versi 4.0.0, Babelfish menyediakan dukungan terbatas untuk Pencarian Teks Lengkap (). FTS FTSadalah fitur yang kuat dalam database relasional yang memungkinkan pencarian dan pengindeksan data teks yang efisien. Ini memungkinkan Anda untuk melakukan pencarian teks yang kompleks dan mengambil hasil yang relevan dengan cepat. FTSsangat berharga untuk aplikasi yang berhubungan dengan volume besar data tekstual, seperti sistem manajemen konten, platform e-commerce, dan arsip dokumen.

Memahami fitur yang didukung Pencarian Teks Lengkap Babelfish

Babelfish mendukung fitur Pencarian Teks Lengkap berikut:

- CONTAINSKlausul:
  - Dukungan dasar untuk CONTAINS klausa.

```
CONTAINS (
     {
        column_name
     }
     , '<contains_search_condition>'
)
```

> **Note**
>
> Saat ini, hanya bahasa Inggris yang didukung.

  - Penanganan dan terjemahan string `simple_term` pencarian yang komprehensif.
- `FULLTEXT INDEX`Klausul:
  - Hanya mendukung `CREATE FULLTEXT INDEX ON table_name(column_name [...n]) KEY INDEX index_name` pernyataan.
  - Mendukung `DROP FULLTEXT INDEX` pernyataan lengkap.

> **Note**
>
> Untuk mengindeks ulang Indeks Teks Lengkap, Anda harus menjatuhkan Indeks Teks Lengkap dan membuat yang baru di kolom yang sama.

- Karakter khusus dalam kondisi pencarian:
  - Babelfish memastikan bahwa kemunculan tunggal karakter khusus dalam string pencarian ditangani secara efektif.

> **Note**
>
> Sementara Babelfish sekarang mengidentifikasi karakter khusus dalam string pencarian, penting untuk mengenali bahwa hasil yang diperoleh dapat bervariasi dibandingkan dengan yang diperoleh dengan T-. SQL

- Alias tabel di column_name:
  - Dengan dukungan alias tabel, pengguna dapat membuat SQL kueri yang lebih ringkas dan mudah dibaca untuk Pencarian Teks Lengkap.

Keterbatasan dalam Pencarian Teks Lengkap Babelfish

- Saat ini, opsi berikut tidak didukung di Babelfish for `CONTAINS` Clause.
  - Karakter khusus dan Bahasa selain bahasa Inggris tidak didukung. Anda akan menerima pesan kesalahan umum untuk karakter dan bahasa yang tidak didukung

```
Full-text search conditions with special characters or languages other than English
 are not currently supported in Babelfish
```

  - Beberapa kolom seperti `column_list`

  - PROPERTYatribut
  - `prefix_term`,`generation_term`,`generic_proximity_term`,`custom_proximity_term`, dan `weighted_term`
  - Operator Boolean tidak didukung dan Anda akan menerima pesan galat berikut saat digunakan:

    ```
    boolean operators not supported
    ```

  - Nama pengenal dengan titik tidak didukung.
- Saat ini, opsi berikut tidak didukung di Babelfish for `CREATE FULLTEXT INDEX` Clause.
  - [TYPECOLUMNtype_column_name]
  - [LANGUAGElanguage_term]
  - [ STATISTICAL_SEMANTICS ]
  - opsi filegroup katalog
  - dengan opsi
- Membuat katalog teks lengkap tidak didukung. Membuat indeks teks lengkap tidak memerlukan katalog teks lengkap.
- `CREATE FULLTEXT INDEX`tidak mendukung nama pengenal dengan titik.
- Babelfish saat ini tidak mendukung karakter khusus berturut-turut dalam string pencarian. Anda akan menerima pesan galat berikut saat digunakan:

  ```
  Consecutive special characters in the full-text search condition are not currently
   supported in Babelfish
  ```

## Babelfish mendukung tipe data Geospasial

Dimulai dengan versi 3.5.0 dan 4.1.0, Babelfish menyertakan dukungan untuk dua tipe data spasial berikut:

- Jenis data geometri — Tipe data ini dimaksudkan untuk menyimpan data planar atau Euclidean (bumi datar).
- Jenis data geografi — Tipe data ini dimaksudkan untuk menyimpan data ellipsoidal atau round-earth, seperti koordinat lintang dan bujur GPS.

Tipe data ini memungkinkan penyimpanan dan manipulasi data spasial, tetapi dengan keterbatasan.

Memahami tipe data Geospasial di Babelfish

- Tipe data geospasial didukung dalam berbagai objek database seperti tampilan, prosedur, dan tabel.

- Mendukung tipe data titik 2-D untuk menyimpan data lokasi sebagai titik yang ditentukan oleh garis lintang, bujur, dan Pengidentifikasi Sistem Referensi Spasial (SRID) yang valid.
- Aplikasi yang terhubung ke Babelfish melalui driver seperti JDBC, ODBC, DOTNET, dan PYTHON dapat memanfaatkan fitur Geospasial ini.

Fungsi tipe data geometri didukung di Babelfish

- ST GeomFromText (***geometry_tagged_text***, SRID) - Membuat contoh geometri menggunakan representasi Teks Terkenal (WKT).
- ST PointFromText (***point_tagged_text***, SRID) - Membuat contoh titik menggunakan representasi WKT.
- Point (X, Y, SRID) - Membuat contoh titik menggunakan nilai float koordinat x dan y.
- .ST AsText () <geometry_instance>— Mengekstrak representasi WKT dari contoh geometri.
- .stDistance (other_geometry) <geometry_instance>- Menghitung jarak antara dua contoh geometri.
- .STX <geometry_instance>— Mengekstrak koordinat X (bujur) untuk contoh geometri.
- .STY <geometry_instance>— Mengekstrak koordinat Y (lintang) untuk contoh geometri.

Fungsi tipe data geografi yang didukung di Babelfish

- ST GeomFromText (***geography_tagged_text***, SRID) - Membuat instance geografi menggunakan representasi WKT.
- ST PointFromText (***point_tagged_text***, SRID) - Membuat contoh titik menggunakan representasi WKT.
- Point (Lat, Long, SRID) - Membuat instance titik menggunakan nilai float Latitude dan Longitude.
- .ST AsText () <geography_instance>— Mengekstrak representasi WKT dari contoh geografi.
- .stDistance (other_geography) <geography_instance>- Menghitung jarak antara dua contoh geografi.
- .Lat <geography_instance>— Mengekstrak nilai Latitude untuk contoh geografi.

- .Long <geography_instance>— Mengekstrak nilai Bujur untuk contoh geografi.

Keterbatasan pada Babelfish untuk tipe data Geospasial

- Saat ini, Babelfish tidak mendukung fitur yang lebih canggih seperti bendera Z-M untuk instance titik tipe data Geospasial.
- Jenis geometri selain instance titik saat ini tidak didukung:
  - LineString
  - CircularString
  - CompoundCurve
  - Polygon
  - CurvePolygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection
- Saat ini, pengindeksan spasial tidak didukung untuk tipe data Geospasial.
- Hanya fungsi yang terdaftar saat ini didukung untuk tipe data ini. Untuk informasi selengkapnya, lihat Fungsi tipe data geometri didukung di Babelfish dan Fungsi tipe data geografi yang didukung di Babelfish.
- Output fungsi STDistance untuk data Geografi mungkin memiliki variasi presisi kecil dibandingkan dengan T-SQL. Hal ini disebabkan implementasi PostGIS yang mendasarinya. Untuk informasi selengkapnya, lihat ST_Distance
- Untuk kinerja optimal, gunakan tipe data Geospasial bawaan, tanpa membuat lapisan abstraksi tambahan di Babelfish.

> **Tip**
>
> Meskipun Anda dapat membuat tipe data khusus, tidak disarankan untuk membuatnya di atas data Geospasial. Ini dapat menimbulkan kompleksitas, yang berpotensi mengarah pada perilaku tak terduga karena dukungan yang terbatas.

- Dalam Babelfish, nama fungsi Geospasial digunakan sebagai kata kunci dan akan melakukan operasi spasial hanya jika digunakan dengan cara yang dimaksudkan.

> **Tip**
>
> Saat membuat fungsi dan prosedur yang ditentukan pengguna di Babelfish, hindari menggunakan nama yang sama dengan fungsi Geospasial bawaan. Jika Anda memiliki objek database yang ada dengan nama yang sama, gunakan `sp_rename` untuk mengganti nama mereka.

## Memahami partisi di Babelfish

Dimulai dengan versi 4.3.0, Babelfish memperkenalkan partisi tabel dan indeks dengan dukungan terbatas. Bagian berikut memberikan rincian tentang membuat fungsi partisi, mendefinisikan skema partisi, dan mengimplementasikan tabel dan indeks yang dipartisi di Babelfish.

Topik

- Memperkenalkan partisi di Babelfish
- Keterbatasan dan solusi

### Memperkenalkan partisi di Babelfish

- Fungsi partisi:
  - `CREATE PARTITION FUNCTION`: Mendefinisikan bagaimana tabel atau indeks dipartisi dengan menentukan jenis kolom partisi dan rentang nilai untuk setiap partisi.
  - `DROP PARTITION FUNCTION`: Menghapus fungsi partisi yang ada.
- Skema partisi:
  - `CREATE PARTITION SCHEME`: Mendefinisikan pemetaan antara partisi dan filegroup.

    > **Note**
    >
    > Dalam Babelfish, filegroup diperlakukan sebagai objek dummy dan tidak mewakili lokasi penyimpanan fisik.

  - `DROP PARTITION SCHEME`: Menghapus skema partisi yang ada.
- Fungsi sistem:

  - `$PARTITION`: Fungsi sistem ini mengembalikan nomor partisi yang nilai tertentu dalam kolom partisi akan berada dalam tabel partisi tertentu.
- Tabel dan indeks yang dipartisi:
  - `CREATE TABLE ... ON partition_scheme_name (partition_column_name)`: Membuat tabel dipartisi berdasarkan skema partisi tertentu dan kolom partisi.
  - `CREATE INDEX ... ON partition_scheme_name (partition_column_name)`: Membuat indeks dipartisi berdasarkan skema partisi tertentu dan kolom partisi.
- Tampilan sistem untuk mempartisi metadata:

  Tampilan sistem berikut ditambahkan untuk menyediakan metadata yang terkait dengan partisi:
  - `sys.destination_data_spaces`
  - `sys.partitions`
  - `sys.partition_functions`
  - `sys.partition_parameters`
  - `sys.partition_range_values`
  - `sys.partition_schemes`

Keterbatasan dan solusi

Kemampuan partisi SQL Server berikut belum didukung oleh Babelfish:

- `ALTER PARTITION FUNCTION` dan `ALTER PARTITION SCHEME`.

  > **Note**
  >
  > Babelfish tidak mendukung operasi split dan merge. Tentukan semua partisi dalam fungsi partisi selama pembuatan karena Anda tidak dapat menambah atau menghapus partisi nanti.

- Kolom yang dihitung sebagai kolom partisi.
- `INSERT BULK` dan `BCP` utilitas untuk tabel yang dipartisi.
- `LEFT` opsi batas untuk fungsi partisi.
- `SQL_VARIANT` tipe data untuk fungsi partisi.

- `TRUNCATE TABLE ... WITH PARTITION`.
- `ALTER TABLE ... SWITCH PARTITION`.
- Indeks partisi yang tidak selaras seperti skema partisi dan kolom partisi yang berbeda dari tabel yang dipartisi.
- Migrasi DMS dari sumber Babelfish hanya didukung untuk tugas Full Load pada tabel yang dipartisi.
- Penggunaan pemeriksaan dalam fungsi partisi.
- Menggunakan kolom partisi dengan pemeriksaan selain pemeriksaan database default.
- Babelfish tidak mendukung opsi sintaks ini tetapi menyediakan solusi:
  - Penggunaan skema partisi dengan kendala atau indeks dalam pernyataan CREATE TABLE.
  - MENGUBAH TABEL... TAMBAHKAN KENDALA... ON partition_scheme_name (partition_column_name).

> **Note**
>
> Anda dapat mengatur pintu `babelfishpg_tsql.escape_hatch_storage_on_partition` keluar untuk diabaikan. Ini akan memungkinkan parser untuk mengabaikan opsi skema partisi yang digunakan dengan kendala atau indeks, dan backend akan membuat kendala atau indeks individual untuk setiap partisi.

# Memecahkan Masalah Babelfish

Di bawah ini adalah solusi dan ide pemecahan masalah untuk beberapa masalah klaster DB Babelfish.

Topik


- Kegagalan koneksi


## Kegagalan koneksi

Penyebab umum kegagalan koneksi ke klaster DB Aurora baru dengan Babelfish adalah sebagai berikut:

- Grup keamanan tidak mengizinkan akses - Jika Anda mengalami masalah saat menyambung ke Babelfish, pastikan Anda menambahkan alamat IP Anda ke grup EC2 keamanan Amazon default. Anda dapat menggunakan https://checkip.amazonaws.com/untuk menentukan alamat IP Anda dan kemudian menambahkannya ke aturan in-bound Anda untuk port TDS dan port PostgreSQL. Untuk informasi selengkapnya, lihat Menambahkan aturan ke grup keamanan di Panduan EC2 Pengguna Amazon.
- Konfigurasi SSL tidak cocok - Jika parameter `rds.force_ssl` diaktifkan (diatur ke 1) pada Aurora PostgreSQL, klien harus terhubung ke Babelfish melalui SSL. Jika klien Anda tidak diatur dengan benar, Anda akan melihat pesan kesalahan seperti berikut ini:

  ```
  Cannot connect to your-Babelfish-DB-cluster, 1433
  --------------------
  ADDITIONAL INFORMATION:
  no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
  "database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
  ...
  ```

  Kesalahan ini menunjukkan kemungkinan masalah konfigurasi SSL antara klien lokal Anda dan klaster Babelfish DB, dan klaster mengharuskan klien untuk menggunakan SSL (parameter `rds.force_ssl` disetel ke 1). Untuk informasi selengkapnya tentang mengonfigurasi SSL, lihat Menggunakan SSL dengan instans DB PostgreSQL di Panduan Pengguna Amazon RDS.

  Jika Anda menggunakan SQL Server Management Studio (SSMS) untuk terhubung ke Babelfish dan Anda melihat kesalahan ini, Anda dapat memilih opsi koneksi Enkripsi koneksi dan Percayai

sertifikat server pada panel Properti Koneksi dan coba lagi. Pengaturan ini menangani persyaratan koneksi SSL untuk SSMS.

Untuk informasi selengkapnya tentang pemecahan masalah koneksi Aurora, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

# Menonaktifkan Babelfish

Saat Anda tidak lagi membutuhkan Babelfish, Anda dapat menonaktifkan fungsionalitas Babelfish.

Ketahui beberapa pertimbangannya:

- Dalam beberapa kasus, Anda mungkin menonaktifkan Babelfish sebelum menyelesaikan migrasi ke Aurora PostgreSQL. Jika Anda melakukannya dan DDL Anda bergantung pada tipe data SQL Server atau Anda menggunakan fungsionalitas T-SQL apa pun dalam kode Anda, kode Anda akan gagal.
- Jika setelah menyediakan instans Babelfish Anda menonaktifkan ekstensi Babelfish, Anda tidak dapat menyediakan basis data yang sama lagi di klaster yang sama.

Untuk menonaktifkan Babelfish, ubah grup parameter Anda, atur `rds.babelfish_status` ke `OFF`. Anda dapat terus menggunakan tipe data SQL Server Anda dalam keadaan Babelfish nonaktif, dengan mengatur `rds.babelfish_status` ke `datatypesonly`.

Jika Anda menonaktifkan Babelfish di grup parameter, semua klaster yang menggunakan grup parameter tersebut kehilangan fungsionalitas Babelfish.

Untuk informasi selengkapnya tentang memodifikasi grup parameter, lihat . Untuk informasi tentang parameter khusus Babelfish, lihat Pengaturan grup parameter klaster DB untuk Babelfish.

# Mengelola Babelfish untuk pembaruan versi Aurora Postgre SQL

Babelfish adalah opsi yang tersedia dengan Aurora Postgre SQL versi 13.4 dan rilis yang lebih tinggi. Pembaruan untuk Babelfish tersedia dengan rilis baru tertentu dari mesin database Aurora Postgre. SQL Untuk informasi lebih lanjut, lihat Catatan Rilis untuk Aurora Postgre. SQL

> **Note**
>
> Cluster Babelfish DB yang berjalan pada versi Aurora Postgre 13 tidak SQL dapat ditingkatkan ke Aurora Postgre 14.3, 14.4, dan 14.5. SQL Selain itu, Babelfish tidak mendukung peningkatan langsung dari 13.x ke 15.x. Anda harus terlebih dahulu meningkatkan klaster DB 13.x Anda ke 14.6 dan versi yang lebih tinggi lalu meningkatkan ke versi 15.x.

Untuk daftar fungsionalitas yang didukung di berbagai rilis Babelfish, lihat Fungsionalitas yang didukung di Babelfish berdasarkan versi.

Untuk daftar fungsionalitas yang saat ini tidak didukung, lihat Fungsionalitas yang tidak didukung di Babelfish.

Anda bisa mendapatkan daftar SQL versi Aurora Postgre yang mendukung Babelfish dengan menanyakan Wilayah AWS describe-db-engine-versions AWS CLI perintah, sebagai berikut.

Untuk Linux, macOS, atau Unix:

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
```

Untuk informasi selengkapnya, silakan lihat describe-db-engine-versionsdalam AWS CLI Command Reference.

Dalam topik berikut, Anda dapat mempelajari cara mengidentifikasi versi Babelfish yang berjalan di cluster Aurora Postgre SQL DB Anda, dan cara meningkatkan ke versi baru.

Daftar Isi


- Mengidentifikasi versi Babelfish Anda
- Meningkatkan klaster Babelfish Anda ke versi baru

- Meningkatkan Babelfish ke versi minor baru
- Meningkatkan Babelfish ke versi mayor baru
  - Sebelum meningkatkan Babelfish ke versi mayor baru
  - Melakukan peningkatan versi mayor
  - Setelah meningkatkan ke versi mayor baru
  - Contoh: Meningkatkan klaster DB Babelfish ke rilis mayor
- Menggunakan parameter versi produk Babelfish
  - Mengonfigurasi parameter versi produk Babelfish
  - Kueri dan parameter yang terpengaruh
  - Antarmuka dengan parameter babelfishpg_tsql.version


## Mengidentifikasi versi Babelfish Anda

Anda dapat meminta Babelfish untuk menemukan detail tentang versi Babelfish, versi Aurora Postgre, dan SQL versi Microsoft Server yang kompatibel. SQL Anda dapat menggunakan TDS port atau port PostgreSQL.


- To use the TDS port to query for version information
- To use the PostgreSQL port to query for version information


Untuk menggunakan TDS port untuk menanyakan informasi versi

1. Gunakan `sqlcmd` atau `ssms` untuk terhubung ke titik akhir untuk klaster DB Babelfish Anda.

   ```
   sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
       login-id -P password -d db_name
   ```

2. Untuk mengidentifikasi versi Babelfish, jalankan kueri berikut ini:

   ```
   1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
   2> GO
   ```

   Kueri ini mengembalikan hasil yang serupa dengan berikut ini:

   ```
   serverproperty
   ------------------------------
   ```

```
3.4.0

(1 rows affected)
```

3. Untuk mengidentifikasi versi cluster Aurora Postgre SQL DB, jalankan kueri berikut:

```
1> SELECT aurora_version() AS aurora_version
2> GO
```

Kueri ini mengembalikan hasil yang serupa dengan berikut ini:

```
aurora_version


-----------------------------------------------
15.5.0

(1 rows affected)
```

4. Untuk mengidentifikasi versi Microsoft SQL Server yang kompatibel, jalankan kueri berikut:

```
1> SELECT @@VERSION AS version
2> GO
```

Kueri ini mengembalikan hasil yang serupa dengan berikut ini:

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
Dec 7 2023 09:43:06
Copyright (c) Amazon Web Services
PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

(1 rows affected)
```

Sebagai contoh yang menunjukkan satu perbedaan kecil antara Babelfish dan Microsoft SQL Server, Anda dapat menjalankan kueri berikut. Pada Babelfish, kueri kembali1, sementara di Microsoft SQL Server, kueri kembali. NULL

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

Anda juga dapat menggunakan SQL port Postgre untuk mendapatkan informasi versi, seperti yang ditunjukkan dalam prosedur berikut.

Untuk menggunakan SQL port Postgre untuk meminta informasi versi

1. Gunakan `psql` atau `pgAdmin` untuk terhubung ke titik akhir untuk klaster DB Babelfish Anda.

```
psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
     port=5432 dbname=babelfish_db user=sa
```

2. Aktifkan fitur tambahan (`\x`) dari `psql` untuk output yang lebih mudah dibaca.

```
babelfish_db=> \x
babelfish_db=> SELECT
babelfish_db=> aurora_version() AS aurora_version,
babelfish_db=> version() AS postgresql_version,
babelfish_db=> sys.version() AS Babelfish_compatibility,
babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
```

   Kueri ini menghasilkan output serupa dengan berikut ini:

```
-[ RECORD 1 ]-----------
+-----------------------------------------------------------------------------------------------
aurora_version          | 15.5.0
postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
 x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
 Compatibility - 12.0.2000.8                    +
                         | Dec 7 2023 09:43:06
                                    +
                         | Copyright (c) Amazon Web Services
                                     +
                         | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
babelfish_version       | 3.4.0
```

## Meningkatkan klaster Babelfish Anda ke versi baru

Versi baru Babelfish menjadi tersedia dengan beberapa rilis baru dari mesin database Aurora Postgre SQL setelah versi 13.4. Setiap rilis baru Babelfish memiliki nomor versi sendiri. Seperti Aurora

PostgreSQL, Babelfish menggunakan. *major* *minor*. *patch*skema penamaan untuk versi. Misalnya, rilis Babelfish pertama, Babelfish versi 1.0.0, tersedia sebagai bagian dari Aurora Postgre 13.4.0. SQL

Babelfish tidak memerlukan proses instalasi terpisah. Seperti yang dibahas dalamMembuat klaster DB Babelfish for Aurora PostgreSQL, Aktifkan Babelfish adalah opsi yang Anda pilih saat membuat cluster Aurora Postgre DB. SQL

Demikian pula, Anda tidak dapat meningkatkan Babelfish secara independen dari klaster Aurora DB yang mendukung. Untuk memutakhirkan cluster Babelfish untuk Aurora Postgre SQL DB yang ada ke versi baru Babelfish, Anda meningkatkan cluster Aurora SQL Postgre DB ke versi baru yang mendukung versi Babelfish yang ingin Anda gunakan. Prosedur yang Anda ikuti untuk peningkatan tergantung pada versi Aurora Postgre SQL yang mendukung penyebaran Babelfish Anda, sebagai berikut.

Peningkatan versi mayor

Anda harus memutakhirkan versi Aurora Postgre berikut SQL ke Aurora Postgre SQL 14.6 dan versi yang lebih tinggi sebelum meningkatkan ke versi Aurora Postgre 15.2. SQL

- Aurora Postgre SQL 13.8 dan semua versi yang lebih tinggi
- Aurora Postgre SQL 13.7.1 dan semua versi minor yang lebih tinggi
- Aurora Postgre SQL 13.6.4 dan semua versi minor yang lebih tinggi

Anda dapat meningkatkan Aurora Postgre SQL 14.6 dan versi yang lebih tinggi ke Aurora Postgre 15.2 dan versi yang lebih tinggi. SQL

Upgrade cluster Aurora SQL Postgre DB ke versi utama baru melibatkan beberapa tugas awal. Untuk informasi selengkapnya, lihat Melakukan upgrade versi utama. Agar berhasil meningkatkan Babelfish Anda untuk klaster DB Aurora Postgre, Anda perlu membuat grup parameter cluster SQL DB khusus untuk versi Aurora Postgre yang baru. SQL Grup parameter baru ini harus berisi pengaturan parameter Babelfish yang sama dengan klaster yang Anda tingkatkan. Untuk informasi selengkapnya dan untuk tabel sumber dan peningkatan pemutakhiran versi mayor, lihat Meningkatkan Babelfish ke versi mayor baru.

Peningkatan dan patch versi minor

Versi dan patch minor tidak memerlukan pembuatan grup parameter klaster DB baru untuk peningkatan. Versi dan patch minor dapat menggunakan proses peningkatan versi minor, baik itu diterapkan secara otomatis atau manual. Untuk informasi selengkapnya dan tabel sumber dan target versi, lihat Meningkatkan Babelfish ke versi minor baru.

> **Note**
>
> Sebelum melakukan upgrade mayor atau minor, terapkan semua tugas pemeliharaan yang tertunda ke cluster Babelfish untuk Aurora Postgre Anda. SQL

Topik

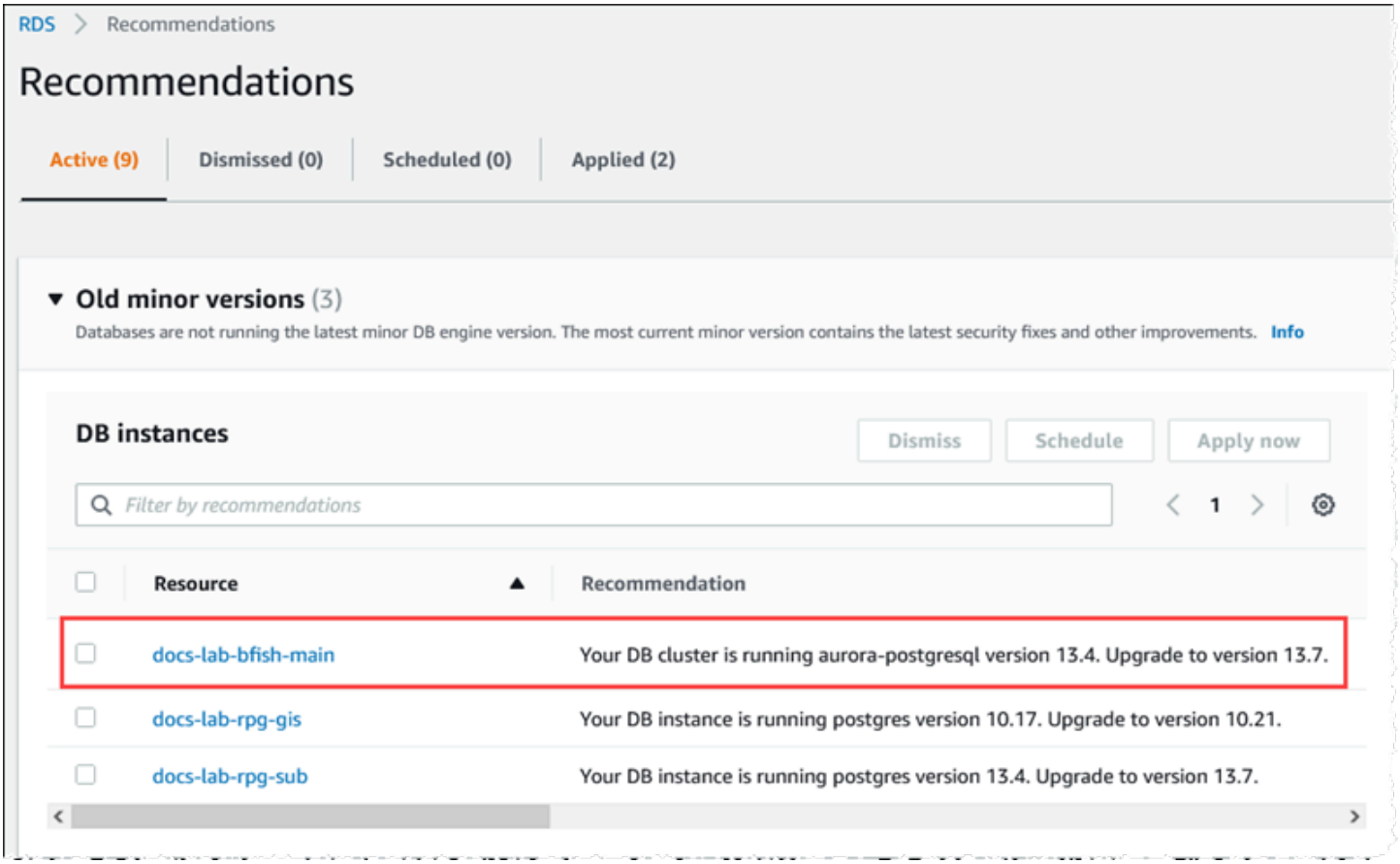
- Meningkatkan Babelfish ke versi minor baru
- Meningkatkan Babelfish ke versi mayor baru

## Meningkatkan Babelfish ke versi minor baru

Pembaruan versi minor bertujuan untuk mempertahankan kompatibilitas mundur. Namun, dalam beberapa kasus, perbaikan keamanan kritis atau perbaikan bug penting mungkin tidak sepenuhnya kompatibel ke belakang. Versi patch mencakup perbaikan penting untuk versi minor setelah dirilis. Misalnya, label versi untuk rilis pertama Aurora PostgreSQL 13.4 adalah Aurora PostgreSQL 13.4.0. Beberapa patch untuk versi minor telah dirilis hingga saat ini, termasuk Aurora PostgreSQL 13.4.1, 13.4.2, dan 13.4.4. Anda dapat menemukan patch yang tersedia untuk setiap versi Aurora PostgreSQL dalam daftar rilis Patch di bagian atas catatan rilis Aurora PostgreSQL untuk versi tersebut. Untuk mengetahui contohnya, lihat PostgreSQL 14.3 di Catatan Rilis untuk Aurora PostgreSQL.

Jika klaster DB Aurora PostgreSQL Anda dikonfigurasi dengan opsi Peningkatan versi minor otomatis, klaster DB Babelfish for Aurora PostgreSQL Anda ditingkatkan secara otomatis selama jendela pemeliharaan klaster. Untuk mempelajari lebih lanjut tentang peningkatan versi minor otomatis (AmVU) dan cara menggunakannya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora. Jika klaster Anda tidak menggunakan AmVU, Anda dapat meningkatkan klaster DB Babelfish for Aurora PostgreSQL secara manual ke versi minor baru baik dengan merespons tugas pemeliharaan, atau mengubah klaster untuk menggunakan versi baru.

Ketika Anda memilih versi Aurora PostgreSQL untuk diinstal dan ketika Anda melihat cluster Aurora PostgreSQL DB yang ada di, versi akan menampilkan. AWS Management Console*major* *minor*digit saja. Misalnya, gambar berikut dari Konsol untuk klaster DB Babelfish for Aurora PostgreSQL dengan Aurora PostgreSQL 13.4 merekomendasikan untuk meningkatkan klaster ke versi 13.7, rilis minor baru dari Aurora PostgreSQL.




Untuk mendapatkan detail versi lengkap, termasuk *patch* levelnya, Anda dapat menanyakan klaster Aurora PostgreSQL DB menggunakan fungsi Aurora PostgreSQL. `aurora_version` Untuk informasi selengkapnya, lihat aurora_version di Referensi fungsi Aurora PostgreSQL. Anda dapat menemukan contoh penggunaan fungsi dalam prosedur To use the PostgreSQL port to query for version information di Mengidentifikasi versi Babelfish Anda.

Tabel berikut menunjukkan versi Aurora PostgreSQL dan Babelfish dan versi target yang tersedia yang dapat mendukung proses peningkatan versi minor.

| Versi sumber saat ini | Target peningkatan terbaru |
|---|---|
| Aurora PostgreSQL (Ikan Babelfish) | Aurora PostgreSQL (Ikan Babelfish) |
| 17.4 (5,1) | 17.5 (5.2) |
| 16,8 (4,5) | 16.9 (4.6) |
| 16.6 (4.4.0) | 16.9 (4.6), 16.8 (4.5.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| 16.4 (4.3.0) | 16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0) |
| 16.3 (4.2.0) | 16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0) |
| 16.2 (4.1.0) | 16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0) |
| 16.1 (4.0.0) | 16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0) |
| 15.12 (3.9.0) | 15.13 (3.10) |
| 15.10 (3.8.0) | 15.13 (3.10), 15.12 (3.9.0) |
| 15.8 (3.7.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0) |
| 15.7 (3.6.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0) |
| 15.6 (3.5.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0) |
| 15.5 (3.4.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0) |
| 15.4 (3.3.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0) |
| 15.12 (3.9.0), 15.3 (3.2.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0) |
| 15.2 (3.1.0) | 15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0) |
| 14.17 (2.12.0) | 14.18 (2.13.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| 14.15 (2.11.0) | 14.18 (2.13.0), 14.17 (2.12.0) |
| 14.13 (2.10.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0) |
| 14.12 (2.9.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0) |
| 14.11 (2.8.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0) |
| 14.10 (2.7.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0) |
| 14.9 (2.6.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0) |
| 14.8 (2.5.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0) |
| 14.7 (2.4.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0), 14.8 (2.5.0) |
| 14.6 (2.3.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0), 14.8 (2.5.0), 14.7 (2.4.0) |
| 14.5 (2.2.0) | 14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0), 14.8 (2.5.0), 14.7 (2.4.0), 14.6 (2.3.0) |
| 14.3 (2.1.0) | 14.18 (2.13.0), 14.6 (2.3.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| 13.8 (1.4.0) | 13.9 (1.5.0) |
| 13.7 (1.3.0) | 13.9 (1.5.0), 13.8 (1.4.0) |

Meningkatkan Babelfish ke versi mayor baru

Untuk peningkatan versi mayor, Anda harus terlebih dahulu meningkatkan klaster DB Babelfish for Aurora PostgreSQL ke versi yang mendukung peningkatan versi mayor. Untuk mewujudkannya, terapkan pembaruan patch atau peningkatan versi minor ke klaster DB Anda. Untuk informasi selengkapnya, lihat Meningkatkan Babelfish ke versi minor baru.

Tabel berikut menunjukkan versi Aurora PostgreSQL dan versi Babelfish yang dapat mendukung peningkatan versi mayor.

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| Aurora PostgreSQL (Ikan Babelfish) | Aurora PostgreSQL (Ikan Babelfish) |
| 16.9 (4.6.0) | 17.5 (5.2.0) |
| 16.8 (4.5.0) | 17.5 (5.2.0), 17.4 (5.1.0) |
| 16.6 (4.4.0) | 17.5 (5.2.0), 17.4 (5.1.0) |
| 16.4 (4.3.0) | 17.5 (5.2.0), 17.4 (5.1.0) |
| 16.3 (4.2.0) | 17.5 (5.2.0), 17.4 (5.1.0) |
| 16.2 (4.1.0) | 17.5 (5.2.0), 17.4 (5.1.0) |
| 16.1 (4.0.0) | 17.5 (5.2.0), 17.4 (5.1.0) |
| 15.13 (3.10) | 17.5 (5.2.0)<br>16.9 (4.6.0) |
| 15.12 (3.9.0) | 17.5 (5.2.0), 17.4 (5.1.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| | 16.9 (4.6.0), 16.8 (4.5.0) |
| 15.10 (3.8.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0) |
| 15.8 (3.7.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0) |
| 15.7 (3.6.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0) |
| 15.6 (3.5.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0) |
| 15.5 (3.4.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) |
| 15.4 (3.3.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) |
| 15.3 (3.2.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) |
| 15.2 (3.1.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| 14.18 (2.13.0) | 17.5 (5.2.0)<br>16.9 (4.6.0)<br>15.13 (3.10.0) |
| 14.17 (2.12.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br>16.9 (4.6.0), 16.8 (4.5.0)<br>15.13 (3.10.0), 15.12 (3.9.0) |
| 14.15 (2.11.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0)<br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0) |
| 14.13 (2.10.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0)<br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0) |
| 14.12 (2.9.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0)<br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| 14.11 (2.8.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0)<br><br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0) |
| 14.10 (2.7.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)<br><br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0) |
| 14.9 (2.6.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)<br><br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0) |
| 14.8 (2.5.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)<br><br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0) |

| Versi sumber saat ini | Target peningkatan terbaru |
| --- | --- |
| 14.7 (2.4.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)<br><br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0), 15.2 (3.1.0) |
| 14.6 (2.3.0) | 17.5 (5.2.0), 17.4 (5.1.0)<br><br>16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)<br><br>15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0), 15.2 (3.1.0) |
| 13.9 (1.5.0) | 14.6 (2.3.0) |
| 13.8 (1.4.0) | 14.6 (2.3.0) |
| 13.7 (1.3.0) | 14.6 (2.3.0) |

Sebelum meningkatkan Babelfish ke versi mayor baru

Peningkatan mungkin melibatkan pemadaman singkat. Oleh karena itu, sebaiknya Anda melakukan atau menjadwalkan peningkatan selama periode pemeliharaan atau selama periode penggunaan rendah lainnya.

Sebelum melakukan peningkatan versi mayor

1. Identifikasi versi Babelfish dari klaster DB Aurora PostgreSQL Anda yang ada dengan menggunakan perintah yang diuraikan di Mengidentifikasi versi Babelfish Anda. Versi Aurora PostgreSQL dan informasi versi Babelfish ditangani oleh PostgreSQL, jadi ikuti langkah-langkah terperinci dalam prosedur To use the PostgreSQL port to query for version information untuk mengetahui detailnya.

2. Verifikasi apakah versi Anda mendukung peningkatan versi mayor. Untuk daftar versi yang mendukung fitur peningkatan versi mayor, lihat Meningkatkan Babelfish ke versi minor baru dan lakukan tugas prapeningkatan yang diperlukan.

   Misalnya, jika versi Babelfish Anda berjalan pada klaster DB Aurora PostgreSQL 13.5 dan Anda ingin meningkatkan ke Aurora PostgreSQL 15.2, maka pertama-tama terapkan semua rilis dan patch minor untuk meningkatkan klaster Anda ke Aurora PostgreSQL 14.6 atau versi yang lebih tinggi. Saat klaster Anda berada di versi 14.6 atau lebih tinggi, lanjutkan dengan proses peningkatan versi mayor.

3. Buat snapshot manual dari klaster DB Babelfish Anda saat ini sebagai cadangan. Cadangan ini memungkinkan Anda memulihkan klaster ke versi Aurora PostgreSQL, versi Babelfish, dan memulihkan semua data ke status sebelum peningkatan. Untuk informasi selengkapnya, lihat Membuat snapshot klaster DB. Pastikan untuk menyimpan grup parameter klaster DB kustom Anda yang ada untuk menggunakannya lagi jika Anda memutuskan untuk memulihkan klaster ini ke status prapeningkatan. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB dan Pertimbangan grup parameter.

4. Siapkan grup parameter klaster DB kustom untuk versi Aurora PostgreSQL target. Duplikasi pengaturan untuk parameter Babelfish dari klaster DB Babelfish for Aurora PostgreSQL Anda saat ini. Untuk menemukan daftar semua parameter Babelfish, lihat Pengaturan grup parameter klaster DB untuk Babelfish. Untuk peningkatan versi mayor, parameter berikut memerlukan pengaturan yang sama dengan klaster DB sumber. Agar peningkatan berhasil, semua pengaturan harus sama.

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale
   - babelfishpg_tsql.database_name
   - babelfishpg_tsql.default_locale
   - babelfishpg_tsql.migration_mode
   - babelfishpg_tsql.server_collation_name

   > **Warning**
   >
   > Jika pengaturan untuk parameter Babelfish dalam grup parameter klaster DB kustom untuk versi PostgreSQL Aurora baru tidak cocok dengan nilai parameter

klaster yang Anda tingkatkan, operasi `ModifyDBCluster` gagal. Pesan `InvalidParameterCombination` kesalahan muncul di AWS Management Console atau di output dari `modify-db-cluster` AWS CLI perintah.

5. Gunakan AWS Management Console or AWS CLI untuk membuat grup parameter cluster DB kustom. Pilih keluarga Aurora PostgreSQL yang berlaku untuk versi Aurora PostgreSQL yang Anda inginkan untuk peningkatan.

> **Tip**
>
> Kelompok parameter dikelola di Wilayah AWS level tersebut. Ketika Anda bekerja dengan AWS CLI, Anda dapat mengkonfigurasi dengan Region default alih-alih menentukan `--region` dalam perintah. Untuk mempelajari lebih lanjut tentang menggunakan AWS CLI, lihat Penyiapan cepat di Panduan AWS Command Line Interface Pengguna.

Melakukan peningkatan versi mayor

1. Tingkatkan klaster DB Aurora PostgreSQL ke versi mayor baru. Untuk informasi selengkapnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.
2. Boot ulang instans penulis klaster, sehingga pengaturan parameter dapat berlaku.

Setelah meningkatkan ke versi mayor baru

Setelah peningkatan versi mayor ke versi Aurora PostgreSQL baru, nilai `IDENTITY` dalam tabel dengan kolom `IDENTITY` mungkin lebih besar (+32) daripada nilai sebelum peningkatan. Hasilnya adalah ketika baris berikutnya dimasukkan ke dalam tabel tersebut, nilai kolom identitas yang dibuat melompat ke angka +32 dan memulai urutan dari sana. Kondisi ini tidak akan berdampak negatif pada fungsi klaster DB Babelfish Anda. Namun, jika mau, Anda dapat mengatur ulang objek urutan berdasarkan nilai maksimum kolom. Untuk melakukannya, sambungkan ke port T-SQL pada instans penulis Babelfish Anda menggunakan `sqlcmd` atau klien SQL Server lainnya. Untuk informasi selengkapnya, lihat Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda.

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
     sa -P ******** -d dbname
```

Saat terhubung, gunakan perintah SQL berikut untuk menghasilkan pernyataan yang dapat Anda gunakan untuk melakukan seeding terhadap objek urutan terkait. Perintah SQL ini berfungsi untuk konfigurasi Babelfish basis data tunggal dan beberapa basis data. Untuk informasi selengkapnya tentang kedua model deployment ini, lihat Menggunakan Babelfish dengan satu atau beberapa basis data.

```
DECLARE @schema_prefix NVARCHAR(200) = ''
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
    WHERE columns.is_identity = 1
GO
```

Kueri menghasilkan serangkaian pernyataan SELECT yang kemudian dapat Anda jalankan untuk mengatur ulang nilai IDENTITY maksimum dan menutup celah apa pun. Berikut ini menunjukkan output saat menggunakan basis data SQL Server sampel, Northwind, berjalan pada klaster Babelfish.

```
------------------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));

SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));
```

```
SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));


(5 rows affected)
```

Jalankan pernyataan satu per satu untuk mengatur ulang nilai urutan.

Contoh: Meningkatkan klaster DB Babelfish ke rilis mayor

Dalam contoh ini, Anda dapat menemukan serangkaian AWS CLI perintah yang menjelaskan cara memutakhirkan cluster Aurora PostgreSQL 13.6.4 DB yang menjalankan Babelfish versi 1.2.2 ke Aurora PostgreSQL 14.6. Pertama, Anda membuat grup parameter klaster DB kustom untuk Aurora PostgreSQL 14. Selanjutnya, Anda memodifikasi nilai parameter agar sesuai dengan sumber Aurora PostgreSQL versi 13 Anda. Terakhir, Anda melakukan peningkatan dengan memodifikasi klaster sumber. Untuk informasi selengkapnya, lihat Pengaturan grup parameter klaster DB untuk Babelfish. Dalam topik itu, Anda juga dapat menemukan informasi tentang menggunakan AWS Management Console untuk melakukan upgrade.

Gunakan perintah create-db-cluster-parameter-group CLI untuk membuat grup parameter cluster DB untuk versi baru.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

Saat Anda mengeluarkan perintah ini, grup parameter klaster DB kustom dibuat di Wilayah AWS. Output Anda akan terlihat seperti berikut ini.

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
```

```
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

Untuk informasi selengkapnya, lihat Membuat grup parameter cluster DB di Amazon Aurora.

Gunakan perintah modify-db-cluster-parameter-group CLI untuk memodifikasi pengaturan sehingga cocok dengan cluster sumber.

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

Responsnya terlihat seperti berikut.

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

Gunakan perintah modify-db-clusterCLI untuk memodifikasi cluster untuk menggunakan versi baru dan grup parameter cluster DB kustom baru. Anda juga menentukan argumen `--allow-major-version-upgrade`, seperti yang ditunjukkan dalam sampel berikut.

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

Gunakan perintah reboot-db-instanceCLI untuk me-reboot instance penulis cluster, sehingga pengaturan parameter dapat berlaku.

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

## Menggunakan parameter versi produk Babelfish

Parameter Grand Unified Configuration (GUC) baru yang `babelfishpg_tds.product_version` disebut diperkenalkan dari versi Babelfish 2.4.0 dan 3.1.0. Parameter ini memungkinkan Anda untuk mengatur nomor versi produk SQL Server sebagai output dari Babelfish.

Parameternya adalah string ID versi 4 bagian, dan setiap bagian harus dipisahkan dengan “.”.

Sintaks

```
Major.Minor.Build.Revision
```

- Versi Utama: Angka antara 11 dan 16.
- Versi Minor: Angka antara 0 dan 255.
- Versi Build: Angka antara 0 dan 65535.
- Revisi: 0 dan angka positif berapa pun.

### Mengonfigurasi parameter versi produk Babelfish

Anda harus menggunakan grup parameter klaster untuk mengatur parameter babelfishpg_tds.product_version di konsol. Untuk informasi selengkapnya tentang cara memodifikasi

parameter cluster DB, lihatMemodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.

Saat Anda menyetel parameter versi produk ke nilai yang tidak valid, perubahan tidak akan berlaku. Meskipun konsol mungkin menunjukkan nilai baru, parameter akan mempertahankan nilai sebelumnya. Periksa file log mesin untuk detail tentang pesan kesalahan.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Kueri dan parameter yang terpengaruh

| Kueri/Parameter | Hasil | Waktu efektif |
|---|---|---|
| SELECT @@VERSION | Mengembalikan versi SQL Server yang ditentukan pengguna (nilai babelfish pg_tsql.version = Default) | Langsung |
| SELECT SERVERPROPERTY('ProductVersion') | Mengembalikan versi SQL Server yang ditentukan pengguna | Langsung |
| SELECT SERVERPROPERTY('ProductMajorVersion') | Mengembalikan Versi Utama dari versi SQL Server yang ditentukan pengguna | Langsung |

| Kueri/Parameter | Hasil | Waktu efektif |
| --- | --- | --- |
| VERSIONtoken dalam Pesan PRELOGIN Respons | Server mengembalikan PRELOGIN pesan dengan versi SQL Server yang ditentukan pengguna | Berlaku ketika pengguna membuat sesi baru |
| SQLServerVersiondi LoginAck saat menggunakan JDBC | DatabaseMetaData. getDatabaseProductVersion () mengembalikan versi SQL Server yang ditentukan pengguna | Berlaku ketika pengguna membuat sesi baru |

Antarmuka dengan parameter babelfishpg_tsql.version

Anda dapat mengatur output @@ VERSION menggunakan parameter babelfishpg_tsql.version dan babelfishpg_tds.product_version. Contoh berikut menunjukkan bagaimana antarmuka dua parameter ini.

- Ketika parameter babelfishpg_tsql.version adalah "default" dan babelfishpg_tds.product_version adalah 15.0.2000.8.
  - Output dari @@version – 15.0.2000.8.
- Ketika parameter babelfishpg_tsql.version disetel ke 13.0.2000.8 dan parameter babelfishpg_tds.product_version adalah 15.0.2000.8.
  - Output dari @@version – 13.0.2000.8.

# Referensi Babelfish for Aurora PostgreSQL

Topik


- Fungsionalitas yang didukung di Babelfish berdasarkan versi
- Fungsionalitas yang tidak didukung di Babelfish
- Bekerja dengan Babelfish untuk prosedur Aurora Postgre SQL


## Fungsionalitas yang didukung di Babelfish berdasarkan versi

Dalam tabel berikut, Anda dapat menemukan daftar semua fungsi yang didukung oleh versi Babelfish yang berbeda. Untuk daftar fungsionalitas yang tidak didukung, lihat Fungsionalitas yang tidak didukung di Babelfish. Untuk informasi tentang berbagai rilis Babelfish, lihat Catatan Rilis untuk Aurora PostgreSQL.

Fitur Aurora dan PostgreSQL

Dalam tabel berikut, Anda dapat menemukan fitur Aurora dan PostgreSQL yang didukung oleh versi Babelfish yang berbeda.

| Fitur Aurora dan PostgreSQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Layanan Aurora ML | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Otentikasi database dengan Kerberos menggunakan AWS Directory Service | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| Buang dan pulihkan | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| ekstensi pg_stat_statement | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |

| Fitur Aurora dan PostgreSQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pgvector | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Zero-downtime patching (ZDP) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |

Fungsionalitas atau sintaks T-SQL

Dalam tabel berikut, Anda dapat menemukan fungsionalitas atau sintaks T-SQL yang didukung oleh versi Babelfish yang berbeda.

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Referensi nama objek 4 bagian untuk pernyataan SELECT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| MENGUBAH sintaks OTORISASI untuk mengubah pemilik database | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| *<db_name>* UBAH DATABASE MEMODIFIKASI NAMA = *<new_db_name>* | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| ALTER FUNCTION | Y | Y | Y | Y | Y | T | T | T | T | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MENGUBAH PROSEDUR | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| MENGUBAH PERAN | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| UBAH PENGGUNA... DENGAN LOGIN | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| ALTER VIEW | Y | T | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Kata kunci AS di CREATE FUNCTION | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Klausul DI ZONA WAKTU | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Contoh Babelfish sebagai server tertaut | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| Operator perbandingan! < dan! > | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| BUAT INDEKS... ON partition_scheme_name (partition_column_name) | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CREATE Alih-alih Triggers (DHTML) pada Tampilan SQL Server | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| BUAT FUNGSI PARTISI | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| BUAT SKEMA PARTISI | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| CREATE ROLE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| BUAT TABEL... ON partition_scheme_name (partition_column_name) | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| BUAT PEMICU | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| MEMBUAT ATAU MENGUBAH TAMPILAN | Y | T | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Buat indeks unik | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Eksekusi prosedur lintas basis data | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Referensi silang — database PILIH, SISIPKAN, PERBARUI, HAPUS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Parameter yang diketik kursor hanya untuk parameter input (bukan output) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Migrasi data menggunakan utilitas klien bcp | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Tipe data TIMESTAMP, ROWVERSION (untuk informasi penggunaan, lihat Fitur dengan implementasi terbatas | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Kata kunci DEFAULT dalam panggilan ke prosedur dan fungsi yang disimpan | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| DBCC CHECKIDENT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| DROP DATABASE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DROP IF EXISTS (untuk objek SCHEMA, DATABASE, dan USER) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Indeks DROP INDEX ON schema.table | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| DROP INDEX schema.table.index | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| FUNGSI PARTISI DROP | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| SKEMA PARTISI DROP | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| DROP ROLE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| AKTIFKAN/ NONAKTIFKAN PEMICU | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| PENCARIAN TEKS LENGKAP | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| Pencarian Teks Lengkap dengan klausa CONTAINS | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Jenis data spasial Geometri dan Geografi | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| HIBAH | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IZIN HIBAH [,... n] PADA SKEMA | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| GUC babelfish pg_tds.product_ver sion | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| Pengidentifikasi dengan karakter titik terkemuka | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| BUKAN pemicu pada tabel | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ALIH-ALIH pemicu pada tampilan | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| KILL | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| $partition.partiti on_function_name (partition_column_ value) | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PIVOT (gunakan dalam definisi tampilan dari 4.4.0 dan 3.8.0, ekspresi tabel umum dari 3.4.0, bergabung dari 3.4.0) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| MENCABUT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| CABUT izin [,... n] PADA SKEMA | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| PILIH... OFFSET... AMBIL klausa | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| PILIH UNTUK JSON AUTO | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| SET BABELFISH _SHOWPLAN_ALL ON (dan OFF) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| SETEL PROFIL BABELFISH _STATISTICS PADA (NONAKTIF) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| MENGATUR CONTEXT_INFO | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ATUR LOCK_TIMEOUT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| SET NO_BROWSETABLE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| SET rowcount | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| SET SHOWPLAN_ALL | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| SET STATISTICS IO | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| SET TRANSACTION ISOLATION LEVEL REPEATABLE READ | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| SET TRANSACTION ISOLATION LEVEL SERIALIZABLE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| SET SINTAKS TINGKAT ISOLASI TRANSAKSI | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| SSMS: Menghubungkan dengan dialog koneksi penjelajah objek | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
| SSMS: Migrasi data dengan Wizard Import/Export | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| SSMS: Dukungan sebagian untuk penjelajah objek | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| STDEV | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | Tidak |
| STDEVP | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | Tidak |
| Pemicu dengan beberapa tindakan DML dapat mereferensikan tabel transisi | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Petunjuk T-SQL (metode gabungan, penggunaan indeks, MAXDOP) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| Sintaks braket persegi T-SQL dengan predikat LIKE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | Tidak |
| UNPIVOT | Y | T | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |

| Fungsionalitas atau sintaks T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nilai string yang tidak dikutip dalam panggilan prosedur tersimpan dan nilai default | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| VAR | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| VARP | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |

Fungsi bawaan T-SQL

Dalam tabel berikut, Anda dapat menemukan fungsi T-SQL Built-in yang didukung oleh versi Babelfish yang berbeda.

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| APP_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| ATN2 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| CHARINDEX | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| PILIH | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COL_LENGTH | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| COL_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| COLUMNS_UPDATED | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| COLUMNPROPERTY (CharMaxLen, AllowsNull hanya) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| CONCAT_WS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| CONTEXT_INFO | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| KURSOR_STATUS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| DATABASE_PRINCIPAL_ID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| DATEADD | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 201.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DATEDIFF | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| DATEDIFF_BIG | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| DATEFROMPARTS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| NAMA DATA | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| DATEPART | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| DATETIMEF ROMPARTS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| DATETIME2 DARIBAGIAN | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| DATETIMEO FFSETFROM PARTS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| DATETRUNC | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| DATE_BUCKET | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EOMONTH | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak | |
| JALANKAN SEBAGAI PENELEPON | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak | | |
| fn_listextendedproperty | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak | |
| UNTUK JSON | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak | | |
| FULLTEXTSERVICEPROPERTY | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | | |
| HAS_DBACCESS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | | |
| HAS_PERMS_BY_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | | |
| HOST_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak | | |
| HOST_ID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak | |
| IDENTITAS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak | |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IS_ANGGOTA | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IS_ROLEMEMBER | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IS_SRVROLEMEMBER | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ISJSON | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| JSON_MODIFY | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| JSON_QUERY | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| JSON_NILAI | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| NILAI BERIKUTNYA UNTUK | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| OBJECT_DEFINITION | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| OBJECT_SCHEMA_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OPENJSON | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| OPENQUERY | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ORIGINAL_LOGIN | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| PARSENAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | Tidak |
| PATINDEX | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ROWCOUNT_BESAR | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | Tidak |
| SCHEMA_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| SESSION_CONTEXT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | Tidak |
| SESSION_USER | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| SID_BINARY (selalu mengembalikan NULL) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SMALLDATE TIMEFROMPARTS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | Tidak | |
| PERSEGI | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | |
| STR | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak | |
| STRING_AGG | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | T | T | T | T | T | T | T | T | Tidak | |
| STRING_SPLIT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | |
| SUSER_SID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | |
| SUSER_SNAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya | |
| SWITCHOFFSET | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak | |
| SYSTEM_USER | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak | |
| WAKTUDARI BAGIAN | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | Tidak | |

| Fungsi bawaan T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 21.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TODATETIMEOFFSET | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| TO_CHAR | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| TRIGGER_NESTLEVEL (tanpa argumen saja) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| TRY_CONVERT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| TYPE_ID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| TYPE_NAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| UPDATE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

Katalog T-SQL INFORMATION_SCHEMA

Dalam tabel berikut, Anda dapat menemukan katalog T-SQL INFORMATION_SCHEMA yang didukung oleh versi Babelfish yang berbeda.

| Katalog T-SQL INFORMATION_SCHEMA | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CHECK_KENDALA | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| COLUMN_DOMAIN_USAGE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| KOLOM | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| CONSTRAINT_COLUMN_USAGE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| WILAYAH | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| KEY_COLUMN_USAGE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| RUTINITAS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| TABEL | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| TABLE_KENDALA | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| PANDANGAN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| Katalog T-SQL INFORMATI ON_SCHEMA | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 21.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

Variabel @@ yang ditentukan sistem T-SQL

Dalam tabel berikut, Anda dapat menemukan variabel @@ yang ditentukan Sistem T-SQL yang didukung oleh versi Babelfish yang berbeda.

| Variabel @@ yang ditentukan sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 21.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| @ @CURSOR_R OWS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @DATEFIRST | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @DBTS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @ERROR | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @ERROR =213 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @FETCH_ST ATUS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @IDENTITY | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| Variabel @@ yang ditentukan sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 201.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @LANGUAGE | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @LOCK_TIMEOUT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @MAX_CONNECTIONS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @MAX_PRECISION | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @MICROSOFTVERSION | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @NESTLEVEL | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @PROCID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @ROWCOUNT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @SERVERNAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Variabel @@ yang ditentukan sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 201.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| @ @SERVICENAME | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @SPID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @TRANCOUNT | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| @ @VERSION (perbedaan format catatan seperti yang dijelaskan dalam[Perbedaan T-SQL di Babelfish]. | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

Prosedur tersimpan Sistem T-SQL

Dalam tabel berikut, Anda dapat menemukan prosedur tersimpan Sistem T-SQL yang didukung oleh versi Babelfish yang berbeda.

| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 201.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sp_addextendedproperti | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| sp_addlinkedserver | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| sp_addlinkedsrvlogin | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
| | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| sp_addrole | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| sp_addrolemember | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| sp_babelfish_volatility | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| sp_column_privileges | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_kolom | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_columns_100 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_columns_managed | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_kursor | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursor_list | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sp_cursorclose | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursorexecute | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursorfetch | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursorbuka | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursoroption | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursorprepare | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursorprepexec | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_cursorunprepare | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_database | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_datatype_info | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sp_datatype_info_100 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_describe_kursor | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_describe_first_result_set | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_describe_undeclared_parameters | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_dropextendedproperti | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| sp_droplinkedsrvlogin | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| sp_droprole | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| sp_droprolemember | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| sp_dropserver | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| sp_enum_oledb_providers | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
| sp_mengeksekusi | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_execute_postgresql (BUAT, UBAH, JATUHKAN) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak | |
| sp_executesql | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_fkeys | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_getapplock | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_helpdb | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_helpdbfixedrole | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| sp_helplinkedsrvlogin | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| sp_helprole | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_helprolemember | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sp_helpsrvrolemember | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| sp_helpuser | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_linkedservers | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| sp_oledb_ro_usrname | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_pkeys | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_awalan | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| sp_persiapkan | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_procedure re_params _100_managed | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sp_releaseapplock | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_ganti nama | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
| sp_berganti nama | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sp_rename (untuk object_type 'DATABASE') | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sp_reset_connection | Y | Y | Y | Y | Y | T | T | T | T | T | Y | Y | Y | T | T | T | T | T | T | T | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | Tidak |
| sp_serveroption (opsi connect_timeout) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| sp_set_session_context | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| sp_special_columns | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_sproc_columns | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_sproc_columns_100 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_statistik | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Prosedur tersimpan Sistem T-SQL | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sp_statistik_100 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_stored_procedures | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_table_privileges | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_tablecollations_100 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_tables | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_testlinkedserver | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| sp_unprepare | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sp_updateextendedproperty | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| sp_siapa | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| xp_qv | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

T-SQL Properties didukung pada fungsi sistem CONNECTIONPROPERTY

Dalam tabel berikut, Anda dapat menemukan T-SQL Properties didukung pada fungsi sistem CONNECTIONPROPERTY yang didukung oleh versi Babelfish yang berbeda.

| T-SQL Properties didukung pada fungsi sistem CONNECTIONPROPERTY | 5.5.0 | 5.4.0 | 4.4.0 | 4.4.0 | 4.4.0 | 4.4.0 | 4.4.0 | 4.4.0 | 4.4.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 3.3.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.2.0 | 2.1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| auth_scheme | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| client_net_address | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| local_net_address | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| local_tcp_port | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| net_transport | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| protocol_type | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| physical_net_transport | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

T-SQL Properties didukung pada fungsi sistem OBJECTPROPERTY

Dalam tabel berikut, Anda dapat menemukan T-SQL Properties didukung pada fungsi sistem OBJECTPROPERTY yang didukung oleh versi Babelfish yang berbeda.

| T-SQL Properties didukung pada fungsi sistem OBJECTPROPERTY | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 21.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IsInlineFunction | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsScalarFunction | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsTableFunction | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

Properti T-SQL didukung pada fungsi SERVERPROPERTY

Dalam tabel berikut, Anda dapat menemukan T-SQL Properties didukung pada fungsi SERVERPROPERTY yang didukung oleh versi Babelfish yang berbeda.

| Properti T-SQL didukung pada fungsi SERVERPROPERTY | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 21.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Babelfish | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Kolasi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| Properti T-SQL didukung pada fungsi SERVERPROPERTY | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| CollationId | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Edisi | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| EditionID | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| EngineEdition | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| InstanceName | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |
| IsAdvancedAnalyticsInstalled | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsBigDataCluster | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsFullTextInstalled | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsIntegratedSecurityOnly | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Properti T-SQL didukung pada fungsi SERVERPROPERTY | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IsLocalDB | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsPolyBaseInstalled | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| IsSingleUser | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Adalah XTPSupported | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Jepang_CI_AI | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Japanese_CI_AS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| Japanese_CS_AS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| LicenseType | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| MachineName | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Tidak |

| Properti T-SQL didukung pada fungsi SERVERPROPERTY | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ProductLevel | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| ProductMajorVersion | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ProductMinorVersion | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ProductUpdateLevel | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Tidak |
| ProductVersion | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| ServerName | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

Tampilan SQL Server didukung oleh Babelfish

Dalam tabel berikut, Anda dapat menemukan tampilan SQL Server yang didukung oleh versi Babelfish yang berbeda.

| Tampilan SQL Server didukung oleh Babelfish | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| information_schema.key_column_usage | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| Tampilan SQL Server didukung oleh Babelfish | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | Tidak |
| information_schema .routines | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| information_schema .schemata | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| information_schema .sequences | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| sys.all_columns | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.all_objects | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.all_parameters | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |
| sys.all_sql_modules | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.all_views | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.columns | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tampilan SQL Server didukung oleh Babelfish | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
| sys.configurations | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.data_spaces | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.database_file | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.database_mirroring | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.database_principal | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.database_role_member | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.database | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.destination_data_spaces | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.dm_exec_connections | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Tampilan SQL Server didukung oleh Babelfish | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sys.dm_exec_sessions | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.dm_hadr_database_replica_states | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.dm_os_host_info | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.dm_os_sys_info | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.endpoints | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.extended_properties | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | Tidak |
| sys.indexes | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.partisi | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.partition_functions | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |

| Tampilan SQL Server didukung oleh Babelfish | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sys.partition_parameters | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.partition_range_values | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.partition_schemes | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.skema | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.server_principals | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.server_role_member | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | Tidak |
| sys.sql_modules | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.sysconfigurres | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.syscurconfigs | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |

| Tampilan SQL Server didukung oleh Babelfish | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 21.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sys.syslogins | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| sys.sysprocesses | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.sysuser | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Y | Y | Y | Y | Y | Y | Y | T | T | T | T | T | Tidak |
| sys.table_types | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.tables | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sys.jenis | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | T | Tidak |
| sys.xml_schema_collections | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| syslanguages | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Ya |
| sysobjects.crdate | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | T | Tidak |

## Fungsionalitas yang tidak didukung di Babelfish

Dalam tabel dan daftar berikut, Anda dapat menemukan fungsionalitas yang saat ini tidak didukung di Babelfish. Pembaruan untuk Babelfish termasuk dalam versi Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Catatan rilis Aurora PostgreSQL.

Topik


- Fungsionalitas yang saat ini tidak didukung
- Pengaturan yang tidak didukung
- Perintah yang tidak didukung
- Nama kolom atau atribut yang tidak didukung
- Tipe data yang tidak didukung
- Tipe objek yang tidak didukung
- Fungsi yang tidak didukung
- Sintaks yang tidak didukung


### Fungsionalitas yang saat ini tidak didukung

Dalam tabel, Anda dapat menemukan informasi tentang fungsionalitas tertentu yang saat ini tidak didukung.

| Fungsionalitas atau sintaks | Deskripsi |
|---|---|
| Modul perakitan dan rutinitas SQL Common Language Runtime (CLR) | Fungsionalitas yang terkait dengan modul perakitan dan rutinitas CLR tidak didukung. |
| Atribut kolom | ROWGUIDCOL, SPARSE, FILESTREAM, dan MASKED tidak didukung. |
| Basis data bermuatan | Basis data bermuatan dengan login yang diautentikasi di tingkat basis data alih-alih di tingkat server tidak didukung. |
| DDL lintas basis data | Melakukan pernyataan DDL yang mereferensikan atau beroperasi pada objek di beberapa database belum didukung. |
| Kursor (dapat diperbarui) | Kursor yang dapat diperbarui tidak didukung. |

| Fungsionalitas atau sintaks | Deskripsi |
| --- | --- |
| Kursor (global) | Kursor GLOBAL tidak didukung. |
| Kursor (perilaku pengambilan) | Perilaku pengambilan kursor berikut tidak didukung: FETCH PRIOR, FIRST, LAST, ABSOLUTE, abd RELATIVE |
| Parameter output tipe kursor | Variabel dan parameter tipe kursor tidak didukung untuk parameter output (kesalahan muncul). |
| Opsi kursor | SCROLL, KEYSET, DYNAMIC, FAST_FORWARD, SCROLL_LOCKS, OPTIMISTIC, TYPE_WARNING, dan FOR UPDATE |
| Enkripsi data | Enkripsi data tidak didukung. |
| Aplikasi tingkat data (DAC) | Operasi impor atau ekspor aplikasi tingkat data (DAC) dengan paket DAC (.dacpac) atau file cadangan DAC (.bacpac) tidak didukung. |
| Perintah DBCC | Perintah Konsol Basis Data (DBCC) Microsoft SQL Server tidak didukung. DBCC CHECKIDENT didukung di Babelfish 3.4.0 dan rilis yang lebih tinggi. |
| DROP IF EXISTS | Sintaks ini tidak didukung untuk objek USER dan SCHEMA. Ini didukung untuk objek TABLE, VIEW, PROCEDURE, FUNCTION, dan DATABASE. |
| Enkripsi | Fungsi dan pernyataan default tidak mendukung enkripsi. |
| Koneksi ENCRYPT_CLIENT_CERT | Koneksi sertifikat klien tidak didukung. |
| Pernyataan EXECUTE AS | Pernyataan ini tidak didukung. |
| Klausa EXECUTE AS SELF | Klausa ini tidak didukung dalam fungsi, prosedur, atau pemicu. |
| Klausa EXECUTE AS USER | Klausa ini tidak didukung dalam fungsi, prosedur, atau pemicu. |

| Fungsionalitas atau sintaks | Deskripsi |
| --- | --- |
| Batasan kunci asing yang mereferensikan nama basis data | Batasan kunci asing yang mereferensikan nama basis data tidak didukung. |
| FORMAT | Tipe yang ditentukan pengguna tidak didukung. |
| Deklarasi fungsi dengan lebih dari 100 parameter | Deklarasi fungsi yang berisi lebih dari 100 parameter tidak didukung. |
| Panggilan fungsi yang menyertakan DEFAULT sebagai nilai parameter | DEFAULT bukan nilai parameter yang didukung untuk panggilan fungsi. DEFAULT sebagai nilai parameter untuk panggilan fungsi didukung dari Babelfish 3.4.0 dan rilis yang lebih tinggi. |
| Fungsi, didefinisikan secara eksternal | Fungsi eksternal, termasuk fungsi SQL CLR, tidak didukung. |
| Tabel sementara global (tabel dengan nama yang dimulai dengan ##) | Tabel temporer global tidak didukung. |
| Fungsionalitas grafik | Semua fungsionalitas grafik SQL tidak didukung. |
| Prosedur tersimpan Umum yang Diperpanjang | Prosedur tersimpan sistem yang menyediakan antarmuka dari instance SQL Server ke program eksternal, karena berbagai aktivitas pemeliharaan tidak didukung. Ini termasuk `xp_cmdshell` dan prosedur tersimpan sistem lainnya. Untuk informasi selengkapnya, lihat Prosedur tersimpan Umum yang Diperpanjang. |
| Pengidentifikasi (variabel atau parameter) dengan beberapa karakter @ utama | Pengidentifikasi yang dimulai dengan lebih dari satu `@` utama tidak didukung. |
| Pengidentifikasi, nama tabel atau kolom yang berisi karakter @ atau ]] | Nama tabel atau kolom yang berisi tanda `@` atau tanda kurung siku tidak didukung. |

| Fungsionalitas atau sintaks | Deskripsi |
| --- | --- |
| Indeks inline | Indeks inline tidak didukung. |
| Menginvokasi prosedur yang namanya ada dalam variabel | Menggunakan variabel sebagai nama prosedur tidak didukung. |
| Tampilan terwujud | Tampilan terwujud tidak didukung. |
| Klausa NOT FOR REPLICATION | Sintaks ini diterima dan diabaikan. |
| Fungsi escape ODBC | Fungsi escape ODBC tidak didukung. |
| Panggilan prosedur yang menyertakan DEFAULT sebagai nilai parameter | DEFAULT bukan nilai parameter yang didukung. DEFAULT sebagai nilai parameter untuk panggilan fungsi didukung dari Babelfish 3.4.0 dan rilis yang lebih tinggi. |
| Deklarasi prosedur dengan lebih dari 100 parameter | Deklarasi yang berisi lebih dari 100 parameter tidak didukung. |
| Prosedur, didefinisikan secara eksternal | Prosedur yang ditentukan secara eksternal, termasuk prosedur SQL CLR, tidak didukung. |
| Versioning prosedur | Versioning prosedur tidak didukung. |
| Prosedur WITH RECOMPILE | WITH RECOMPILE (saat digunakan bersama dengan pernyataan DECLARE dan EXECUTE) tidak didukung. |
| Referensi objek jarak jauh | Menjalankan prosedur tersimpan terhadap server tertaut Babelfish tidak didukung. Nama objek empat bagian hanya berfungsi untuk membaca dan tidak berfungsi untuk memodifikasi tabel jarak jauh. UPDATE dapat mereferensikan tabel jarak jauh dalam klausa FROM tanpa memodifikasinya. Untuk informasi selengkapnya, lihat Babelfish mendukung server tertaut. |
| Keamanan tingkat baris | Keamanan tingkat baris dengan CREATE SECURITY POLICY dan fungsi bernilai tabel inline tidak didukung. |

| Fungsionalitas atau sintaks | Deskripsi |
| --- | --- |
| Fungsi broker layanan | Fungsi broker layanan tidak didukung. |
| PROPERTI SESI | Properti yang tidak didukung: ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, CONCAT_NULL_YIELDS _NULL, dan NUMERIC_ROUNDABORT |
| SET LANGUAGE | Sintaks ini tidak didukung dengan nilai apa pun selain `english` atauus_`english` . |
| SP_CONFIGURE | Prosedur tersimpan sistem ini tidak didukung. |
| Kata kunci SPARSE SQL | Kata kunci SPARSE diterima dan diabaikan. |
| Sintaks konstruktor nilai tabel (klausa FROM) | Sintaks yang tidak didukung adalah untuk tabel turunan yang dibangun dengan klausa FROM. |
| Tabel temporal | Tabel temporal tidak didukung. |
| Prosedur sementara tidak dijatuhkan secara otomatis | Fungsionalitas ini tidak didukung. |
| Pemicu, didefinisikan secara eksternal | Pemicu ini tidak didukung, termasuk SQL Common Language Runtime (CLR). |
| Tanpa klausa SCHEMABIN DING | Membuat tampilan tanpa SCHEMABINDING tidak didukung, tetapi tampilan dibuat seolah-olah WITH SCHEMABINDING ditentukan. Menggunakan SCHEMABINDING saat membuat fungsi, prosedur, pemicu diabaikan secara diam-diam. |

Pengaturan yang tidak didukung

Pengaturan berikut tidak didukung:

- SET ANSI_NULL_DFLT_OFF ON
- SET ANSI_NULL_DFLT_ON OFF
- SET ANSI_PADDING OFF
- SET ANSI_WARNINGS OFF

- SET ARITHABORT OFF
- SET ARITHIGNORE ON
- SET CURSOR_CLOSE_ON_COMMIT ON
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON (perintah tidak berfungsi seperti yang diharapkan)
- SET FMTONLY ON (perintah tidak berfungsi seperti yang diharapkan. Ini hanya menekan eksekusi pernyataan SELECT tetapi tidak yang lain.)

Perintah yang tidak didukung

Fungsionalitas tertentu untuk perintah berikut tidak didukung:

- ADD SIGNATURE
- ALTER DATABASE, ALTER DATABASE SET
- BACKUP/RESTORE DATABASE/LOG
- BACPAC and DACPAC FILES RESTORE
- BUAT, UBAH, JATUHKAN OTORISASI. OTORISASI ALTER didukung untuk objek database.
- CREATE, ALTER, DROP AVAILABILITY GROUP
- CREATE, ALTER, DROP BROKER PRIORITY
- CREATE, ALTER, DROP COLUMN ENCRYPTION KEY
- CREATE, ALTER, DROP DATABASE ENCRYPTION KEY
- CREATE, ALTER, DROP, BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- CHECKPOINT

Nama kolom atau atribut yang tidak didukung

Nama kolom berikut tidak didukung:

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

### Tipe data yang tidak didukung

Tipe data berikut tidak didukung:

- HIERARCHYID

### Tipe objek yang tidak didukung

Tipe objek berikut tidak didukung:

- COLUMN MASTER KEY
- CREATE, ALTER EXTERNAL DATA SOURCE
- CREATE, ALTER, DROP DATABASE AUDIT SPECIFICATION
- CREATE, ALTER, DROP EXTERNAL LIBRARY
- CREATE, ALTER, DROP SERVER AUDIT
- CREATE, ALTER, DROP SERVER AUDIT SPECIFICATION
- BUAT, UBAH, JATUHKAN, KUNCI OPEN/CLOSE SIMETRIS
- CREATE, DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION
- Tampilan yang diindeks
- SERVICE MASTER KEY
- SYNONYM

### Fungsi yang tidak didukung

Fungsi default berikut tidak didukung:

#### Fungsi agregat

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- STRING_AGG menggunakan klausa WITHIN GROUP

Fungsi kriptografi

- Fungsi CERTENCODED
- Fungsi CERTID
- Fungsi CERTPROPERTY

Fungsi metadata

- COLUMNPROPERTY
- TYPEPROPERTY
- Fungsi SERVERPROPERTY - Properti berikut tidak didukung:
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS
  - HadrManagerStatus
  - InstanceDefaultDataPath
  - InstanceDefaultLogPath
  - IsClustered
  - IsHadrEnabled
  - LCID
  - NumLicenses
  - ProcessID
  - ProductBuild
  - ProductBuildType
  - ProductUpdateReference
  - ResourceLastUpdateDateTime
  - ResourceVersion
  - ServerName
  - SqlCharSet
  - SqlCharSetName
  - SqlSortOrder
  - SqlSortOrderName

- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

Fungsi keamanan

- CERTPRIVATEKEY
- LOGINPROPERTY

Pernyataan, operator, fungsi lainnya

- Fungsi EVENTDATA
- GET_TRANSMISSION_STATUS
- OPENXML

Sintaks yang tidak didukung

Sintaks berikut tidak didukung:

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER INDEX
- MENGUBAH FUNGSI PARTISI
- MENGUBAH SKEMA PARTISI
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- UBAH LAYANAN, BACKUP/RESTORE Klausul KUNCI MASTER LAYANAN
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT

- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE, ALTER, DROP APPLICATION ROLE
- CREATE, ALTER, DROP ASSEMBLY
- CREATE, ALTER, DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE, ALTER, DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- CREATE, ALTER, DROP EXTERNAL RESOURCE POOL
- CREATE, ALTER, DROP FULLTEXT CATALOG
- CREATE, ALTER, DROP FULLTEXT INDEX
- CREATE, ALTER, DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- BUAT, UBAH, JATUHKAN, KUNCI OPEN/CLOSE, BACKUP/RESTORE MASTER
- CREATE, ALTER, DROP QUEUE
- CREATE, ALTER, DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP RESOURCE POOL
- CREATE, ALTER, DROP ROUTE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- Klausa CREATE, ALTER, DROP SELECTIVE XML INDEX
- CREATE, ALTER, DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE

- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE, ALTER, DROP WORKLOAD GROUP
- MENGUBAH PEMICU
- CREATE TABLE... Klausa GRANT
- CREATE TABLE... Klausa IDENTITY
- CREATE USER – Sintaks ini tidak didukung. Pernyataan PostgreSQL CREATE USER tidak membuat pengguna yang setara dengan sintaks CREATE USER SQL Server. Untuk informasi selengkapnya, lihat Perbedaan T-SQL di Babelfish.
- DENY
- END, MOVE CONVERSATION
- EXECUTE dengan opsi AS LOGIN atau AT
- GET CONVERSATION GROUP
- Klausa GROUP BY ALL
- Klausa GROUP BY CUBE
- Klausa GROUP BY ROLLUP
- INSERT... DEFAULT VALUES
- MERGE
- TEKS BACA
- REVERT
- SELECT TOP x PERCENT WHERE x <> 100
- SELECT TOP... WITH TIES
- SELECT... FOR BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SELECT... UNTUK JALUR XML.
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON

- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE
- MENGATUR WAKTU STATISTIK
- SET STATISTICS XML
- Pernyataan SHUTDOWN
- UPDATE STATISTICS
- UPDATETEXT
- Menggunakan EXECUTE untuk memanggil fungsi SQL
- VIEW... Klausa CHECK OPTION
- VIEW... Klausa VIEW_METADATA
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- Konsep WITH XMLNAMESPACES
- WRITETEXT
- Ekspresi XPATH

# Bekerja dengan Babelfish untuk prosedur Aurora Postgre SQL

## Gambaran Umum

Anda dapat menggunakan prosedur berikut untuk instans Amazon RDS DB yang menjalankan Babelfish for Aurora SQL Postgre untuk kinerja kueri yang lebih baik:

- sp_babelfish_volatility
- sp_execute_postgresql

## sp_babelfish_volatility

Volatilitas SQL fungsi Postgre membantu pengoptimal untuk eksekusi kueri yang lebih baik yang bila digunakan di bagian klausa tertentu memiliki dampak signifikan pada kinerja kueri.

### Sintaks

```
sp_babelfish_volatility ‘function_name’, ‘volatility’
```

### Argumen

function_name (opsional)

Anda dapat menentukan nilai argumen ini dengan nama dua bagian sebagai `schema_name.function_name` atau hanya `function_name`. Jika hanya menentukan `function_name`, nama skemanya adalah skema default untuk pengguna saat ini.

volatilitas (opsional)

SQLNilai volatilitas Postgre yang valid adalah`stable`,, `volatile` atau. `immutable` Untuk informasi selengkapnya, silakan lihat https://www.postgresql.org/docs/current/xfunc-volatility.html

> **Note**
>
> Saat `sp_babelfish_volatility` dipanggil dengan `function_name` yang memiliki beberapa definisi, hal tersebut akan menimbulkan kesalahan.

Set hasil

Jika parameter tidak disebutkan, maka set hasil ditampilkan di bawah kolom berikut: `schemaname`, `functionname`, `volatility`.

Catatan penggunaan

Volatilitas SQL fungsi Postgre membantu pengoptimal untuk eksekusi kueri yang lebih baik yang bila digunakan di bagian klausa tertentu memiliki dampak signifikan pada kinerja kueri.

Contoh

Contoh berikut menunjukkan cara membuat fungsi sederhana, lalu menjelaskan cara menggunakan `sp_babelfish_volatility` pada fungsi-fungsi ini menggunakan metode yang berbeda.

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

Contoh berikut menunjukkan volatilitas fungsi tersebut:

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

Contoh berikut menunjukkan cara mengubah volatilitas fungsi tersebut:

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
```

```
2> go
```

Saat Anda hanya menentukan function_name, hal tersebut menampilkan nama skema, nama fungsi, dan volatilitas fungsi tersebut. Contoh berikut menampilkan volatilitas fungsi setelah mengubah nilai:

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

Saat Anda tidak menentukan argumen apa pun, daftar fungsi akan ditampilkan (nama skema, nama fungsi, volatilitas fungsi) yang ada dalam basis data saat ini:

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

Anda dapat mengeksekusi SQL pernyataan Postgre dari T- SQL endpoint. Ini menyederhanakan aplikasi Anda karena Anda tidak perlu keluar dari SQL T-port untuk mengeksekusi pernyataan ini.

Sintaks

```
sp_execute_postgresql [ @stmt = ] statement
```

Argumen

Pernyataan [ @stmt ]

Argumennya adalah dari datatype varchar. Argumen ini menerima pernyataan dialek PG.

> **Note**
>
> Anda hanya dapat meneruskan satu pernyataan dialek PG sebagai argumen, jika tidak, maka akan menimbulkan kesalahan berikut.

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```
Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

Catatan penggunaan

CREATE EXTENSION

Membuat dan memuat ekstensi baru ke dalam basis data saat ini.

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

Contoh berikut menunjukkan cara membuat ekstensi:

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
2>go
```

Gunakan perintah berikut untuk mengakses objek ekstensi:

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> Jika nama skema tidak diberikan secara eksplisit selama pembuatan ekstensi, secara default ekstensi diinstal di skema publik. Anda harus memberikan kualifikasi skema untuk mengakses objek ekstensi seperti yang disebutkan di bawah ini:

```
1>select * from [public].pg_stat_statements;
2>go
```

Ekstensi yang didukung

Ekstensi berikut yang tersedia dengan Aurora Postgre SQL berfungsi dengan Babelfish.

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

Batasan

- Pengguna harus memiliki peran sysadmin pada T- SQL dan rds_superuser di postgres untuk menginstal ekstensi.
- Ekstensi tidak dapat diinstal pada skema yang dibuat pengguna dan juga pada skema dbo dan tamu untuk basis data master, tempdb, dan msdb.
- CASCADEopsi tidak didukung.

ALTER EXTENSION

Anda dapat meningkatkan ke versi ekstensi baru menggunakan ALTER ekstensi.

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
2>go
```

Batasan

- Anda dapat memutakhirkan versi ekstensi Anda hanya menggunakan pernyataan ALTER Ekstensi. Opsi lain tidak didukung.

DROP EXTENSION

Membatalkan ekstensi yang ditentukan. Anda juga dapat menggunakan opsi `if exists` atau `restrict` untuk membatalkan ekstensi.

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

Batasan

- CASCADEopsi tidak didukung.

# Kinerja dan penskalaan untuk Amazon Aurora PostgreSQL

Bagian berikut membahas pengelolaan performa dan penskalaan untuk klaster DB Amazon Aurora PostgreSQL. Bagian tersebut juga berisi informasi tentang tugas pemeliharaan lainnya.

Topik

- Penskalaan instans DB Aurora PostgreSQL
- Koneksi maksimum ke instans DB Aurora PostgreSQL
- Batas penyimpanan sementara untuk Aurora PostgreSQL
- Halaman besar untuk Aurora PostgreSQL
- Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan
- Menampilkan status volume untuk klaster DB Aurora PostgreSQL
- Menentukan disk RAM untuk stats_temp_directory
- Mengelola file sementara dengan PostgreSQL

# Penskalaan instans DB Aurora PostgreSQL

Anda dapat menskalakan instans DB Aurora PostgreSQL dengan dua cara, penskalaan instans dan penskalaan baca. Untuk informasi selengkapnya tentang penskalaan baca, lihat Penskalaan baca.

Anda dapat menskalakan klaster DB Aurora PostgreSQL Anda dengan mengubah kelas instans DB untuk setiap instans DB dalam klaster DB. Aurora PostgreSQL mendukung beberapa kelas instans DB yang dioptimalkan untuk Aurora. Jangan gunakan kelas instans db.t2 atau db.t3 untuk klaster Aurora yang lebih besar dengan ukuran lebih dari 40 terabyte (TB).

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.

Penskalaan tidak terjadi secara instan. Diperlukan waktu 15 menit atau lebih untuk menyelesaikan perubahan ke kelas instans DB yang berbeda. Jika Anda menggunakan pendekatan ini untuk mengubah kelas instans DB, Anda perlu menerapkan perubahan selama periode pemeliharaan terjadwal berikutnya (bukan segera) agar tidak memengaruhi pengguna.

Sebagai alternatif untuk mengubah kelas instans DB secara langsung, Anda dapat meminimalkan waktu henti dengan menggunakan fitur ketersediaan tinggi Amazon Aurora. Pertama, tambahkan Replika Aurora ke klaster Anda. Saat membuat replika, pilih ukuran kelas instans DB yang ingin Anda gunakan untuk klaster Anda. Ketika Replika Aurora disinkronkan dengan klaster, maka Anda akan melakukan failover ke Replika yang baru ditambahkan. Untuk mempelajari selengkapnya, lihat Replika Aurora dan Failover cepat dengan Amazon Aurora PostgreSQL.

Untuk spesifikasi terperinci tentang kelas instans DB yang didukung oleh Aurora PostgreSQL, lihat Mesin DB yang didukung untuk kelas instans DB.

## Koneksi maksimum ke instans DB Aurora PostgreSQL

klaster DB Aurora PostgreSQL mengalokasikan sumber daya berdasarkan kelas instans DB dan memori yang tersedia. Setiap koneksi ke klaster DB mengonsumsi sejumlah tambahan sumber daya ini, seperti memori dan CPU. Memori yang dikonsumsi per koneksi bervariasi berdasarkan jenis kueri, jumlah, dan apakah tabel sementara digunakan. Bahkan koneksi idle menghabiskan memori

dan CPU. Hal ini karena ketika kueri dijalankan pada koneksi, lebih banyak memori dialokasikan untuk setiap kueri dan tidak dirilis sepenuhnya, bahkan ketika pemrosesan berhenti. Oleh karena itu, kami menyarankan agar Anda memastikan aplikasi Anda tidak mempertahankan koneksi idle: setiap koneksi idle akan memboroskan sumber daya dan memengaruhi performa secara negatif. Untuk informasi selengkapnya, lihat Sumber daya yang dikonsumsi oleh koneksi PostgreSQL idle.

Jumlah maksimum koneksi yang diizinkan oleh instans DB Aurora PostgreSQL akan ditentukan berdasarkan nilai parameter `max_connections` yang ditentukan dalam grup parameter untuk instans DB tersebut. Pengaturan ideal untuk `max_connections` parameter adalah yang mendukung semua koneksi klien yang dibutuhkan aplikasi Anda, tanpa kelebihan koneksi yang tidak digunakan, ditambah setidaknya 3 koneksi lagi untuk mendukung AWS otomatisasi. Sebelum mengubah pengaturan parameter `max_connections`, sebaiknya pertimbangkan hal berikut:

- Jika nilai `max_connections` terlalu rendah, instans DB Aurora PostgreSQL mungkin tidak memiliki koneksi yang cukup saat klien mencoba terhubung. Jika hal ini terjadi, coba hubungkan menggunakan pesan "raise error" `psql` seperti berikut ini:

  ```
  psql: FATAL: remaining connection slots are reserved for non-replication superuser
   connections
  ```

- Jika nilai `max_connections` melebihi jumlah koneksi yang benar-benar dibutuhkan, koneksi yang tidak terpakai dapat menyebabkan performa menurun.

Nilai default `max_connections` berasal dari fungsi `LEAST` Aurora PostgreSQL berikut:

`LEAST({DBInstanceClassMemory/9531392},5000)`.

Jika Anda ingin mengubah nilai `max_connections`, Anda perlu membuat grup parameter klaster DB kustom dan mengubah nilainya di sana. Setelah menerapkan grup parameter DB kustom Anda ke klaster Anda, pastikan untuk mem-boot ulang instans primer agar nilai baru mulai berlaku. Untuk informasi selengkapnya, lihat Parameter Amazon Aurora PostgreSQL dan Membuat grup parameter cluster DB di Amazon Aurora.

> **Tip**
>
> Jika aplikasi Anda sering membuka dan menutup koneksi, atau membiarkan sejumlah besar koneksi yang lama aktif tetap terbuka, kami menyarankan agar Anda menggunakan Proksi Amazon RDS. Proksi RDS adalah proksi basis data yang sepenuhnya terkelola dengan ketersediaan tinggi yang menggunakan pooling koneksi untuk berbagi koneksi basis data

dengan aman dan efisien. Untuk mempelajari selengkapnya tentang Proksi RDS, lihat Proksi Amazon RDS untuk Aurora.

Untuk detail tentang caranya Aurora Serverless v2 instance menangani parameter ini, lihatKoneksi maksimum untuk Aurora Serverless v2.

## Batas penyimpanan sementara untuk Aurora PostgreSQL

Aurora PostgreSQL menyimpan tabel dan indeks dalam subsistem penyimpanan Aurora. Aurora PostgreSQL menggunakan penyimpanan sementara terpisah untuk file sementara non-persisten. Hal ini termasuk file yang digunakan untuk tujuan seperti mengurutkan set data besar selama pemrosesan kueri atau untuk operasi pembuatan indeks. Untuk informasi selengkapnya, lihat artikel Bagaimana cara memecahkan masalah penyimpanan lokal di instans yang kompatibel dengan Aurora PostgreSQL?.

Volume penyimpanan lokal ini didukung oleh Amazon Elastic Block Store dan dapat diperluas dengan menggunakan kelas instans DB yang lebih besar. Untuk informasi selengkapnya tentang penyimpanan, lihat Penyimpanan Amazon Aurora. Anda juga dapat meningkatkan penyimpanan lokal untuk objek sementara dengan menggunakan jenis instans yang diaktifkan dan objek sementara yang NVMe diaktifkan oleh Aurora Optimized Reading. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads.

> **Note**
>
> Anda mungkin melihat peristiwa `storage-optimization` saat menskalakan instans DB, misalnya, dari db.r5.2xlarge ke db.r5.4xlarge.

Tabel berikut menunjukkan jumlah maksimum penyimpanan sementara yang tersedia untuk setiap kelas instans DB Aurora PostgreSQL. Untuk informasi selengkapnya tentang dukungan kelas instans DB untuk Aurora, lihat Kelas instans Amazon Aurora DB.

| Kelas instans DB | Penyimpanan sementara maksimum yang tersedia (GiB) |
|---|---|
| db.x2g.16xlarge | 1829 |

| Kelas instans DB | Penyimpanan sementara maksimum yang tersedia (GiB) |
|---|---|
| db.x2g.12xlarge | 1606 |
| db.x2g.8xlarge | 1071 |
| db.x2g.4xlarge | 535 |
| db.x2g.2xlarge | 268 |
| db.x2g.xlarge | 134 |
| db.x2g.large | 67 |
| db.r8g.48xlarge | 3072 |
| db.r8g.24xlarge | 1536 |
| db.r8g.16xlarge | 998 |
| db.r8g.12xlarge | 749 |
| db.r8g.8xlarge | 499 |
| db.r8g.4xlarge | 250 |
| db.r8g.2xlarge | 125 |
| db.r8g.xlarge | 63 |
| db.r8g.large | 31 |
| db.r7g.16xlarge | 1008 |
| db.r7g.12xlarge | 756 |
| db.r7g.8xlarge | 504 |
| db.r7g.4xlarge | 252 |
| db.r7g.2xlarge | 126 |

| Kelas instans DB | Penyimpanan sementara maksimum yang tersedia (GiB) |
|---|---|
| db.r7g.xlarge | 63 |
| db.r7g.large | 32 |
| db.r7i.48xlarge | 3072 |
| db.r7i.24xlarge | 1500 |
| db.r7i.16xlarge | 1008 |
| db.r7i.12xlarge | 748 |
| db.r7i.8xlarge | 504 |
| db.r7i.4xlarge | 249 |
| db.r7i.2xlarge | 124 |
| db.r7i.xlarge | 62 |
| db.r7i.large | 31 |
| db.r6g.16xlarge | 1008 |
| db.r6g.12xlarge | 756 |
| db.r6g.8xlarge | 504 |
| db.r6g.4xlarge | 252 |
| db.r6g.2xlarge | 126 |
| db.r6g.xlarge | 63 |
| db.r6g.large | 32 |
| db.r6i.32xlarge | 1829 |
| db.r6i.24xlarge | 1500 |

| Kelas instans DB | Penyimpanan sementara maksimum yang tersedia (GiB) |
|---|---|
| db.r6i.16xlarge | 1008 |
| db.r6i.12xlarge | 748 |
| db.r6i.8xlarge | 504 |
| db.r6i.4xlarge | 249 |
| db.r6i.2xlarge | 124 |
| db.r6i.xlarge | 62 |
| db.r6i.large | 31 |
| db.r5.24xlarge | 1500 |
| db.r5.16xlarge | 1008 |
| db.r5.12xlarge | 748 |
| db.r5.8xlarge | 504 |
| db.r5.4xlarge | 249 |
| db.r5.2xlarge | 124 |
| db.r5.xlarge | 62 |
| db.r5.large | 31 |
| db.r4.16xlarge | 960 |
| db.r4.8xlarge | 480 |
| db.r4.4xlarge | 240 |
| db.r4.2xlarge | 120 |
| db.r4.xlarge | 60 |

| Kelas instans DB | Penyimpanan sementara maksimum yang tersedia (GiB) |
|---|---|
| db.r4.large | 30 |
| db.t4g.large | 16.5 |
| db.t4g.medium | 8.13 |
| db.t3.large | 16 |
| db.t3.medium | 7.5 |

> **Note**
>
> NVMe jenis instance yang diaktifkan dapat meningkatkan ruang sementara yang tersedia hingga NVMe ukuran total. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads.

Anda dapat memantau penyimpanan sementara yang tersedia untuk instans DB dengan `FreeLocalStorage` CloudWatch metrik, -> dijelaskan dalam CloudWatch Metrik Amazon untuk Amazon Aurora. (Ini tidak berlaku untuk Aurora Serverless v2.)

Untuk beberapa beban kerja, Anda dapat mengurangi jumlah penyimpanan sementara dengan mengalokasikan lebih banyak memori ke proses yang melakukan operasi. Untuk meningkatkan ketersediaan memori bagi operasi, tingkatkan nilai parameter PostgreSQL work_mem atau maintenance_work_mem.

## Halaman besar untuk Aurora PostgreSQL

Halaman besar adalah fitur manajemen memori yang mengurangi overhead saat instans DB menangani potongan besar memori yang berdekatan, seperti yang digunakan oleh buffer bersama. Fitur PostgreSQL ini didukung oleh semua versi Aurora PostgreSQL yang tersedia saat ini.

Parameter `Huge_pages` diaktifkan secara default untuk semua kelas instans DB selain kelas instans t3.medium, db.t3.large, db.t4g.medium, db.t4g.large. Anda tidak dapat mengubah nilai parameter `huge_pages` atau menonaktifkan fitur ini di kelas instans Aurora PostgreSQL yang didukung.

Pada instance Aurora PostgreSQL DB yang tidak mendukung fitur memori halaman besar, penggunaan memori proses tertentu mungkin meningkat tanpa perubahan beban kerja yang sesuai.

Sistem mengalokasikan segmen memori bersama seperti cache buffer selama startup server. Ketika halaman memori besar tidak tersedia, sistem tidak membebankan alokasi ini ke proses postmaster. Sebaliknya, itu termasuk memori dalam proses yang pertama kali mengakses setiap halaman 4KB di segmen memori bersama.

> **Note**
>
> Koneksi aktif berbagi memori yang dialokasikan sesuai kebutuhan, terlepas dari bagaimana penggunaan memori bersama dilacak di seluruh proses.

## Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan

Anda dapat menguji toleransi kesalahan klaster DB Aurora PostgreSQL Anda menggunakan kueri injeksi kesalahan. Kueri injeksi kesalahan dikeluarkan sebagai perintah SQL ke instans Amazon Aurora. Kueri injeksi kesalahan memungkinkan Anda menyebabkan crash pada instans sehingga Anda dapat menguji failover dan pemulihan. Anda juga dapat menyimulasikan kegagalan Replika Aurora, kegagalan disk, dan kepadatan disk. Kueri injeksi kesalahan didukung oleh semua versi Aurora PostgreSQL yang tersedia, sebagai berikut.

- Aurora PostgreSQL versi 12, 13, 14, dan lebih tinggi
- Aurora PostgreSQL versi 11.7 dan lebih tinggi
- Aurora PostgreSQL versi 10.11 dan lebih tinggi

Topik


- Menguji crash instans
- Menguji kegagalan Replika Aurora
- Menguji kegagalan disk
- Menguji kepadatan disk


Ketika kueri injeksi kesalahan menentukan suatu crash, kueri ini akan menyebabkan crash pada instans DB Aurora PostgreSQL. Kueri injeksi kesalahan lainnya menghasilkan simulasi peristiwa

kegagalan, tetapi tidak menyebabkan peristiwa terjadi. Jika Anda mengirim kueri injeksi kesalahan, Anda juga menentukan durasi waktu untuk simulasi peristiwa kegagalan yang akan terjadi.

Anda dapat mengirim kueri injeksi kesalahan ke salah satu instans Replika Aurora dengan menghubungkan ke titik akhir untuk Replika Aurora. Untuk informasi selengkapnya, lihat Koneksi titik akhir Amazon Aurora.

## Menguji crash instans

Anda dapat memaksa crash instans Aurora PostgreSQL menggunakan fungsi kueri injeksi kesalahan `aurora_inject_crash()`.

Untuk kueri injeksi kesalahan ini, failover tidak akan terjadi. Jika Anda ingin menguji failover, Anda dapat memilih tindakan instance Failover untuk cluster DB Anda di konsol RDS, atau menggunakan failover-db-cluster AWS CLI perintah atau operasi Failover DBCluster RDS API.

Sintaksis

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

Opsi

Kueri injeksi kesalahan ini menggunakan salah satu dari jenis crash berikut. Jenis crash tidak peka huruf besar/kecil:

'instans'

Crash basis data yang kompatibel dengan PostgreSQL untuk instans Amazon Aurora disimulasikan.

'dispatcher'

Crash dispatcher pada instans primer untuk klaster DB Aurora disimulasikan. Dispatcher menulis pembaruan ke volume klaster untuk klaster DB Amazon Aurora.

'simpul'

Crash basis data yang kompatibel dengan PostgreSQL dan dispatcher untuk instans Amazon Aurora disimulasikan.

## Menguji kegagalan Replika Aurora

Anda dapat menyimulasikan kegagalan Replika Aurora menggunakan fungsi kueri injeksi kesalahan `aurora_inject_replica_failure()`.

Kegagalan Replika Aurora akan memblokir replikasi ke Replika Aurora atau semua Replika Aurora dalam klaster DB untuk interval waktu yang ditentukan. Ketika interval waktu selesai, Replika Aurora yang terdampak akan secara otomatis tersinkronisasi dengan instans primer.

### Sintaksis

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
);
```

### Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

percentage_of_failure

Persentase replikasi yang akan diblokir selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada replikasi yang akan diblokir. Jika Anda menetapkan 100, semua replikasi diblokir.

time_interval

Durasi waktu simulasi kegagalan Replika Aurora. Interval dalam detik. Misalnya, jika nilainya adalah 20, simulasi berjalan selama 20 detik.

> **Note**
>
> Berhati-hatilah saat menentukan interval waktu untuk peristiwa kegagalan Replika Aurora Anda. Jika Anda menentukan interval yang terlalu panjang, dan instans penulis Anda menulis data dalam jumlah besar selama peristiwa kegagalan, maka klaster DB Aurora Anda mungkin menganggap bahwa Replika Aurora telah crash lalu menggantinya.

replica_name

Replika Aurora tempat simulasi kegagalan akan diinjeksikan. Tentukan nama Replika Aurora untuk menyimulasikan kegagalan satu Replika Aurora. Tentukan string kosong untuk menyimulasikan kegagalan untuk semua Replika Aurora dalam klaster DB.

Untuk mengidentifikasi nama replika, lihat kolom `server_id` dari fungsi `aurora_replica_status()`. Sebagai contoh:

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

## Menguji kegagalan disk

Anda dapat menyimulasikan kegagalan disk untuk klaster DB Aurora PostgreSQL menggunakan fungsi kueri injeksi kesalahan `aurora_inject_disk_failure()`.

Selama simulasi kegagalan disk, klaster DB Aurora PostgreSQL menandai secara acak segmen disk sebagai mengalami kesalahan. Permintaan untuk segmen tersebut diblokir selama durasi simulasi.

Sintaksis

```
SELECT aurora_inject_disk_failure(
   percentage_of_failure,
   index,
   is_disk,
   time_interval
);
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

percentage_of_failure

Persentase disk yang akan ditandai sebagai mengalami kesalahan selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menentukan 0, tidak ada disk yang akan ditandai sebagai mengalami kesalahan. Jika Anda menentukan 100, seluruh disk akan ditandai sebagai mengalami kesalahan.

indeks

Blok data logis tertentu tempat peristiwa kegagalan akan disimulasikan. Jika Anda melampaui rentang blok logis atau data simpul penyimpanan yang tersedia, Anda akan menerima pesan kesalahan yang memberitahukan nilai indeks maksimum yang dapat Anda tentukan. Untuk menghindari kesalahan ini, lihat Menampilkan status volume untuk klaster DB Aurora PostgreSQL.

is_disk

Mengindikasikan apakah kegagalan injeksi terjadi pada blok logis atau simpul penyimpanan. Menentukan benar berarti kegagalan injeksi terjadi pada blok logis. Menentukan salah berarti kegagalan injeksi terjadi pada simpul penyimpanan.

time_interval

Jumlah waktu untuk mensimulasikan kegagalan disk. Interval dalam detik. Misalnya, jika nilainya adalah 20, simulasi berjalan selama 20 detik.

## Menguji kepadatan disk

Anda dapat mensimulasikan kemacetan disk untuk cluster Aurora PostgreSQL DB dengan menggunakan fungsi query injeksi kesalahan. `aurora_inject_disk_congestion()`

Selama simulasi kepadatan disk, klaster DB Aurora PostgreSQL secara acak menandai segmen disk sebagai padat. Permintaan untuk segmen tersebut ditunda antara waktu penundaan minimum dan maksimum yang ditentukan untuk durasi simulasi.

Sintaksis

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
   time_interval,
   minimum,
   maximum
);
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

percentage_of_failure

Persentase disk yang akan ditandai sebagai padat selama peristiwa kegagalan. Ini adalah nilai double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada disk yang ditandai sebagai padat. Jika Anda menetapkan 100, seluruh disk akan ditandai sebagai padat.

indeks

Blok logis data atau simpul penyimpanan tertentu yang akan digunakan dalam simulasi peristiwa kegagalan.

Jika Anda melampaui rentang blok logis atau simpul penyimpanan data yang tersedia, Anda akan menerima pesan kesalahan yang memberitahukan nilai indeks maksimum yang dapat Anda tentukan. Untuk menghindari kesalahan ini, lihat Menampilkan status volume untuk klaster DB Aurora PostgreSQL.

is_disk

Mengindikasikan apakah kegagalan injeksi terjadi pada blok logis atau simpul penyimpanan. Menentukan benar berarti kegagalan injeksi terjadi pada blok logis. Menentukan salah berarti kegagalan injeksi terjadi pada simpul penyimpanan.

time_interval

Jumlah waktu untuk mensimulasikan kemacetan disk. Interval dalam detik. Misalnya, jika nilainya adalah 20, simulasi berjalan selama 20 detik.

minimum, maximum

Penundaan kepadatan minimum dan maksimum, dalam milidetik. Nilai yang valid berkisar dari 0,0 hingga 100,0 milidetik. Segmen disk yang ditandai padat akan ditunda selama jangka waktu acak dalam rentang durasi minimum dan maksimum untuk simulasi. Nilai maksimum harus lebih besar dari nilai minimum.

## Menampilkan status volume untuk klaster DB Aurora PostgreSQL

Di Amazon Aurora, volume klaster DB terdiri dari kumpulan blok logis. Masing-masing blok ini merepresentasikan 10 gigabyte penyimpanan yang dialokasikan. Blok-blok ini disebut grup perlindungan.

Data di setiap grup perlindungan direplikasi di enam perangkat penyimpanan fisik, yang disebut simpul penyimpanan. Node penyimpanan ini dialokasikan di tiga Availability Zones (AZs) di wilayah tempat cluster DB berada. Pada gilirannya, setiap simpul penyimpanan berisi satu atau beberapa

blok logis data untuk volume klaster DB. Untuk informasi selengkapnya tentang grup perlindungan dan node penyimpanan, lihat Memperkenalkan mesin penyimpanan Aurora di Blog AWS Database. Untuk mempelajari selengkapnya tentang volume klaster Aurora secara umum, lihat Penyimpanan Amazon Aurora.

Gunakan fungsi `aurora_show_volume_status()` untuk menghasilkan variabel status server berikut:

- `Disks` — Jumlah total blok logis data untuk volume klaster DB.
- `Nodes` — Jumlah total simpul penyimpanan untuk volume klaster DB.

Anda dapat menggunakan fungsi `aurora_show_volume_status()` untuk membantu menghindari kesalahan saat menggunakan fungsi injeksi kesalahan `aurora_inject_disk_failure()`. Fungsi injeksi kesalahan `aurora_inject_disk_failure()` menyimulasikan kegagalan seluruh simpul penyimpanan atau satu blok logis data dalam simpul penyimpanan. Dalam fungsi tersebut, tentukan nilai indeks blok logis spesifik data atau simpul penyimpanan. Namun, pernyataan tersebut akan menghasilkan kesalahan jika Anda menentukan nilai indeks yang lebih besar dari jumlah blok logis data atau simpul penyimpanan yang digunakan oleh volume klaster DB. Untuk informasi selengkapnya tentang kueri injeksi kesalahan, lihat Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan.

> **Note**
>
> Fungsi `aurora_show_volume_status()` tersedia untuk Aurora PostgreSQL versi 10.11. Untuk informasi selengkapnya tentang versi Aurora PostgreSQL, lihat Amazon Aurora Postgre SQL rilis dan versi mesin.

Sintaksis

```
SELECT * FROM aurora_show_volume_status();
```

Contoh

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
    96 |    45
```

# Menentukan disk RAM untuk stats_temp_directory

Anda dapat menggunakan parameter Aurora PostgreSQL, `rds.pg_stat_ramdisk_size`, untuk menentukan memori sistem yang dialokasikan ke disk RAM untuk menyimpan `stats_temp_directory` PostgreSQL. Parameter disk RAM hanya tersedia di Aurora PostgreSQL 14 dan versi lebih rendah.

Di bawah beban kerja tertentu, pengaturan parameter ini dapat meningkatkan performa dan menurunkan kebutuhan IO. Untuk informasi selengkapnya tentang `stats_temp_directory`, lihat Run-time Statistics dalam dokumentasi PostgreSQL. Mulai PostgreSQL versi 15, komunitas PostgreSQL beralih menggunakan memori bersama dinamis. Jadi, `stats_temp_directory` tidak perlu diatur.

Untuk mengaktifkan disk RAM untuk `stats_temp_directory` Anda, setel parameter `rds.pg_stat_ramdisk_size` ke nilai bukan nol di grup parameter klaster DB yang digunakan oleh klaster DB Anda. Parameter ini menunjukkan MB, jadi Anda harus menggunakan nilai bilangan bulat. Ekspresi, rumus, dan fungsi tidak valid untuk parameter `rds.pg_stat_ramdisk_size`. Pastikan untuk memulai ulang klaster DB agar perubahan diterapkan. Untuk informasi tentang mengatur parameter, lihat . Untuk informasi selengkapnya tentang memulai ulang klaster DB, lihat Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora.

Sebagai contoh, AWS CLI perintah berikut menetapkan parameter disk RAM ke 256 MB.

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

Setelah Anda memulai ulang klaster DB, jalankan perintah berikut untuk melihat status `stats_temp_directory`:

```
postgres=> SHOW stats_temp_directory;
```

Perintah tersebut akan menghasilkan hal berikut:

```
stats_temp_directory
---------------------------
/rdsdbramdisk/pg_stat_tmp
(1 row)
```

# Mengelola file sementara dengan PostgreSQL

Di PostgreSQL, kueri kompleks mungkin melakukan beberapa operasi pengurutan atau hash pada saat yang sama, dengan setiap operasi menggunakan memori instance untuk menyimpan hasil hingga nilai yang ditentukan dalam parameter. `work_mem` Jika memori instans tidak cukup, file sementara akan dibuat untuk menyimpan hasil. Hasil ini ditulis ke disk untuk menyelesaikan eksekusi kueri. Kemudian, file-file ini secara otomatis dihapus setelah kueri selesai. Dalam Aurora PostgreSQL, file ini berbagi penyimpanan lokal dengan file log lainnya. Anda dapat memantau ruang penyimpanan lokal klaster PostgreSQL DB Aurora Anda dengan menonton metrik Amazon. CloudWatch `FreeLocalStorage` Untuk informasi selengkapnya, lihat Memecahkan masalah penyimpanan lokal.

Sebaiknya gunakan Aurora Optimized instans cluster untuk beban kerja yang melibatkan beberapa kueri bersamaan yang meningkatkan penggunaan file sementara. Cluster ini menggunakan penyimpanan tingkat blok solid state drive (SSDNVMe) berbasis Non-Volatile Memory Express () lokal untuk menempatkan file sementara. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads.

Anda dapat menggunakan parameter dan fungsi berikut untuk mengelola file sementara dalam instans Anda.

- **`temp_file_limit`** – Parameter ini membatalkan kueri apa pun yang melebihi ukuran temp_files dalam KB. Batas ini mencegah kueri apa pun berjalan tanpa henti dan menghabiskan ruang disk dengan file sementara. Anda dapat memperkirakan nilai menggunakan hasil dari parameter `log_temp_files`. Sebagai praktik terbaik, periksa perilaku beban kerja dan tetapkan batas sesuai dengan estimasi. Contoh berikut menunjukkan bagaimana kueri dibatalkan saat melampaui batas.

```
postgres=>select * from pgbench_accounts, pg_class, big_table;
```

```
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **`log_temp_files`** – Parameter ini mengirimkan pesan ke postgresql.log ketika file sementara dari sebuah sesi dihapus. Parameter ini menghasilkan log setelah kueri berhasil diselesaikan. Oleh karena itu, ini mungkin tidak membantu dalam memecahkan masalah kueri yang aktif dan berjalan lama.

Contoh berikut menunjukkan bahwa ketika kueri berhasil diselesaikan, entri dicatat dalam file postgresql.log, sedangkan file sementara dibersihkan.

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** – Fungsi yang tersedia dari RDS untuk PostgreSQL 13 dan versi yang lebih baru ini memberikan visibilitas terhadap penggunaan file sementara saat ini. Kueri yang sudah selesai tidak muncul di hasil fungsi. Dalam contoh berikut, Anda dapat melihat hasil dari fungsi ini.

```
postgres=>select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=>select query from pg_stat_activity where pid = 8355;

query
--------------------------------------------------------------------------------
```

```
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

Nama file mencakup ID pemrosesan (PID) dari sesi yang menghasilkan file sementara. Kueri yang lebih maju, seperti pada contoh berikut, melakukan penjumlahan file sementara untuk setiap PID.

```
postgres=>select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
------+-------+------------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** – Jika Anda mengaktifkan parameter pg_stat_statements, Anda dapat melihat rata-rata penggunaan file sementara per panggilan. Anda dapat mengidentifikasi query_id dari kueri dan menggunakannya untuk memeriksa penggunaan file sementara seperti yang ditunjukkan pada contoh berikut.

```
postgres=>select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=>select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+---------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                      388678
(1 row)
```

- **Performance Insights** – Di dasbor Wawasan Performa, Anda dapat melihat penggunaan file sementara dengan mengaktifkan metrik temp_bytes dan temp_files. Kemudian, Anda dapat melihat rata-rata kedua metrik ini dan melihat sejauh mana kesesuaiannya dengan beban kerja kueri. Tampilan dalam Wawasan Performa tidak secara khusus menampilkan kueri yang menghasilkan file sementara. Namun, jika Anda menggabungkan Wawasan Performa dengan kueri yang ditampilkan untuk `pg_ls_tmpdir`, Anda dapat memecahkan masalah, menganalisis, dan menentukan perubahan dalam beban kerja kueri.

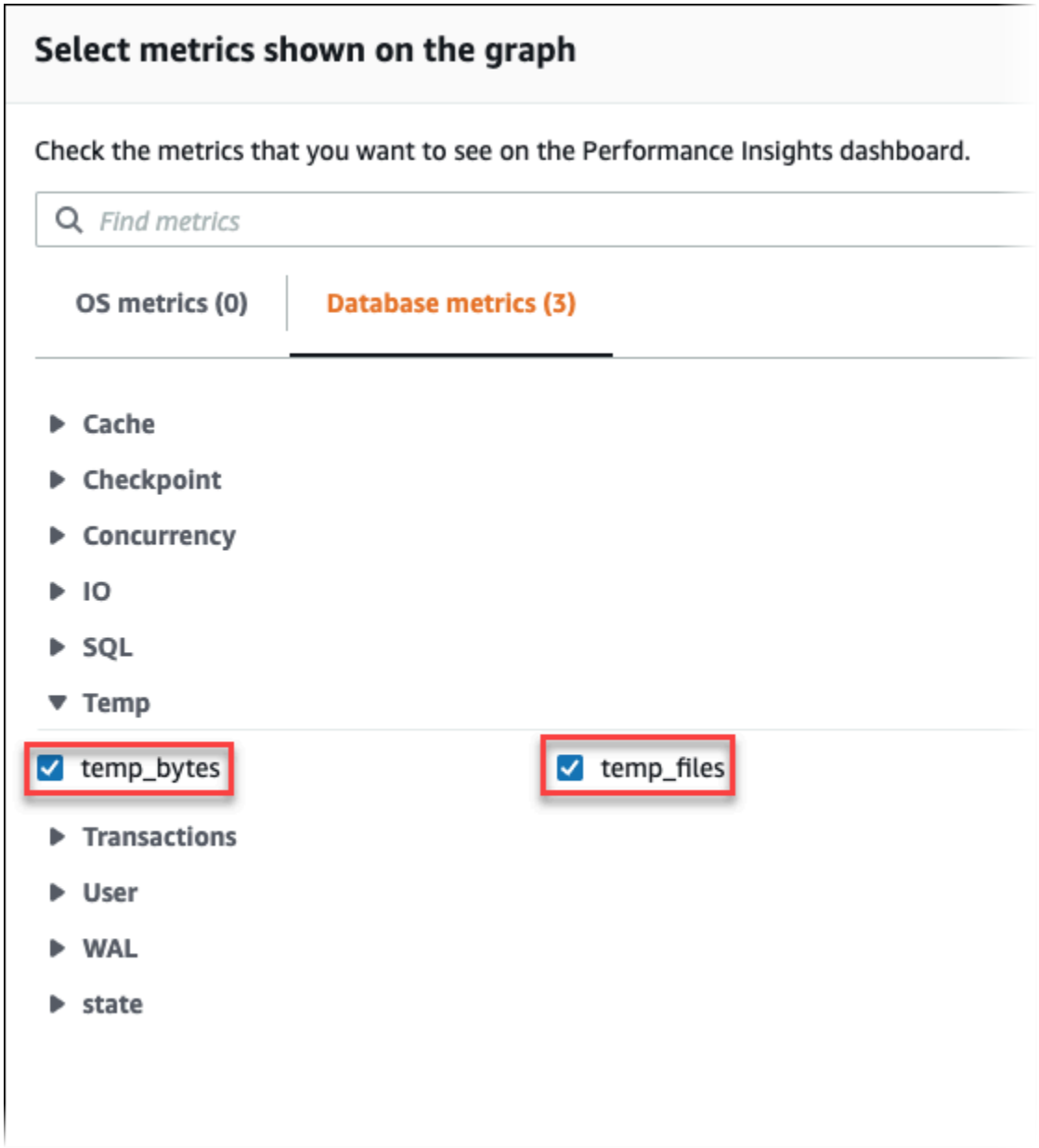
  Untuk informasi selengkapnya tentang cara menganalisis metrik dan kueri dengan Performance Insights, lihat. Menganalisis metrik dengan dasbor Wawasan Performa

  Untuk contoh melihat penggunaan file sementara dengan Performance Insights, lihat Melihat penggunaan file sementara dengan Performance Insights

## Melihat penggunaan file sementara dengan Performance Insights

Anda dapat menggunakan Performance Insights untuk melihat penggunaan file sementara dengan mengaktifkan metrik temp_bytes dan temp_files. Tampilan di Performance Insights tidak menampilkan kueri spesifik yang menghasilkan file sementara, namun jika Anda menggabungkan Performance Insights dengan kueri yang ditampilkan`pg_ls_tmpdir`, Anda dapat memecahkan masalah, menganalisis, dan menentukan perubahan dalam beban kerja kueri.

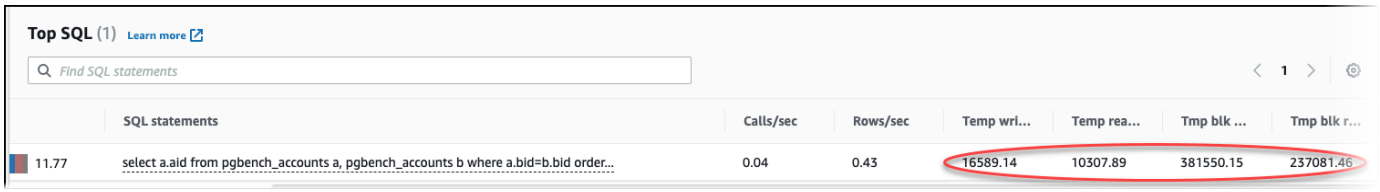
1. Di dasbor Wawasan Performa, pilih Kelola Metrik.
2. Pilih Metrik basis data, lalu pilih metrik temp_bytes dan temp_files seperti yang ditunjukkan pada gambar berikut.



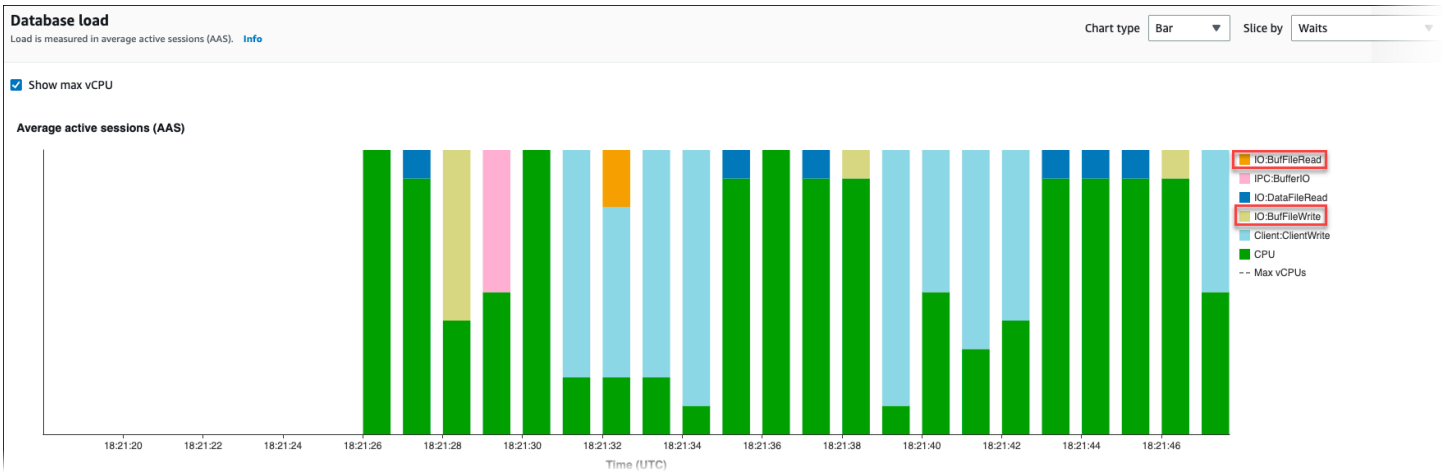

3. Di SQL tab Atas, pilih ikon Preferensi.
4. Di jendela Preferensi, nyalakan statistik berikut untuk muncul di SQL tab Atas dan pilih Lanjutkan.

   - Temp writes/detik
   - Temp reads/detik
   - Tmp blk write/panggilan
   - Tmp blk read/panggilan

5. File sementara rusak saat digabungkan kueri yang ditampilkan untuk `pg_ls_tmpdir`, seperti yang ditunjukkan pada contoh berikut.




Peristiwa `IO:BufFileRead` dan `IO:BufFileWrite` terjadi ketika kueri teratas di beban kerja Anda sering membuat file sementara. Anda dapat menggunakan Performance Insights untuk mengidentifikasi kueri teratas yang menunggu `IO:BufFileRead` dan `IO:BufFileWrite` dengan meninjau Average Active Session (AAS) di bagian Pemuatan Database dan Top. SQL




Untuk informasi selengkapnya tentang cara menganalisis kueri teratas dan muatan berdasarkan peristiwa tunggu dengan Wawasan Performa, lihat Ikhtisar SQL tab Top. Anda harus mengidentifikasi dan menyetel kueri yang menyebabkan peningkatan penggunaan file sementara dan peristiwa tunggu terkait. Untuk informasi lebih lanjut tentang peristiwa tunggu dan remediasi ini, lihat : BufFileRead dan IO: BufFileWrite.

> **Note**
>
> Parameter `work_mem` mengontrol ketika operasi pengurutan kehabisan memori dan hasilnya ditulis ke dalam file sementara. Sebaiknya Anda tidak mengubah pengaturan parameter ini lebih tinggi dari nilai default karena akan memungkinkan setiap sesi basis data mengonsumsi lebih banyak memori. Selain itu, satu sesi yang melakukan penggabungan dan pengurutan kompleks dapat melakukan operasi paralel di mana setiap operasi mengonsumsi memori. Sebagai praktik terbaik, ketika Anda memiliki laporan besar dengan beberapa penggabungan dan pengurutan, atur parameter ini pada tingkat sesi dengan menggunakan perintah `SET`

`work_mem`. Kemudian, perubahan hanya diterapkan pada sesi saat ini dan tidak mengubah nilai secara global.

# Menyetel dengan peristiwa tunggu di Aurora PostgreSQL

Peristiwa tunggu adalah alat penyetelan penting untuk Aurora PostgreSQL. Ketika Anda dapat mengetahui mengapa sesi menunggu sumber daya dan apa yang sedang dilakukan, Anda lebih mampu mengurangi kemacetan. Anda dapat menggunakan informasi di bagian ini untuk menemukan kemungkinan penyebab dan tindakan korektif. Sebelum mempelajari bagian ini, kami sangat menyarankan agar Anda memahami konsep dasar Aurora, terutama topik-topik berikut:

- Penyimpanan Amazon Aurora
- Mengelola performa dan penskalaan untuk klaster DB Aurora

**Important**

Peristiwa tunggu di bagian ini khusus untuk Aurora PostgreSQL. Gunakan informasi di bagian ini untuk menyetel Amazon Aurora saja, bukan RDS for PostgreSQL.
Beberapa peristiwa tunggu di bagian ini tidak memiliki analog dalam versi sumber terbuka dari mesin basis data ini. Peristiwa tunggu lainnya memiliki nama yang sama dengan peristiwa di mesin sumber terbuka, tetapi berperilaku berbeda. Misalnya, penyimpanan Amazon Aurora bekerja secara berbeda dari penyimpanan sumber terbuka, sehingga peristiwa tunggu terkait penyimpanan menunjukkan kondisi sumber daya yang berbeda.

Topik

- Konsep penting untuk penyetelan Aurora PostgreSQL
- Peristiwa tunggu Aurora PostgreSQL
- Klien: ClientRead
- Klien: ClientWrite
- CPU
- IO:BufFileRead dan IO:BufFileWrite
- IO: DataFileRead
- IO: XactSync

- IPC:DamRecordTxAck
- IPC: Acara tunggu paralel
- Lock:advisory
- Lock:extend
- Lock:Relation
- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO (IPC: Bufferio)
- LWLock:lock_manager
- LWLock:MultiXact
- LWLock:pg_stat_statement
- Timeout:PgSleep

## Konsep penting untuk penyetelan Aurora PostgreSQL

Sebelum Anda menyetel basis data Aurora PostgreSQL, pastikan untuk mempelajari apa itu peristiwa tunggu dan mengapa itu terjadi. Juga tinjau memori dasar dan arsitektur disk Aurora PostgreSQL. Untuk diagram arsitektur yang bermanfaat, lihat wikibook PostgreSQL.

Topik

- Peristiwa tunggu Aurora PostgreSQL
- Memori Aurora PostgreSQL
- Proses Aurora PostgreSQL

### Peristiwa tunggu Aurora PostgreSQL

Peristiwa tunggu menunjukkan sumber daya yang sedang menunggu sesi. Misalnya, peristiwa tunggu `Client:ClientRead` terjadi ketika Aurora PostgreSQL menunggu untuk menerima data dari klien. Sumber daya khas yang ditunggu sesi meliputi yang berikut:

- Akses single-threaded ke buffer, misalnya, saat sesi mencoba mengubah buffer

- Baris yang saat ini dikunci oleh sesi lain
- Pembacaan file data
- Penulisan file log

Misalnya, untuk memenuhi kueri, sesi mungkin melakukan pemindaian tabel lengkap. Jika data belum ada dalam memori, sesi menunggu disk I/O selesai. Ketika buffer dibaca ke dalam memori, sesi mungkin perlu menunggu karena sesi lain mengakses buffer yang sama. Basis data mencatat peristiwa tunggu dengan menggunakan peristiwa tunggu standar. Peristiwa tersebut dikelompokkan ke dalam kategori.

Peristiwa tunggu tidak dengan sendirinya menunjukkan masalah performa. Misalnya, jika data yang diminta tidak ada dalam memori, membaca data dari disk diperlukan. Jika satu sesi mengunci baris untuk pembaruan, sesi lain akan menunggu baris tersebut dibuka sehingga dapat memperbaruinya. Komit perlu menunggu penulisan ke file log selesai. Peristiwa tunggu merupakan bagian integral dari fungsi normal basis data.

Sejumlah besar peristiwa tunggu biasanya menunjukkan masalah performa. Dalam kasus seperti itu, Anda dapat menggunakan data peristiwa tunggu untuk menentukan tempat sesi menghabiskan waktu. Misalnya, jika laporan yang biasanya berjalan dalam hitungan menit sekarang berjalan selama berjam-jam, Anda dapat mengidentifikasi peristiwa tunggu yang berkontribusi paling besar terhadap total waktu tunggu. Jika Anda dapat menentukan penyebab peristiwa tunggu teratas, terkadang Anda dapat membuat perubahan yang meningkatkan performa. Misalnya, jika sesi Anda menunggu pada baris yang telah dikunci oleh sesi lain, Anda dapat mengakhiri sesi penguncian.

## Memori Aurora PostgreSQL

Memori Aurora PostgreSQL dibagi menjadi bersama dan lokal.

Topik

- Memori bersama di Aurora PostgreSQL
- Memori lokal di Aurora PostgreSQL

### Memori bersama di Aurora PostgreSQL

Aurora PostgreSQL mengalokasikan memori bersama saat instans dimulai. Memori bersama dibagi menjadi beberapa sub-area. Anda dapat menemukan deskripsi untuk yang paling penting berikut ini.

Topik

- Buffer bersama
- Buffer log write ahead (WAL)

Buffer bersama

Kolam buffer bersama adalah area memori Aurora PostgreSQL yang menampung semua halaman yang sedang atau telah digunakan oleh koneksi aplikasi. Halaman adalah versi memori dari blok disk. Kumpulan buffer bersama menyimpan blok data yang dibaca dari disk. Kumpulan tersebut mengurangi kebutuhan untuk membaca ulang data dari disk, sehingga membuat basis data beroperasi lebih efisien.

Setiap tabel dan indeks disimpan sebagai array halaman dengan ukuran tetap. Setiap blok berisi beberapa tuple, yang sesuai dengan baris. Tuple dapat disimpan di halaman mana pun.

Kolam buffer bersama memiliki memori terbatas. Jika permintaan baru memerlukan halaman yang tidak ada dalam memori, dan tidak ada lagi memori, Aurora PostgreSQL mengosongkan halaman yang jarang digunakan untuk mengakomodasi permintaan tersebut. Kebijakan pengosongan diimplementasikan oleh algoritma clock sweep.

Parameter `shared_buffers` menentukan berapa banyak memori server dikhususkan untuk menyimpan data dalam cache.

Buffer log write ahead (WAL)

Buffer log write-ahead (WAL) menyimpan data transaksi yang kemudian ditulis Aurora PostgreSQL ke penyimpanan persisten. Menggunakan mekanisme WAL, Aurora PostgreSQL dapat melakukan hal berikut:

- Memulihkan data setelah kegagalan
- Kurangi disk I/O dengan menghindari sering menulis ke disk

Ketika klien mengubah data, Aurora PostgreSQL menulis perubahan ke buffer WAL. Ketika klien mengeluarkan `COMMIT`, proses penulis WAL menulis data transaksi ke file WAL.

Parameter `wal_level` menentukan berapa banyak informasi yang ditulis ke WAL.

Memori lokal di Aurora PostgreSQL

Setiap proses backend mengalokasikan memori lokal untuk pemrosesan kueri.

Topik


- Area memori kerja
- Area memori kerja pemeliharaan
- Area buffer sementara


Area memori kerja

Area memori kerja menyimpan data sementara untuk kueri yang melakukan pengurutan dan hash. Misalnya, kueri dengan klausa `ORDER BY` melakukan pengurutan. Kueri menggunakan tabel hash dalam gabungan dan agregasi hash.

Parameter `work_mem` jumlah memori yang akan digunakan oleh operasi pengurutan internal dan tabel hash sebelum menulis ke file disk sementara. Nilai default-nya adalah 4 MB. Beberapa sesi dapat berjalan secara bersamaan, dan setiap sesi dapat menjalankan operasi pemeliharaan secara paralel. Karena alasan ini, total memori kerja yang digunakan dapat menjadi kelipatan dari pengaturan `work_mem`.

Area memori kerja pemeliharaan

Area memori kerja pemeliharaan menyimpan data untuk operasi pemeliharaan. Operasi ini termasuk memvakum, membuat indeks, dan menambahkan kunci asing.

Parameter `maintenance_work_mem` menentukan jumlah maksimum memori yang akan digunakan oleh operasi pemeliharaan. Nilai default-nya adalah 64 MB. Sebuah sesi basis data hanya dapat menjalankan satu operasi pemeliharaan dalam satu waktu.

Area buffer sementara

Area buffer sementara menyimpan tabel sementara untuk setiap sesi basis data.

Setiap sesi mengalokasikan buffer sementara sesuai kebutuhan hingga batas yang Anda tentukan. Saat sesi berakhir, server menghapus buffer.

Parameter `temp_buffers` mengatur jumlah maksimum buffer sementara yang digunakan oleh setiap sesi. Sebelum penggunaan pertama tabel sementara dalam sesi, Anda dapat mengubah nilai `temp_buffers`.

## Proses Aurora PostgreSQL

Aurora PostgreSQL menggunakan beberapa proses.

Topik


- Proses postmaster
- Proses backend
- Proses latar belakang


### Proses postmaster

Proses postmaster adalah proses pertama yang dimulai ketika Anda memulai Aurora PostgreSQL. Proses postmaster memiliki tanggung jawab utama berikut:

- Membagi dan memantau proses latar belakang
- Menerima permintaan autentikasi dari proses klien, dan mengautentikasi permintaan sebelum mengizinkan basis data untuk melayani permintaan

### Proses backend

Jika postmaster mengautentikasi permintaan klien, postmaster akan melakukan proses backend baru, juga disebut proses postgres. Satu proses klien terhubung ke persis satu proses backend. Proses klien dan proses backend berkomunikasi secara langsung tanpa intervensi oleh proses postmaster.

### Proses latar belakang

Proses postmaster membagi beberapa proses yang melakukan tugas backend yang berbeda. Beberapa hal yang lebih penting termasuk berikut ini:

- Penulis WAL

  Aurora PostgreSQL menulis data dalam buffer WAL (log write ahead) ke file log. Prinsip log write ahead adalah bahwa basis data tidak dapat menulis perubahan pada file data sampai setelah basis data menulis catatan log yang menjelaskan perubahan tersebut ke disk. Mekanisme WAL mengurangi disk I/O, dan memungkinkan Aurora PostgreSQL untuk menggunakan log untuk memulihkan basis data setelah kegagalan.
- Penulis latar belakang

  Proses ini secara berkala menulis halaman kotor (diubah) dari buffer memori ke file data. Sebuah halaman menjadi kotor ketika proses backend memodifikasinya dalam memori.

- Daemon autovacuum

  Daemon terdiri dari hal-hal berikut:

  - Peluncur autovacuum
  - Proses pekerja autovacuum

  Saat autovacuum diaktifkan, autovacuum tersebut memeriksa tabel yang memiliki banyak tuple yang dimasukkan, diperbarui, atau dihapus. Daemon memiliki tanggung jawab sebagai berikut:

  - Memulihkan atau menggunakan kembali ruang disk yang ditempati oleh baris yang diperbarui atau dihapus
  - Memperbarui statistik yang digunakan oleh perencana
  - Melindungi dari kehilangan data lama karena wraparound ID transaksi

  Fitur autovacuum mengotomatiskan eksekusi `VACUUM` dan perintah `ANALYZE`. `VACUUM` memiliki varian berikut: standar dan penuh. Vakum standar berjalan secara paralel dengan operasi basis data lainnya. `VACUUM FULL` membutuhkan kunci eksklusif di tabel yang sedang dikerjakannya. Dengan demikian, tidak dapat berjalan secara paralel dengan operasi yang mengakses tabel yang sama. `VACUUM`menciptakan sejumlah besar I/O lalu lintas, yang dapat menyebabkan kinerja yang buruk untuk sesi aktif lainnya.

## Peristiwa tunggu Aurora PostgreSQL

Tabel berikut mencantumkan peristiwa tunggu untuk Aurora PostgreSQL yang paling sering menunjukkan masalah performa, dan meringkas penyebab paling umum dan tindakan korektif. Peristiwa tunggu berikut adalah subset dari daftar di Peristiwa tunggu Amazon Aurora PostgreSQL.

| Peristiwa tunggu | Definisi |
| --- | --- |
| Klien: ClientRead | Peristiwa ini terjadi ketika Aurora PostgreSQL menunggu untuk menerima data dari klien. |
| Klien: ClientWrite | Peristiwa ini terjadi ketika Aurora PostgreSQL menunggu untuk menulis data ke klien. |
| CPU | Peristiwa ini terjadi ketika thread aktif di CPU atau sedang menunggu CPU. |

| Peristiwa tunggu | Definisi |
| --- | --- |
| IO:BufFileRead dan IO:BufFileWrite | Peristiwa ini terjadi ketika Aurora PostgreSQL membuat file sementara. |
| IO: DataFileRead | Peristiwa ini terjadi ketika koneksi menunggu pada proses backend untuk membaca halaman yang diperlukan dari penyimpanan karena halaman tidak tersedia dalam memori bersama. |
| IO: XactSync | Peristiwa ini terjadi ketika basis data menunggu subsistem penyimpanan Aurora untuk mengakui commit transaksi reguler, atau commit atau pengembalian transaksi yang disiapkan. |
| IPC:DamRecordTxAck | Peristiwa ini terjadi ketika Aurora PostgreSQL dalam sesi menggunakan aliran aktivitas basis data menghasilkan peristiwa aliran aktivitas, lalu menunggu peristiwa tersebut menjadi durabel. |
| Lock:advisory | Peristiwa ini terjadi ketika aplikasi PostgreSQL menggunakan kunci untuk mengoordinasikan aktivitas di beberapa sesi. |
| Lock:extend | Peristiwa ini terjadi ketika proses backend menunggu untuk mengunci relasi untuk memperpanjangnya selagi proses lain memiliki kunci pada relasi itu untuk tujuan yang sama. |
| Lock:Relation | Peristiwa ini terjadi ketika kueri menunggu untuk memperoleh kunci pada tabel atau tampilan yang saat ini dikunci oleh transaksi lain. |
| Lock:transactionid | Peristiwa ini terjadi ketika transaksi sedang menunggu kunci tingkat baris. |
| Lock:tuple | Peristiwa ini terjadi ketika proses backend menunggu untuk mendapatkan kunci pada tuple. |

| Peristiwa tunggu | Definisi |
| --- | --- |
| LWLock:buffer_content (BufferContent) | Peristiwa ini terjadi ketika suatu sesi menunggu untuk membaca atau menulis halaman data di memori selagi sesi lain mengunci halaman tersebut untuk penulisan. |
| LWLock:buffer_mapping | Peristiwa ini terjadi ketika sesi menunggu untuk mengaitkan blok data dengan buffer di kolam buffer bersama. |
| LWLock:BufferIO (IPC: Bufferio) | Peristiwa ini terjadi ketika Aurora PostgreSQL atau RDS untuk PostgreSQL sedang menunggu proses lain untuk menyelesaikan operasi (I/O) mereka ketika secara bersamaan mencoba mengakses halaman. input/output |
| LWLock:lock_manager | Peristiwa ini terjadi ketika mesin Aurora PostgreSQ L mempertahankan area memori kunci bersama untuk mengalokasikan, memeriksa, dan mendealok asikan kunci ketika kunci jalur cepat tidak memungkinkan. |
| LWLock:MultiXact | Jenis peristiwa ini terjadi ketika Aurora PostgreSQ L menjaga sesi terbuka untuk menyelesaikan beberapa transaksi yang melibatkan baris yang sama dalam tabel. Peristiwa tunggu menunjukk an aspek mana dari beberapa pemrosesan transaksi yang menghasilkan peristiwa tunggu, yaitu,: MultiXactOffset SLRU, LWLock:, LWLock: MultiXactMember SLRUMultiXactOffsetBuffer, atau LWLock:. LWLock MultiXactMemberBuffer |
| Timeout:PgSleep | Peristiwa ini terjadi ketika proses server telah memanggil fungsi `pg_sleep` dan menunggu batas waktu tidur berakhir. |

# Klien: ClientRead

Peristiwa `Client:ClientRead` terjadi saat Aurora PostgreSQL menunggu untuk menerima data dari klien.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk Aurora PostgreSQL versi 10 dan yang lebih tinggi.

## Konteks

Klaster DB Aurora PostgreSQL menunggu untuk menerima data dari klien. Klaster DB Aurora PostgreSQL harus menerima data dari klien agar dapat mengirim lebih banyak data ke klien. Waktu saat klaster menunggu sebelum menerima data dari klien adalah peristiwa `Client:ClientRead`.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab umum peristiwa `Client:ClientRead` muncul dalam peristiwa tunggu teratas mencakup yang berikut:

Peningkatan latensi jaringan

Mungkin terdapat peningkatan latensi jaringan antara klaster DB Aurora PostgreSQL dan klien. Latensi jaringan yang lebih tinggi akan menambah waktu yang diperlukan agar klaster DB dapat menerima data dari klien.

Peningkatan beban pada klien

Mungkin terdapat tekanan CPU atau saturasi jaringan pada klien. Peningkatan beban pada klien dapat menunda transmisi data dari klien ke klaster DB Aurora PostgreSQL.

Round trip jaringan yang berlebihan

Sejumlah besar round trip jaringan antara klaster DB Aurora PostgreSQL dan klien dapat menunda transmisi data dari klien ke klaster DB Aurora PostgreSQL.

Operasi penyalinan besar

Selama operasi penyalinan, data ditransfer dari sistem file klien ke klaster DB Aurora PostgreSQL. Mengirim data dalam jumlah besar ke klaster DB dapat menunda transmisi data dari klien ke klaster DB.

Koneksi klien idle

Koneksi ke instans DB Aurora PostgreSQL berada dalam status idle dalam transaksi dan sedang menunggu klien mengirim lebih banyak data atau mengeluarkan perintah. Status ini dapat menyebabkan peningkatan peristiwa `Client:ClientRead`.

PgBouncer digunakan untuk penyatuan koneksi

PgBouncer memiliki pengaturan konfigurasi jaringan tingkat rendah yang disebut`pkt_buf`, yang diatur ke 4.096 secara default. Jika beban kerja mengirimkan paket kueri yang lebih besar dari 4.096 byte PgBouncer, kami sarankan untuk meningkatkan pengaturan menjadi 8.192. `pkt_buf` Jika pengaturan baru tidak mengurangi jumlah peristiwa `Client:ClientRead`, sebaiknya tingkatkan pengaturan `pkt_buf` ke nilai yang lebih besar, seperti 16.384 atau 32.768. Jika teks kueri berukuran besar, pengaturan yang lebih besar dapat sangat membantu.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster
- Menskalakan klien
- Gunakan instans generasi saat ini
- Meningkatkan bandwidth jaringan
- Pantau nilai maksimum untuk performa jaringan
- Pantau transaksi dalam status "idle dalam transaksi"

Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster

Untuk mengurangi latensi jaringan dan meningkatkan throughput jaringan, tempatkan klien di subnet cloud privat virtual (VPC) dan Zona Ketersediaan yang sama seperti klaster DB Aurora PostgreSQL. Pastikan klien secara geografis berada sedekat mungkin dengan klaster DB.

Menskalakan klien

Menggunakan Amazon CloudWatch atau metrik host lainnya, tentukan apakah klien Anda saat ini dibatasi oleh CPU atau bandwidth jaringan, atau keduanya. Jika klien dibatasi, skalakan klien sesuai yang diperlukan.

Gunakan instans generasi saat ini

Dalam kasus tertentu, Anda mungkin tidak menggunakan kelas instans DB yang mendukung frame jumbo. Jika Anda menjalankan aplikasi di Amazon EC2, pertimbangkan untuk menggunakan instans generasi saat ini untuk klien. Selain itu, konfigurasikan unit transmisi maksimum (MTU) pada sistem operasi klien. Teknik ini dapat mengurangi jumlah round trip jaringan dan meningkatkan throughput jaringan. Untuk informasi selengkapnya, lihat Jumbo frame (9001 MTU) di Panduan Pengguna Amazon EC2.

Untuk informasi tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB. Untuk menentukan kelas instans DB yang setara dengan jenis instans Amazon EC2, tempatkan `db.` sebelum nama jenis instans Amazon EC2. Misalnya, instans Amazon EC2 `r5.8xlarge` setara dengan kelas instans DB `db.r5.8xlarge`.

Meningkatkan bandwidth jaringan

Gunakan `NetworkReceiveThroughput` dan CloudWatch metrik `NetworkTransmitThroughput` Amazon untuk memantau lalu lintas jaringan masuk dan keluar di cluster DB. Metrik ini dapat membantu mengetahui apakah bandwidth jaringan cukup untuk beban kerja Anda.

Jika bandwidth jaringan Anda tidak cukup, tingkatkan. Jika AWS klien atau instans DB Anda mencapai batas bandwidth jaringan, satu-satunya cara untuk meningkatkan bandwidth adalah dengan meningkatkan ukuran instans DB Anda.

Untuk informasi selengkapnya tentang CloudWatch metrik, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

Pantau nilai maksimum untuk performa jaringan

Jika Anda menggunakan klien Amazon EC2, Amazon EC2 menyediakan nilai maksimum untuk metrik performa jaringan, termasuk bandwidth jaringan masuk dan keluar agregat. Amazon EC2 juga menyediakan pelacakan koneksi untuk memastikan paket dikembalikan sebagaimana diharapkan dan akses layanan tautan-lokal untuk layanan seperti Sistem Nama Domain (DNS). Untuk memantau nilai maksimum ini, gunakan driver jaringan yang ditingkatkan saat ini dan pantau performa jaringan untuk klien Anda.

Untuk informasi selengkapnya, lihat Memantau kinerja jaringan untuk instans Amazon EC2 Anda di Panduan Pengguna Amazon EC2 dan Memantau kinerja jaringan untuk instans Amazon EC2 Anda di Panduan Pengguna Amazon EC2.

Pantau transaksi dalam status "idle dalam transaksi"

Periksa apakah Anda memiliki peningkatan jumlah koneksi `idle in transaction`. Untuk melakukannya, pantau kolom `state` dalam tabel `pg_stat_activity`. Anda mungkin dapat mengidentifikasi sumber koneksi dengan menjalankan kueri seperti yang berikut ini.

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

## Klien: ClientWrite

Peristiwa `Client:ClientWrite` terjadi saat Aurora PostgreSQL menunggu untuk menulis data ke klien.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk Aurora PostgreSQL versi 10 dan yang lebih tinggi.

### Konteks

Proses klien harus membaca semua data yang diterima dari klaster DB Aurora PostgreSQL sebelum klaster tersebut dapat mengirim lebih banyak data. Waktu saat klaster menunggu sebelum mengirim lebih banyak data ke klien adalah peristiwa `Client:ClientWrite`.

Throughput jaringan yang berkurang antara klaster DB Aurora PostgreSQL dan klien dapat menyebabkan peristiwa ini. Tekanan CPU dan saturasi jaringan pada klien juga dapat menyebabkan

peristiwa ini. Tekanan CPU adalah ketika CPU sepenuhnya digunakan dan ada tugas yang menunggu waktu CPU. Saturasi jaringan adalah saat jaringan antara basis data dan klien membawa lebih banyak data dari yang dapat ditanganinya.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab umum peristiwa `Client:ClientWrite` muncul dalam peristiwa tunggu teratas mencakup yang berikut:

Peningkatan latensi jaringan

Mungkin terdapat peningkatan latensi jaringan antara klaster DB Aurora PostgreSQL dan klien. Latensi jaringan yang lebih tinggi akan menambah waktu yang diperlukan klien untuk menerima data.

Peningkatan beban pada klien

Mungkin terdapat tekanan CPU atau saturasi jaringan pada klien. Peningkatan beban pada klien menunda penerimaan data dari klaster DB Aurora PostgreSQL.

Data dalam volume besar yang dikirim ke klien

Klaster DB Aurora PostgreSQL mungkin mengirimkan data dalam jumlah besar ke klien. Klien mungkin tidak dapat menerima data secepat klaster mengirimkannya. Aktivitas seperti penyalinan tabel besar dapat mengakibatkan peningkatan peristiwa `Client:ClientWrite`.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster
- Menggunakan instans generasi saat ini
- Mengurangi jumlah data yang dikirim ke klien
- Skalakan klien Anda

### Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster

Untuk mengurangi latensi jaringan dan meningkatkan throughput jaringan, tempatkan klien di subnet cloud privat virtual (VPC) dan Zona Ketersediaan yang sama seperti klaster DB Aurora PostgreSQL.

Menggunakan instans generasi saat ini

Dalam kasus tertentu, Anda mungkin tidak menggunakan kelas instans DB yang mendukung frame jumbo. Jika Anda menjalankan aplikasi di Amazon EC2, pertimbangkan untuk menggunakan instans generasi saat ini untuk klien. Selain itu, konfigurasikan unit transmisi maksimum (MTU) pada sistem operasi klien. Teknik ini dapat mengurangi jumlah round trip jaringan dan meningkatkan throughput jaringan. Untuk informasi selengkapnya, lihat Jumbo frame (9001 MTU) di Panduan Pengguna Amazon EC2.

Untuk informasi tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB. Untuk menentukan kelas instans DB yang setara dengan jenis instans Amazon EC2, tempatkan `db.` sebelum nama jenis instans Amazon EC2. Misalnya, instans Amazon EC2 `r5.8xlarge` setara dengan kelas instans DB `db.r5.8xlarge`.

Mengurangi jumlah data yang dikirim ke klien

Jika memungkinkan, sesuaikan aplikasi Anda untuk mengurangi jumlah data yang dikirim klaster DB Aurora PostgreSQL ke klien. Penyesuaian ini akan mengurangi pertentangan CPU dan jaringan pada klien.

Skalakan klien Anda

Menggunakan Amazon CloudWatch atau metrik host lainnya, tentukan apakah klien Anda saat ini dibatasi oleh CPU atau bandwidth jaringan, atau keduanya. Jika klien dibatasi, skalakan klien sesuai yang diperlukan.

# CPU

Peristiwa ini terjadi saat thread aktif di CPU atau sedang menunggu CPU.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

## Versi mesin yang didukung

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

## Konteks

CPU (Central Processing Unit) adalah komponen komputer yang menjalankan instruksi. Misalnya, instruksi CPU melakukan operasi aritmetika dan bertukar data dalam memori. Jika kueri meningkatkan jumlah instruksi yang dilakukannya melalui mesin basis data, waktu yang dihabiskan untuk menjalankan kueri akan meningkat. Penjadwalan CPU memberikan waktu CPU untuk suatu proses. Penjadwalan diatur oleh kernel sistem operasi.

Topik


- Bagaimana cara mengetahui kapan peristiwa tunggu ini terjadi?
- Metrik DBLoadCPU
- Metrik os.cpuUtilization
- Kemungkinan penyebab penjadwalan CPU


### Bagaimana cara mengetahui kapan peristiwa tunggu ini terjadi?

Peristiwa tunggu `CPU` ini menunjukkan bahwa proses backend aktif di CPU atau sedang menunggu CPU. Anda mengetahuinya terjadi saat kueri menunjukkan informasi berikut:

- Kolom `pg_stat_activity.state` memiliki nilai `active`.
- Kolom `wait_event_type` dan `wait_event` di `pg_stat_activity` adalah `null`.

Untuk melihat proses backend yang menggunakan atau menunggu CPU, jalankan kueri berikut.

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

### Metrik DBLoadCPU

Metrik Wawasan Performa untuk CPU adalah `DBLoadCPU`. Nilai untuk `DBLoadCPU` dapat berbeda dengan nilai untuk metrik Amazon CloudWatch `CPUUtilization`. Metrik CloudWatch dikumpulkan dari HyperVisor untuk instans basis data.

Metrik os.cpuUtilization

Metrik sistem operasi Wawasan Performa memberikan informasi terperinci tentang pemanfaatan CPU. Misalnya, Anda dapat menampilkan metrik berikut:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Wawasan Performa melaporkan penggunaan CPU oleh mesin basis data sebagai `os.cpuUtilization.nice.avg`.

Kemungkinan penyebab penjadwalan CPU

Dari perspektif sistem operasi, CPU aktif saat tidak menjalankan thread idle. CPU aktif saat menjalankan komputasi dan saat menunggu pada memori I/O. Jenis I/O ini mendominasi beban kerja basis data pada umumnya.

Proses cenderung menunggu untuk dijadwalkan pada CPU saat kondisi berikut terpenuhi:

- Metrik `CPUUtilization` CloudWatch mendekati 100%.
- Beban rata-rata lebih besar dari jumlah vCPU, yang menunjukkan beban berat. Anda dapat menemukan metrik `loadAverageMinute` di bagian metrik OS dalam Wawasan Performa.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa tunggu CPU terjadi lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Topik

- Kemungkinan penyebab lonjakan mendadak
- Kemungkinan penyebab frekuensi tinggi jangka panjang
- Corner case

Kemungkinan penyebab lonjakan mendadak

Penyebab lonjakan mendadak yang paling memungkinkan adalah sebagai berikut:

- Aplikasi Anda membuka terlalu banyak koneksi bersamaan ke basis data. Skenario ini dikenal sebagai "storm koneksi".
- Beban kerja aplikasi berubah dengan salah satu cara berikut:
  - Kueri baru
  - Peningkatan ukuran set data
  - Pemeliharaan atau pembuatan indeks
  - Fungsi baru
  - Operator baru
  - Peningkatan eksekusi kueri paralel
- Rencana eksekusi kueri Anda telah berubah. Dalam kasus tertentu, perubahan dapat menyebabkan peningkatan buffer. Misalnya, kueri sekarang menggunakan pemindaian berurutan saat sebelumnya menggunakan indeks. Dalam hal ini, kueri membutuhkan lebih banyak CPU untuk mencapai tujuan yang sama.

Kemungkinan penyebab frekuensi tinggi jangka panjang

Berikut adalah penyebab paling memungkinkan dari peristiwa yang berulang dalam jangka waktu lama:

- Terlalu banyak proses backend yang berjalan bersamaan pada CPU. Proses-proses ini dapat menjadi pekerja paralel.
- Kueri berperforma suboptimal karena membutuhkan buffer dalam jumlah besar.

Corner case

Jika tidak ada kemungkinan penyebab yang merupakan penyebab sebenarnya, situasi berikut mungkin terjadi:

- CPU menukar proses masuk dan keluar.
- Peralihan konteks CPU telah meningkat.
- Kode Aurora PostgreSQL tidak memiliki peristiwa tunggu.

## Tindakan

Jika peristiwa tunggu CPU mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Tanggapi peristiwa ini hanya saat performa menurun.

Topik

- Menyelidiki apakah basis data menyebabkan peningkatan CPU
- Mengetahui apakah jumlah koneksi meningkat
- Merespons perubahan beban kerja

### Menyelidiki apakah basis data menyebabkan peningkatan CPU

Periksa metrik `os.cpuUtilization.nice.avg` dalam Wawasan Performa. Jika nilai ini jauh lebih kecil daripada penggunaan CPU, proses non-basis data adalah kontributor utama ke CPU.

### Mengetahui apakah jumlah koneksi meningkat

Periksa metrik `DatabaseConnections` di Amazon CloudWatch. Tindakan Anda bergantung pada apakah jumlahnya meningkat atau menurun selama periode peningkatan peristiwa tunggu CPU.

#### Koneksi meningkat

Jika jumlah koneksi meningkat, bandingkan jumlah proses backend yang mengonsumsi CPU terhadap jumlah vCPU. Skenario berikut mungkin terjadi:

- Jumlah proses backend yang mengonsumsi CPU lebih kecil dari jumlah vCPU.

  Dalam hal ini, jumlah koneksi tidak menjadi masalah. Namun, Anda mungkin masih mencoba mengurangi pemanfaatan CPU.
- Jumlah proses backend yang mengonsumsi CPU lebih besar dari jumlah vCPU.

  Jika demikian, pertimbangkan opsi berikut:
  - Kurangi jumlah proses backend yang terhubung ke basis data Anda. Misalnya, terapkan solusi penggabungan koneksi seperti Proksi RDS. Untuk mempelajari selengkapnya, lihat Proksi Amazon RDS untuk Aurora.
  - Tingkatkan ukuran instans untuk mendapatkan jumlah vCPU yang lebih tinggi.
  - Jika berlaku, arahkan ulang beberapa beban kerja hanya-baca ke simpul pembaca.

Koneksi tidak meningkat

Periksa metrik `blks_hit` dalam Wawasan Performa. Cari korelasi antara peningkatan `blks_hit` dan penggunaan CPU. Skenario berikut mungkin terjadi:

- Penggunaan CPU dan `blks_hit` berkorelasi.

  Dalam hal ini, temukan pernyataan SQL teratas yang terkait dengan penggunaan CPU, lalu cari perubahan rencana. Anda dapat menggunakan salah satu teknik berikut:
  - Jelaskan rencana secara manual, lalu bandingkan dengan rencana eksekusi yang diperkirakan.
  - Cari peningkatan hit blokir per detik dan hit blokir lokal per detik. Di bagian SQL Teratas pada dasbor Wawasan Performa, pilih Preferensi.
- Penggunaan CPU dan `blks_hit` tidak berkorelasi.

  Jika demikian, ketahui apakah salah satu hal berikut terjadi:
  - Aplikasi ini dengan cepat terhubung ke dan memutuskan sambungan dari basis data.

    Jalankan diagnosis perilaku ini dengan mengaktifkan `log_connections` dan `log_disconnections`, lalu menganalisis log PostgreSQL. Pertimbangkan untuk menggunakan penganalisis log `pgbadger`. Untuk informasi selengkapnya, lihat https://github.com/darold/pgbadger.
  - OS kelebihan beban.

    Dalam hal ini, Wawasan Performa menunjukkan bahwa proses backend menggunakan CPU untuk waktu yang lebih lama dari biasanya. Cari bukti di metrik `os.cpuUtilization` Wawasan Performa atau metrik `CPUUtilization` CloudWatch. Jika sistem operasi kelebihan beban, lihat metrik Peningkatan Pemantauan untuk mendiagnosis lebih lanjut. Secara khusus, lihat daftar proses dan persentase CPU yang dikonsumsi oleh setiap proses.
  - Pernyataan SQL teratas mengonsumsi terlalu banyak CPU.

    Periksa pernyataan yang terkait dengan penggunaan CPU untuk mengetahui apakah pernyataan tersebut dapat menggunakan lebih sedikit CPU. Jalankan perintah `EXPLAIN`, lalu fokus pada simpul rencana yang memiliki dampak terbesar. Pertimbangkan untuk menggunakan visualizer rencana eksekusi PostgreSQL. Untuk mencoba alat bantu ini, lihat http://explain.dalibo.com/.

Merespons perubahan beban kerja

Jika beban kerja Anda berubah, cari jenis perubahan berikut:

Kueri baru

Periksa apakah kueri baru diharapkan. Jika ya, pastikan rencana eksekusinya dan jumlah eksekusi per detik diharapkan.

Peningkatan ukuran set data

Ketahui apakah partisi, jika belum diterapkan, dapat membantu. Strategi ini dapat mengurangi jumlah halaman yang perlu diambil oleh kueri.

Pemeliharaan atau pembuatan indeks

Periksa apakah jadwal pemeliharaan diharapkan. Praktik terbaik adalah menjadwalkan aktivitas pemeliharaan di luar aktivitas puncak.

Fungsi baru

Periksa apakah fungsi ini berjalan seperti yang diharapkan selama pengujian. Secara khusus, periksa apakah jumlah eksekusi per detik diharapkan.

Operator baru

Periksa apakah operator baru berfungsi seperti yang diharapkan selama pengujian.

Peningkatan dalam menjalankan kueri paralel

Ketahui apakah salah satu situasi berikut telah terjadi:

- Relasi atau indeks yang terlibat secara tiba-tiba berkembang ukurannya, sehingga sangat berbeda dengan `min_parallel_table_scan_size` atau `min_parallel_index_scan_size`.
- Perubahan terkini telah dilakukan pada `parallel_setup_cost` atau `parallel_tuple_cost`.
- Perubahan terkini telah dilakukan pada `max_parallel_workers` atau `max_parallel_workers_per_gather`.

## IO:BufFileRead dan IO:BufFileWrite

Peristiwa `IO:BufFileRead` dan `IO:BufFileWrite` terjadi saat Aurora PostgreSQL membuat file sementara. Saat operasi membutuhkan lebih banyak memori daripada yang saat ini ditentukan oleh parameter memori kerja, operasi akan menulis data sementara ke penyimpanan persisten. Operasi ini terkadang disebut "spilling to disk".

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

## Konteks

`IO:BufFileRead` dan `IO:BufFileWrite` berkaitan dengan area memori kerja dan area memori kerja pemeliharaan. Untuk informasi selengkapnya tentang area memori lokal ini, lihat Area memori kerja dan Area memori kerja pemeliharaan.

Nilai default untuk `work_mem` adalah 4 MB. Jika satu sesi melakukan operasi secara paralel, setiap pekerja yang menangani paralelisme menggunakan memori 4 MB. Untuk alasan ini, atur `work_mem` dengan hati-hati. Jika Anda meningkatkan nilai terlalu banyak, basis data yang menjalankan banyak sesi mungkin mengonsumsi terlalu banyak memori. Jika Anda menetapkan nilai terlalu rendah, Aurora PostgreSQL akan membuat file sementara di penyimpanan lokal. Disk I/O untuk file sementara ini dapat mengurangi performa.

Jika Anda mengamati urutan peristiwa berikut, basis data mungkin menghasilkan file sementara:

1. Penurunan ketersediaan secara tiba-tiba dan tajam
2. Pemulihan cepat untuk ruang kosong

Anda mungkin juga melihat pola "gergaji". Pola ini dapat menunjukkan bahwa basis data membuat file kecil secara terus-menerus.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Secara umum, peristiwa tunggu ini disebabkan oleh operasi yang mengonsumsi lebih banyak memori daripada yang dialokasikan oleh parameter `work_mem` atau `maintenance_work_mem`. Untuk mengompensasi, operasi menulis ke file sementara. Penyebab umum peristiwa `IO:BufFileRead` dan `IO:BufFileWrite` mencakup hal berikut:

Kueri yang membutuhkan lebih banyak memori daripada yang ada di area memori kerja

Kueri dengan karakteristik berikut menggunakan area memori kerja:

- Sambungan hash
- Klausa `ORDER BY`
- Klausa `GROUP BY`
- `DISTINCT`
- Fungsi jendela
- `CREATE TABLE AS SELECT`
- Penyegaran tampilan terwujud

Pernyataan yang membutuhkan lebih banyak memori daripada yang ada di area memori kerja pemeliharaan

Pernyataan berikut menggunakan area memori kerja pemeliharaan:

- `CREATE INDEX`
- `CLUSTER`

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi masalah
- Memeriksa kueri gabungan
- Memeriksa kueri ORDER BY dan GROUP BY
- Mencegah penggunaan operasi DISTINCT
- Mempertimbangkan untuk menggunakan fungsi jendela alih-alih fungsi GROUP BY
- Menyelidiki tampilan terwujud dan pernyataan CTAS
- Menggunakan pg_repack saat Anda membuat indeks
- Meningkatkan maintenance_work_mem saat Anda mengelompokkan tabel
- Menyetel memori untuk mencegah IO:BufFileRead dan IO:BufFileWrite

Mengidentifikasi masalah

Asumsikan situasi saat Wawasan Performa tidak diaktifkan dan Anda menduga bahwa `IO:BufFileRead` dan `IO:BufFileWrite` terjadi lebih sering daripada biasanya. Lakukan hal berikut:

1. Periksa metrik `FreeLocalStorage` di Amazon CloudWatch.
2. Cari pola gergaji, yang merupakan serangkaian paku bergerigi.

Pola gergaji menunjukkan konsumsi cepat dan pelepasan penyimpanan, yang sering dikaitkan dengan file sementara. Jika Anda melihat pola ini, aktifkan Wawasan Performa. Saat menggunakan Wawasan Performa, Anda dapat mengidentifikasi waktu terjadinya peristiwa tunggu dan kueri yang terkait dengannya. Solusi Anda bergantung pada kueri spesifik yang menyebabkan peristiwa.

Sebagai alternatif, atur parameter `log_temp_files`. Parameter ini mencatat semua kueri yang menghasilkan lebih dari ambang batas KB file sementara. Jika nilainya `0`, Aurora PostgreSQL mencatat semua file sementara. Jika nilainya `1024`, Aurora PostgreSQL mencatat semua kueri yang menghasilkan file sementara berukuran lebih besar dari 1 MB. Untuk informasi selengkapnya tentang `log_temp_files`, lihat Pencatatan dan Pelaporan Kesalahan dalam dokumentasi PostgreSQL.

Memeriksa kueri gabungan

Aplikasi Anda mungkin menggunakan gabungan. Misalnya, kueri berikut menggabungkan empat tabel.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

Kemungkinan penyebab lonjakan penggunaan file sementara adalah masalah dalam kueri itu sendiri. Misalnya, klausa yang rusak mungkin tidak memfilter gabungan dengan benar. Pertimbangkan gabungan inti kedua dalam contoh berikut.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

Kueri sebelumnya secara keliru menggabungkan `customer.id` ke `customer.id`, sehingga memberikan hasil perkalian Cartesian antara setiap pelanggan dan setiap pesanan. Jenis gabungan yang tak terduga ini menghasilkan file sementara yang besar. Tergantung pada ukuran tabel, kueri Cartesian bahkan dapat mengisi penyimpanan. Aplikasi Anda dapat memiliki gabungan Cartesian jika kondisi berikut terpenuhi:

- Anda melihat penurunan besar dan tajam dalam ketersediaan penyimpanan, yang diikuti oleh pemulihan cepat.
- Tidak ada indeks yang dibuat.
- Tidak ada pernyataan `CREATE TABLE FROM SELECT` yang dikeluarkan.
- Tidak ada tampilan terwujud yang disegarkan.

Untuk melihat apakah tabel sedang digabungkan menggunakan kunci yang tepat, periksa kueri dan petunjuk pemetaan relasional objek Anda. Perlu diperhatikan bahwa kueri tertentu dari aplikasi Anda tidak dipanggil sepanjang waktu, dan beberapa kueri dihasilkan secara dinamis.

Memeriksa kueri ORDER BY dan GROUP BY

Dalam beberapa kasus, klausa `ORDER BY` dapat menghasilkan file sementara yang berlebihan. Pertimbangkan panduan berikut:

- Hanya sertakan kolom dalam klausa `ORDER BY` saat perlu diurutkan. Pedoman ini sangat penting untuk kueri yang menampilkan ribuan baris dan menentukan banyak kolom dalam klausa `ORDER BY`.
- Pertimbangkan untuk membuat indeks guna mempercepat klausa `ORDER BY` saat klausa cocok dengan kolom yang memiliki urutan naik atau turun yang sama. Indeks sebagian lebih disukai karena lebih kecil. Indeks yang lebih kecil lebih cepat untuk dibaca dan dilewati.

- Jika Anda membuat indeks untuk kolom yang dapat menerima nilai kosong, pertimbangkan apakah Anda ingin nilai kosong disimpan di akhir atau di awal indeks.

  Jika memungkinkan, kurangi jumlah baris yang perlu diurutkan dengan memfilter set hasil. Jika Anda menggunakan pernyataan klausa atau subkueri `WITH`, perlu diperhatikan bahwa kueri dalam menghasilkan kumpulan hasil, lalu meneruskannya ke kueri luar. Semakin banyak baris yang dapat difilter kueri, semakin sedikit urutan yang perlu dilakukan kueri.
- Jika Anda tidak perlu mendapatkan set hasil lengkap, gunakan klausa `LIMIT`. Misalnya, jika Anda hanya menginginkan lima baris teratas, kueri yang menggunakan klausa `LIMIT` akan berhenti memberikan hasil. Dengan cara ini, kueri membutuhkan lebih sedikit memori dan file sementara.

Kueri yang menggunakan klausa `GROUP BY` juga dapat memerlukan file sementara. Kueri `GROUP BY` meringkas nilai dengan menggunakan fungsi seperti berikut:

- `COUNT`
- `AVG`
- `MIN`
- `MAX`
- `SUM`
- `STDDEV`

Untuk menyetel kueri `GROUP BY`, ikuti rekomendasi untuk kueri `ORDER BY`.

Mencegah penggunaan operasi DISTINCT

Jika memungkinkan, jangan gunakan operasi `DISTINCT` untuk menghapus baris duplikat. Semakin banyak baris duplikat yang tidak perlu, yang ditampilkan oleh kueri Anda, operasi `DISTINCT` menjadi semakin mahal. Jika memungkinkan, tambahkan filter dalam klausa `WHERE` meskipun Anda menggunakan filter yang sama untuk tabel yang berbeda. Memfilter kueri dan menggabungkan dengan benar akan meningkatkan performa Anda serta mengurangi penggunaan sumber daya. Hal tersebut juga mencegah laporan dan hasil yang salah.

Jika Anda perlu menggunakan `DISTINCT` untuk beberapa baris dari tabel yang sama, pertimbangkan untuk membuat indeks komposit. Mengelompokkan beberapa kolom dalam indeks dapat meningkatkan waktu untuk mengevaluasi baris yang berbeda. Selain itu, jika menggunakan Amazon Aurora PostgreSQL versi 10 atau yang lebih tinggi, Anda dapat menghubungkan statistik di antara beberapa kolom dengan menggunakan perintah `CREATE STATISTICS`.

Mempertimbangkan untuk menggunakan fungsi jendela alih-alih fungsi GROUP BY

Dengan menggunakan GROUP BY, Anda mengubah set hasil, lalu mengambil hasil agregat. Dengan menggunakan fungsi jendela, Anda mengumpulkan data tanpa mengubah set hasil. Fungsi jendela menggunakan klausa OVER untuk melakukan perhitungan di seluruh set yang ditentukan oleh kueri, dengan menghubungkan satu baris dengan yang lain. Anda dapat menggunakan semua fungsi GROUP BY dalam fungsi jendela, tetapi juga menggunakan fungsi seperti berikut:

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

Untuk meminimalkan jumlah file sementara yang dihasilkan oleh fungsi jendela, hapus duplikasi untuk set hasil yang sama saat Anda membutuhkan dua agregasi yang berbeda. Pertimbangkan kueri berikut.

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

Anda dapat menulis ulang kueri dengan klausa WINDOW sebagai berikut:

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

Secara default, perencana eksekusi Aurora PostgreSQL menggabungkan simpul serupa, sehingga tidak menggandakan operasi. Namun, dengan menggunakan deklarasi eksplisit untuk blok jendela, Anda dapat mempertahankan kueri dengan lebih mudah. Anda juga dapat meningkatkan performa dengan mencegah duplikasi.

Menyelidiki tampilan terwujud dan pernyataan CTAS

Saat tampilan terwujud disegarkan, kueri akan dijalankan. Kueri ini dapat berisi operasi seperti GROUP BY, ORDER BY, atau DISTINCT. Selama penyegaran, Anda mungkin mengamati sejumlah besar

file sementara dan peristiwa tunggu `IO:BufFileWrite` dan `IO:BufFileRead`. Demikian pula, saat Anda membuat tabel berdasarkan pernyataan `SELECT`, pernyataan `CREATE TABLE` tersebut menjalankan kueri. Untuk mengurangi file sementara yang dibutuhkan, optimalkan kueri.

Menggunakan pg_repack saat Anda membuat indeks

Saat Anda membuat indeks, mesin mengurutkan set hasil. Saat tabel bertambah besar, dan karena nilai di kolom yang diindeks menjadi lebih beragam, file sementara membutuhkan lebih banyak ruang. Dalam kebanyakan kasus, Anda tidak dapat mencegah pembuatan file sementara untuk tabel besar tanpa memodifikasi area memori kerja pemeliharaan. Untuk informasi selengkapnya, lihat Area memori kerja pemeliharaan.

Solusi yang memungkinkan saat membuat ulang indeks besar adalah menggunakan alat pg_repack. Untuk informasi selengkapnya, lihat Mengatur ulang tabel di basis data PostgreSQL dengan kunci minimal dalam dokumentasi pg_repack.

Meningkatkan maintenance_work_mem saat Anda mengelompokkan tabel

Perintah `CLUSTER` mengelompokkan tabel yang ditentukan oleh table_name berdasarkan indeks yang ada yang ditentukan oleh index_name. Aurora PostgreSQL secara fisik membuat ulang tabel agar sesuai dengan urutan indeks yang diberikan.

Saat penyimpanan magnetik lazim, klaster menjadi umum karena throughput penyimpanan terbatas. Sekarang penyimpanan berbasis SSD sudah umum, klaster menjadi kurang populer. Namun, jika mengelompokkan tabel, Anda masih dapat meningkatkan sedikit performa tergantung pada ukuran tabel, indeks, kueri, dan banyak lagi.

Jika Anda menjalankan perintah `CLUSTER` dan mengamati peristiwa tunggu `IO:BufFileWrite` dan `IO:BufFileRead`, sesuaikan `maintenance_work_mem`. Tingkatkan ukuran memori ke jumlah yang cukup besar. Nilai tinggi berarti mesin dapat menggunakan lebih banyak memori untuk operasi klaster.

Menyetel memori untuk mencegah IO:BufFileRead dan IO:BufFileWrite

Dalam situasi tertentu, Anda perlu menyetel memori. Tujuannya adalah menyeimbangkan persyaratan berikut:

- Nilai `work_mem` (lihat Area memori kerja)
- Memori yang tersisa setelah memangkas nilai `shared_buffers` (lihat Pool buffer)

- Koneksi maksimum yang dibuka dan digunakan, yang dibatasi oleh `max_connections`

Meningkatkan ukuran area memori kerja

Dalam beberapa situasi, satu-satunya pilihan adalah menambah memori yang digunakan oleh sesi Anda. Jika kueri Anda ditulis dengan benar dan menggunakan kunci yang benar untuk bergabung, pertimbangkan untuk meningkatkan nilai `work_mem`. Untuk informasi selengkapnya, lihat Area memori kerja.

Untuk mengetahui jumlah file sementara yang dihasilkan kueri, atur `log_temp_files` ke `0`. Jika meningkatkan nilai `work_mem` ke nilai maksimum yang diidentifikasi dalam log, Anda mencegah kueri menghasilkan file sementara. Namun, `work_mem` menetapkan nilai maksimum per simpul rencana untuk setiap koneksi atau pekerja paralel. Jika basis data memiliki 5.000 koneksi, dan jika masing-masing menggunakan memori 256 MiB, mesin akan membutuhkan RAM 1,2 TiB. Karena itu, instans Anda dapat kehabisan memori.

Mencadangkan memori yang cukup untuk pool buffer bersama

Basis data Anda menggunakan area memori seperti pool buffer bersama, bukan hanya area memori kerja. Pertimbangkan persyaratan area memori tambahan ini sebelum Anda meningkatkan `work_mem`. Untuk informasi selengkapnya tentang pool buffer, lihat Pool buffer.

Misalnya, asumsikan bahwa kelas instans Aurora PostgreSQL Anda adalah db.r5.2xlarge. Kelas ini memiliki memori 64 GiB. Secara default, 75% memori dicadangkan untuk pool buffer bersama. Setelah Anda mengurangi jumlah yang dialokasikan ke area memori bersama, 16.384 MB tetap ada. Jangan alokasikan memori yang tersisa hanya ke area memori kerja karena sistem operasi dan mesin juga memerlukan memori.

Memori yang dapat dialokasikan ke `work_mem` bergantung pada kelas instans. Jika Anda menggunakan kelas instans yang lebih besar, memori yang tersedia akan lebih banyak. Namun, dalam contoh sebelumnya, Anda tidak dapat menggunakan lebih dari 16 GiB. Jika melakukannya, instans Anda menjadi tidak tersedia saat kehabisan memori. Untuk memulihkan instans dari status tidak tersedia, layanan otomatisasi Aurora PostgreSQL secara otomatis dimulai ulang.

Mengelola jumlah koneksi

Misalnya, instans basis data Anda memiliki 5.000 koneksi bersama. Setiap koneksi menggunakan setidaknya 4 MiB `work_mem`. Konsumsi memori yang tinggi dari koneksi cenderung menurunkan performa. Untuk itu, Anda memiliki opsi berikut:

- Tingkatkan ke kelas instans yang lebih besar.
- Kurangi jumlah koneksi basis data bersama dengan menggunakan proksi koneksi atau pooler.

Untuk proksi, pertimbangkan Proksi Amazon RDS, pgBouncer, atau pooler koneksi berdasarkan aplikasi Anda. Solusi ini mengurangi beban CPU. Solusi ini juga mengurangi risiko saat semua koneksi memerlukan area memori kerja. Saat koneksi basis data tersedia lebih sedikit, Anda dapat meningkatkan nilai `work_mem`. Dengan cara ini, Anda mengurangi munculnya peristiwa tunggu `IO:BufFileRead` dan `IO:BufFileWrite`. Selain itu, kueri yang menunggu area memori kerja dipercepat secara signifikan.

## IO: DataFileRead

Peristiwa `IO:DataFileRead` terjadi saat koneksi menunggu proses backend untuk membaca halaman yang diperlukan dari penyimpanan karena halaman tidak tersedia dalam memori bersama.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


### Versi mesin yang didukung

Informasi acara tunggu ini didukung untuk semua versi Aurora Postgre. SQL

### Konteks

Semua kueri dan operasi manipulasi data (DML) mengakses halaman di kumpulan buffer. Pernyataan yang dapat menimbulkan pembacaan mencakup `SELECT`, `UPDATE`, dan `DELETE`. Misalnya, `UPDATE` dapat membaca halaman dari tabel atau indeks. Jika halaman yang diminta atau diperbarui tidak berada dalam pool buffer bersama, pembacaan ini dapat mengarah ke peristiwa `IO:DataFileRead`.

Karena bersifat terbatas, pool buffer bersama dapat diisi. Dalam hal ini, permintaan untuk halaman yang tidak berada dalam memori memaksa basis data untuk membaca blok dari disk. Jika peristiwa `IO:DataFileRead` sering terjadi, pool buffer bersama mungkin terlalu kecil untuk mengakomodasi beban kerja Anda. Masalah ini bersifat akut untuk kueri `SELECT` yang membaca sejumlah besar baris

yang tidak sesuai dengan pool buffer. Untuk informasi selengkapnya tentang pool buffer, lihat Pool buffer.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab umum peristiwa `IO:DataFileRead` tersebut mencakup:

Lonjakan koneksi

Anda mungkin menemukan beberapa koneksi yang menghasilkan jumlah acara IO: DataFileRead wait yang sama. Dalam hal ini, lonjakan (peningkatan tiba-tiba dan besar) dalam peristiwa `IO:DataFileRead` dapat terjadi.

SELECTdan DML pernyataan yang melakukan pemindaian berurutan

Aplikasi Anda mungkin melakukan operasi baru. Operasi yang ada mungkin juga berubah karena rencana eksekusi baru. Dalam kasus ini, cari tabel (terutama tabel besar) yang memiliki nilai `seq_scan` yang lebih besar. Temukan tabel dengan membuat kueri `pg_stat_user_tables`. Untuk melacak kueri yang menghasilkan lebih banyak operasi baca, gunakan ekstensi `pg_stat_statements`.

CTASdan CREATE INDEX untuk kumpulan data yang besar

A CTASadalah `CREATE TABLE AS SELECT` pernyataan. Jika Anda menjalankan CTAS menggunakan kumpulan data besar sebagai sumber, atau membuat indeks pada tabel besar, `IO:DataFileRead` peristiwa dapat terjadi. Saat Anda membuat indeks, basis data mungkin perlu membaca seluruh objek menggunakan pemindaian berurutan. A CTAS menghasilkan `IO:DataFile` pembacaan saat halaman tidak ada dalam memori.

Beberapa pekerja vakum berjalan pada waktu yang sama

Pekerja vakum dapat dipicu secara manual atau otomatis. Sebaiknya adopsi strategi vakum yang agresif. Namun, saat tabel memiliki banyak baris yang diperbarui atau dihapus, peristiwa tunggu `IO:DataFileRead` bertambah. Setelah ruang direklamasi, waktu vakum yang dihabiskan untuk `IO:DataFileRead` akan berkurang.

Menyerap data dalam jumlah besar

Saat aplikasi Anda menyerap data dalam jumlah besar, operasi `ANALYZE` mungkin terjadi lebih sering. Proses `ANALYZE` dapat dipicu oleh peluncur autovacuum atau diinvokasi secara manual.

Operasi `ANALYZE` membaca subset dari tabel. Jumlah halaman yang harus dipindai dihitung menggunakan perkalian 30 dengan nilai `default_statistics_target`. Untuk informasi lebih

lanjut, lihat dokumentasi Postgre SQL. Parameter `default_statistics_target` menerima nilai antara 1 hingga 10.000, dengan nilai default adalah 100.

Kekurangan sumber daya

Jika bandwidth jaringan instance atau CPU dikonsumsi, `IO:DataFileRead` peristiwa mungkin terjadi lebih sering.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Memeriksa filter predikat untuk kueri yang menghasilkan peristiwa tunggu
- Meminimalkan efek operasi pemeliharaan
- Merespons jumlah koneksi yang tinggi

### Memeriksa filter predikat untuk kueri yang menghasilkan peristiwa tunggu

Asumsikan bahwa Anda mengidentifikasi kueri spesifik yang menghasilkan peristiwa tunggu `IO:DataFileRead`. Anda dapat mengidentifikasinya menggunakan teknik berikut:

- Wawasan Performa
- Tampilan katalog seperti yang disediakan oleh ekstensi `pg_stat_statements`
- Tampilan katalog `pg_stat_all_tables`, jika secara berkala menunjukkan peningkatan jumlah pembacaan fisik
- Tampilan `pg_statio_all_tables`, jika menunjukkan bahwa penghitung `_read` meningkat

Sebaiknya Anda menentukan filter yang akan digunakan dalam predikat (klausa `WHERE`) kueri ini. Ikuti pedoman berikut:

- Jalankan perintah `EXPLAIN`. Pada output, identifikasi jenis pemindaian yang digunakan. Pemindaian berurutan tidak selalu menunjukkan adanya masalah. Kueri yang menggunakan pemindaian berurutan secara alami menghasilkan lebih banyak peristiwa `IO:DataFileRead` jika dibandingkan dengan kueri yang menggunakan filter.

  Cari tahu apakah kolom yang tercantum dalam klausa `WHERE` telah diindeks. Jika tidak, coba buat indeks untuk kolom ini. Pendekatan ini mencegah pemindaian berurutan dan mengurangi peristiwa

`IO:DataFileRead`. Jika kueri memiliki filter yang ketat dan masih menghasilkan pemindaian berurutan, evaluasi apakah indeks yang tepat sedang digunakan.

- Cari tahu apakah kueri mengakses tabel yang sangat besar. Dalam beberapa kasus, partisi tabel dapat meningkatkan performa, dengan memungkinkan kueri hanya membaca partisi yang diperlukan.
- Periksa kardinalitas (jumlah total baris) dari operasi gabungan Anda. Perhatikan seberapa ketat nilai yang Anda teruskan di filter untuk klausa `WHERE` Anda. Jika memungkinkan, setel kueri Anda untuk mengurangi jumlah baris yang diteruskan di setiap langkah rencana.

Meminimalkan efek operasi pemeliharaan

Operasi pemeliharaan seperti `VACUUM` dan `ANALYZE` bersifat penting. Sebaiknya jangan dinonaktifkan karena peristiwa tunggu `IO:DataFileRead` berkaitan dengan operasi pemeliharaan ini. Pendekatan berikut dapat meminimalkan efek operasi ini:

- Jalankan operasi pemeliharaan secara manual selama di luar jam sibuk. Teknik ini mencegah basis data mencapai ambang batas untuk operasi otomatis.
- Untuk tabel yang sangat besar, pertimbangkan untuk mempartisi tabel. Teknik ini mengurangi overhead operasi pemeliharaan. Basis data hanya mengakses partisi yang membutuhkan pemeliharaan.
- Saat Anda menyerap data dalam jumlah besar, coba nonaktifkan fitur analisis otomatis.

Fitur autovacuum secara otomatis dipicu untuk tabel saat rumus berikut benar.

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

Tampilan `pg_stat_user_tables` dan katalog `pg_class` berisi beberapa baris. Satu baris dapat sesuai dengan satu baris di tabel Anda. Rumus ini mengasumsikan bahwa `reltuples` adalah untuk tabel tertentu. Parameter `autovacuum_vacuum_scale_factor` (0,20 secara default) dan `autovacuum_vacuum_threshold` (50 tuple secara default) biasanya diatur secara global untuk seluruh instans. Namun, Anda dapat mengatur nilai yang berbeda untuk tabel tertentu.

Topik

- Menemukan tabel yang mengonsumsi ruang secara tidak perlu
- Menemukan indeks yang mengonsumsi ruang yang tidak perlu

- Menemukan tabel yang memenuhi syarat untuk autovacuum

Menemukan tabel yang mengonsumsi ruang secara tidak perlu

Untuk menemukan tabel yang mengonsumsi ruang lebih dari yang diperlukan, jalankan kueri berikut. Saat dijalankan oleh peran pengguna basis data yang tidak memiliki peran `rds_superuser`, kueri ini hanya menampilkan informasi tentang tabel yang diizinkan dibaca oleh peran pengguna. Kueri ini didukung oleh Postgre SQL versi 12 dan versi yang lebih baru.

```
WITH report AS (
   SELECT   schemaname
           ,tblname
           ,n_dead_tup
           ,n_live_tup
           ,block_size*tblpages AS real_size
           ,(tblpages-est_tblpages)*block_size AS extra_size
           ,CASE WHEN tblpages - est_tblpages > 0
              THEN 100 * (tblpages - est_tblpages)/tblpages::float
              ELSE 0
            END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
           ,CASE WHEN tblpages - est_tblpages_ff > 0
              THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
              ELSE 0
            END AS bloat_ratio
           ,is_na
    FROM (
          SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                 ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                 ,tblpages
                 ,fillfactor
                 ,block_size
                 ,tblid
                 ,schemaname
                 ,tblname
                 ,n_dead_tup
                 ,n_live_tup
                 ,heappages
                 ,toastpages
                 ,is_na
```

```
             FROM (
                    SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
                               - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
 tpl_hdr_size%ma END
                               - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
 ceil(tpl_data_size)::int%ma END
                            ) AS tpl_size
                            ,block_size - page_hdr AS size_per_block
                            ,(heappages + toastpages) AS tblpages
                            ,heappages
                            ,toastpages
                            ,reltuples
                            ,toasttuples
                            ,block_size
                            ,page_hdr
                            ,tblid
                            ,schemaname
                            ,tblname
                            ,fillfactor
                            ,is_na
                            ,n_dead_tup
                            ,n_live_tup
                           FROM (
                                 SELECT  tbl.oid                       AS tblid
                                        ,ns.nspname                    AS schemaname
                                        ,tbl.relname                   AS tblname
                                        ,tbl.reltuples                 AS reltuples
                                        ,tbl.relpages                  AS heappages
                                        ,coalesce(toast.relpages, 0)   AS toastpages
                                        ,coalesce(toast.reltuples, 0)  AS toasttuples
                                        ,psat.n_dead_tup               AS n_dead_tup
                                        ,psat.n_live_tup               AS n_live_tup
                                        ,24                            AS page_hdr
                                        ,current_setting('block_size')::numeric AS
 block_size

 ,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
 'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
                                        ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                        ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END              AS tpl_hdr_size
                                        ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                  AS tpl_data_size
```

```
                                      ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)         AS is_na
                                  FROM  pg_attribute       AS att
                                  JOIN  pg_class           AS tbl    ON (att.attrelid =
 tbl.oid)
                                  JOIN  pg_stat_all_tables AS psat   ON (tbl.oid =
 psat.relid)
                                  JOIN  pg_namespace       AS ns     ON (ns.oid =
 tbl.relnamespace)
                             LEFT JOIN  pg_stats           AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                             LEFT JOIN  pg_class           AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                 WHERE  att.attnum > 0
                                   AND  NOT att.attisdropped
                                   AND  tbl.relkind = 'r'
                                GROUP BY  tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                                ORDER BY  schemaname, tblname
                            ) AS s
                ) AS s2
       ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

Anda dapat memeriksa bloat indeks dan tabel di aplikasi Anda. Untuk informasi selengkapnya, lihat Mendiagnosis bloat tabel dan indeks.

Menemukan indeks yang mengonsumsi ruang yang tidak perlu

Untuk menemukan indeks yang mengonsumsi ruang yang tidak perlu, jalankan kueri berikut.

```
-- WARNING: run with a nonsuperuser role, the query inspects
```

```
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
```

```
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
      SELECT
        i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
        current_setting('block_size')::numeric AS bs, fillfactor,
        CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
          WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
          ELSE 4
        END AS maxalign,
        /* per page header, fixed size: 20 for 7.X, 24 for others */
        24 AS pagehdr,
        /* per page btree opaque data */
        16 AS pageopqdata,
        /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
        CASE WHEN max(coalesce(s.null_frac,0)) = 0
          THEN 2 -- IndexTupleData size
          ELSE 2 + (( 32 + 8 - 1 ) / 8)
          -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
        END AS index_tuple_hdr_bm,
        /* data len: we remove null values save space using it fractionnal part from
stats */
        sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
        max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
      FROM pg_attribute AS a
        JOIN (
          SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
            idx.reltuples, idx.relpages, idx.relam,
            indrelid, indexrelid, indkey::smallint[] AS attnum,
            coalesce(substring(
              array_to_string(idx.reloptions, ' ')
               from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
          FROM pg_index
            JOIN pg_class idx ON idx.oid=pg_index.indexrelid
            JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
            JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
          WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
        ) AS i ON a.attrelid = i.indexrelid
        JOIN pg_stats AS s ON s.schemaname = i.nspname
```

```
          AND ((s.tablename = i.tblname AND s.attname =
 pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
          -- stats from tbl
          OR  (s.tablename = i.idxname AND s.attname = a.attname))
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

Menemukan tabel yang memenuhi syarat untuk autovacuum

Untuk menemukan tabel yang memenuhi syarat untuk autovacuum, jalankan kueri berikut.

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
        coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
        as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
```

```
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

Merespons jumlah koneksi yang tinggi

Saat Anda memantau Amazon CloudWatch, Anda mungkin menemukan bahwa `DatabaseConnections` metrik melonjak. Peningkatan ini menunjukkan bertambahnya jumlah koneksi ke basis data Anda. Sebaiknya lakukan pendekatan berikut:

- Membatasi jumlah koneksi yang dapat dibuka aplikasi dengan setiap instans. Jika aplikasi Anda memiliki fitur kumpulan koneksi tertanam, tetapkan jumlah koneksi yang wajar. Dasarkan angka pada apa yang vCPUs dalam contoh Anda dapat paralelkan secara efektif.

  Jika aplikasi Anda tidak menggunakan fitur kumpulan koneksi, pertimbangkan untuk menggunakan Amazon RDS Proxy atau alternatifnya. Pendekatan ini memungkinkan aplikasi Anda membuka beberapa koneksi dengan penyeimbang beban. Penyeimbang selanjutnya dapat membuka sejumlah koneksi terbatas dengan basis data. Karena lebih sedikit koneksi yang berjalan secara paralel, instans DB Anda melakukan lebih sedikit peralihan konteks di kernel. Kueri harus berkembang lebih cepat, yang mengarah ke lebih sedikit peristiwa tunggu. Untuk informasi selengkapnya, lihat Proksi Amazon RDS untuk Aurora.
- Bila memungkinkan, manfaatkan node pembaca untuk Aurora Postgre SQL dan baca replika untuk Postgre. RDS SQL Saat aplikasi Anda menjalankan operasi hanya-baca, kirim permintaan ini ke

titik akhir khusus pembaca. Teknik ini menyebarkan permintaan aplikasi di semua simpul pembaca, sehingga mengurangi tekanan I/O pada simpul penulis.

- Coba naikkan skala instans DB Anda. Kelas instance berkapasitas lebih tinggi memberikan lebih banyak memori, yang memberi Aurora Postgre kumpulan buffer bersama SQL yang lebih besar untuk menampung halaman. Ukuran yang lebih besar juga memberi instans DB lebih banyak vCPUs untuk menangani koneksi. Lebih vCPUs banyak sangat membantu ketika operasi yang menghasilkan acara `IO:DataFileRead` tunggu ditulis.

# IO: XactSync

Peristiwa `IO:XactSync` terjadi saat basis data menunggu subsistem penyimpanan Aurora mengakui commit transaksi reguler, atau commit ataupun rollback transaksi yang disiapkan. Transaksi yang disiapkan adalah bagian dari dukungan PostgreSQL untuk commit dua fase. Peristiwa ini juga dapat terjadi ketika kueri menunggu transaksi lain untuk dilakukan, terutama dalam kasus di mana komit otomatis dimatikan. Dalam skenario seperti itu, pembaruan mungkin tampak menunggu XactSync meskipun belum dilakukan.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

## Konteks

Peristiwa `IO:XactSync` menunjukkan bahwa instans menghabiskan waktu menunggu subsistem penyimpanan Aurora untuk mengkonfirmasi bahwa data transaksi telah diproses.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `IO:XactSync` ditampilkan lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Saturasi jaringan

Lalu lintas antara klien dan instans DB atau lalu lintas ke subsistem penyimpanan mungkin terlalu berat untuk bandwidth jaringan.

Tekanan CPU

Beban kerja yang berat mungkin mencegah daemon penyimpanan Aurora mendapatkan waktu CPU yang cukup.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Memantau sumber daya
- Menaikkan skala CPU
- Meningkatkan bandwidth jaringan
- Mengurangi jumlah commit

### Memantau sumber daya

Untuk menentukan penyebab peristiwa `IO:XactSync` yang meningkat, periksa metrik berikut:

- `WriteThroughput` dan `CommitThroughput` – Perubahan throughput tulis atau throughput commit dapat menunjukkan peningkatan beban kerja.
- `WriteLatency` dan `CommitLatency` – Perubahan latensi tulis atau latensi commit dapat menunjukkan bahwa subsistem penyimpanan diminta untuk melakukan lebih banyak pekerjaan.
- `CPUUtilization` – Jika penggunaan CPU instans di atas 90%, daemon penyimpanan Aurora mungkin tidak mendapatkan cukup waktu pada CPU. Dalam hal ini, performa I/O menurun.

Untuk informasi selengkapnya tentang metrik ini, lihat Metrik tingkat instans untuk Amazon Aurora.

### Menaikkan skala CPU

Untuk mengatasi masalah kekurangan CPU, coba ubah ke jenis instans dengan kapasitas CPU yang lebih besar. Untuk informasi tentang kapasitas CPU untuk kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Meningkatkan bandwidth jaringan

Untuk menentukan apakah instans mencapai batas bandwidth jaringannya, periksa peristiwa tunggu lainnya berikut:

- `IO:DataFileRead`, `IO:BufferRead`, `IO:BufferWrite`, dan `IO:XactWrite` – Kueri yang menggunakan I/O dalam jumlah besar dapat membuat lebih banyak peristiwa tunggu ini.
- `Client:ClientRead` dan `Client:ClientWrite` – Kueri dengan komunikasi klien dalam jumlah besar dapat membuat lebih banyak peristiwa tunggu ini.

Jika masalahnya ada pada bandwidth jaringan, coba ubah ke jenis instans dengan bandwidth jaringan yang lebih banyak. Untuk informasi tentang performa jaringan untuk kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Mengurangi jumlah commit

Untuk mengurangi jumlah commit, gabungkan pernyataan ke dalam blok transaksi.

## IPC:DamRecordTxAck

Peristiwa `IPC:DamRecordTxAck` terjadi saat Aurora PostgreSQL dalam sesi yang menggunakan aliran aktivitas basis data menghasilkan peristiwa aliran aktivitas, lalu menunggu peristiwa tersebut menjadi tahan lama.

Topik

- Versi mesin yang relevan
- Konteks
- Penyebab
- Tindakan

### Versi mesin yang relevan

Informasi peristiwa tunggu ini relevan untuk semua Aurora PostgreSQL 10.7 dan versi 10 yang lebih tinggi, 11.4 dan versi 11 yang lebih tinggi, serta semua versi 12 dan 13.

### Konteks

Dalam mode sinkron, daya tahan peristiwa aliran aktivitas lebih disukai daripada performa basis data. Saat menunggu penulisan peristiwa yang tahan lama, sesi akan memblokir aktivitas basis data lainnya, sehingga menyebabkan peristiwa tunggu `IPC:DamRecordTxAck`.

### Penyebab

Penyebab paling umum munculnya peristiwa `IPC:DamRecordTxAck` dalam peristiwa tunggu teratas adalah bahwa fitur DAS (Aliran Aktivitas Basis Data) merupakan audit holistik. Aktivitas SQL yang lebih tinggi menghasilkan peristiwa aliran aktivitas yang perlu dicatat.

### Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda:

- Kurangi jumlah pernyataan SQL atau nonaktifkan aliran aktivitas basis data. Tindakan ini akan mengurangi jumlah peristiwa yang membutuhkan penulisan yang tahan lama.
- Ubah ke mode asinkron. Tindakan ini akan membantu mengurangi pertentangan pada peristiwa tunggu `IPC:DamRecordTxAck`.

  Namun, fitur DAS tidak dapat menjamin daya tahan setiap peristiwa dalam mode asinkron.

## IPC: Acara tunggu paralel

Berikut ini `IPC:parallel wait events` menunjukkan bahwa sesi sedang menunggu komunikasi antar-proses yang terkait dengan operasi eksekusi query paralel.

- `IPC:BgWorkerStartup`- Sebuah proses sedang menunggu proses pekerja paralel untuk menyelesaikan urutan startup-nya. Ini terjadi ketika menginisialisasi pekerja untuk eksekusi kueri paralel.
- `IPC:BgWorkerShutdown`- Sebuah proses sedang menunggu proses parallel worker untuk menyelesaikan urutan shutdown-nya. Ini terjadi selama fase pembersihan eksekusi query paralel.
- `IPC:ExecuteGather`- Sebuah proses menunggu untuk menerima data dari proses pekerja paralel selama eksekusi kueri. Ini terjadi ketika proses pemimpin perlu mengumpulkan hasil dari pekerjanya.

- `IPC:ParallelFinish`- Sebuah proses sedang menunggu pekerja paralel untuk menyelesaikan eksekusi mereka dan melaporkan hasil akhir mereka. Ini terjadi selama fase penyelesaian eksekusi query paralel.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

## Konteks

Eksekusi query paralel di PostgreSQL melibatkan beberapa proses yang bekerja sama untuk memproses satu query. Ketika kueri ditentukan cocok untuk paralelisasi, proses pemimpin berkoordinasi dengan satu atau lebih proses pekerja paralel berdasarkan pengaturan parameter. `max_parallel_workers_per_gather` Proses pemimpin membagi pekerjaan di antara pekerja, setiap pekerja memproses porsi datanya secara independen, dan hasilnya dikumpulkan kembali ke proses pemimpin.

> **Note**
>
> Setiap pekerja paralel beroperasi sebagai proses terpisah dengan persyaratan sumber daya yang mirip dengan sesi pengguna penuh. Ini berarti query paralel dengan 4 pekerja dapat mengkonsumsi hingga 5 kali sumber daya (CPU, memori, I/O bandwidth) dibandingkan dengan kueri non-paralel, karena proses pemimpin dan setiap proses pekerja mempertahankan alokasi sumber daya mereka sendiri. Misalnya, pengaturan seperti diterapkan `work_mem` secara individual ke setiap pekerja, berpotensi mengalikan total penggunaan memori di semua proses.

Arsitektur query paralel terdiri dari tiga komponen utama:

- Proses pemimpin: Proses utama yang memulai operasi paralel, membagi beban kerja, dan berkoordinasi dengan proses pekerja.
- Proses pekerja: Proses latar belakang yang mengeksekusi bagian dari kueri secara paralel.
- Gather/Gathering merge: Operasi yang menggabungkan hasil dari beberapa proses pekerja kembali ke pemimpin

Selama eksekusi paralel, proses perlu berkomunikasi satu sama lain melalui mekanisme Inter-Process Communication (IPC). Peristiwa tunggu IPC ini terjadi selama fase yang berbeda:

- Startup pekerja: Ketika pekerja paralel sedang diinisialisasi
- Pertukaran data: Ketika pekerja memproses data dan mengirimkan hasil kepada pemimpin
- Worker shutdown: Ketika eksekusi paralel selesai dan pekerja dihentikan
- Titik sinkronisasi: Ketika proses perlu mengoordinasikan atau menunggu proses lain untuk menyelesaikan tugasnya

Memahami peristiwa tunggu ini sangat penting untuk mendiagnosis masalah kinerja yang terkait dengan eksekusi kueri paralel, terutama di lingkungan konkurensi tinggi di mana beberapa kueri paralel dapat dijalankan secara bersamaan.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa faktor dapat berkontribusi pada peningkatan acara tunggu IPC terkait paralel:

Konkurensi tinggi dari query paralel

Ketika banyak query paralel berjalan secara bersamaan, hal itu dapat menyebabkan pertentangan sumber daya dan peningkatan waktu tunggu untuk operasi IPC. Ini sangat umum dalam sistem dengan volume transaksi tinggi atau beban kerja analitis.

Rencana kueri paralel suboptimal

Jika perencana kueri memilih rencana paralel yang tidak efisien, hal itu dapat mengakibatkan paralelisasi yang tidak perlu atau distribusi kerja yang buruk di antara pekerja. Hal ini dapat menyebabkan peningkatan penantian IPC, terutama untuk acara IPC: ExecuteGather dan IPC:. ParallelFinish Masalah perencanaan ini sering berasal dari statistik yang sudah ketinggalan zaman dan table/index kembung.

Sering startup dan shutdown pekerja paralel

Pertanyaan berumur pendek yang sering memulai dan menghentikan pekerja paralel dapat menyebabkan peningkatan dan peristiwa. `IPC:BgWorkerStartup` `IPC:BgWorkerShutdown` Ini sering terlihat pada beban kerja OLTP dengan banyak kueri kecil yang dapat diparalelkan.

Kendala sumber daya

CPU, memori, atau I/O kapasitas yang terbatas dapat menyebabkan kemacetan dalam eksekusi paralel, yang menyebabkan peningkatan waktu tunggu di semua peristiwa IPC. Misalnya, jika CPU jenuh, proses pekerja mungkin membutuhkan waktu lebih lama untuk memulai atau memproses bagian pekerjaan mereka.

Struktur kueri yang kompleks

Kueri dengan beberapa tingkat paralelisme (misalnya, gabungan paralel diikuti oleh agregasi paralel) dapat menyebabkan pola IPC yang lebih kompleks dan berpotensi meningkatkan waktu tunggu, terutama untuk acara. `IPC:ExecuteGather`

Set hasil besar

Kueri yang menghasilkan kumpulan hasil yang besar dapat menyebabkan peningkatan waktu `IPC:ExecuteGather` tunggu karena proses pemimpin menghabiskan lebih banyak waktu untuk mengumpulkan dan memproses hasil dari proses pekerja.

Memahami faktor-faktor ini dapat membantu dalam mendiagnosis dan mengatasi masalah kinerja yang terkait dengan eksekusi kueri paralel di Aurora PostgreSQL.

## Tindakan

Ketika Anda melihat waits terkait dengan query paralel, biasanya berarti bahwa proses backend sedang mengkoordinasikan atau menunggu proses parallel worker. Penantian ini biasa terjadi selama pelaksanaan rencana paralel. Anda dapat menyelidiki dan mengurangi dampak menunggu ini dengan memantau penggunaan pekerja paralel, meninjau pengaturan parameter, dan menyetel eksekusi kueri dan alokasi sumber daya.

Topik

- Menganalisis rencana kueri untuk paralelisme yang tidak efisien
- Pantau penggunaan kueri paralel
- Tinjau dan sesuaikan pengaturan kueri paralel

- Optimalkan alokasi sumber daya
- Selidiki manajemen koneksi
- Meninjau dan mengoptimalkan operasi pemeliharaan
- Memanfaatkan Query Plan Management (QPM)

Menganalisis rencana kueri untuk paralelisme yang tidak efisien

Eksekusi kueri paralel seringkali dapat menyebabkan ketidakstabilan sistem, lonjakan CPU, dan varians kinerja kueri yang tidak dapat diprediksi. Sangat penting untuk menganalisis secara menyeluruh apakah paralelisme benar-benar meningkatkan beban kerja spesifik Anda. Gunakan EXPLORE ANALYZE untuk meninjau rencana eksekusi query paralel.

Nonaktifkan paralelisme sementara di tingkat sesi untuk membandingkan efisiensi rencana:

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
```

Aktifkan kembali paralelisme dan bandingkan:

```
RESET max_parallel_workers_per_gather;
EXPLAIN ANALYZE <your_query>;
```

Jika menonaktifkan paralelisme menghasilkan hasil yang lebih baik atau lebih konsisten, pertimbangkan untuk menonaktifkannya untuk kueri tertentu di tingkat sesi menggunakan perintah SET. Untuk dampak yang lebih luas, Anda mungkin ingin menonaktifkan paralelisme di tingkat instans dengan menyesuaikan parameter yang relevan di cluster atau grup parameter instance Anda. Untuk informasi selengkapnya, lihat Parameter Amazon Aurora PostgreSQL.

Pantau penggunaan kueri paralel

Gunakan kueri berikut untuk mendapatkan visibilitas ke aktivitas dan kapasitas kueri paralel:

Periksa proses pekerja paralel aktif:

```
SELECT
```

```
    COUNT(*)
FROM
    pg_stat_activity
WHERE
    backend_type = 'parallel worker';
```

Kueri ini menunjukkan jumlah proses pekerja paralel aktif. Nilai tinggi mungkin menunjukkan bahwa `max_parallel_workers` Anda dikonfigurasi dengan nilai tinggi dan Anda mungkin ingin mempertimbangkan untuk menguranginya.

Periksa kueri paralel bersamaan:

```
SELECT
    COUNT(DISTINCT leader_pid)
FROM
    pg_stat_activity
WHERE
    leader_pid IS NOT NULL;
```

Kueri ini mengembalikan jumlah proses pemimpin berbeda yang telah meluncurkan query paralel. Angka yang tinggi di sini menunjukkan bahwa beberapa sesi menjalankan query paralel secara bersamaan, yang dapat meningkatkan permintaan pada CPU dan memori.

Tinjau dan sesuaikan pengaturan kueri paralel

Tinjau parameter berikut untuk memastikan parameter tersebut selaras dengan beban kerja Anda:

- `max_parallel_workers`: Jumlah total pekerja paralel di semua sesi.
- `max_parallel_workers_per_gather`: Pekerja maks per kueri.

Untuk beban kerja OLAP, meningkatkan nilai-nilai ini dapat meningkatkan kinerja. Untuk beban kerja OLTP, nilai yang lebih rendah umumnya lebih disukai.

```
SHOW max_parallel_workers;
SHOW max_parallel_workers_per_gather;
```

Optimalkan alokasi sumber daya

Pantau pemanfaatan CPU dan pertimbangkan untuk menyesuaikan jumlah v CPUs jika secara konsisten tinggi dan jika aplikasi Anda mendapat manfaat dari query paralel. Pastikan memori yang memadai tersedia untuk operasi paralel.

- Gunakan metrik Performance Insights untuk menentukan apakah sistem terikat CPU.
- Setiap pekerja paralel menggunakan miliknya sendiri`work_mem`. Pastikan penggunaan memori total berada dalam batas instans.

Query paralel dapat mengkonsumsi sumber daya yang jauh lebih banyak daripada query non-paralel, karena setiap proses pekerja adalah proses yang benar-benar terpisah yang memiliki dampak yang kira-kira sama pada sistem sebagai sesi pengguna tambahan. Ini harus diperhitungkan saat memilih nilai untuk pengaturan ini, serta saat mengonfigurasi pengaturan lain yang mengontrol pemanfaatan sumber daya, seperti. `work_mem` Untuk informasi selengkapnya, lihat Dokumentasi PostgreSQL. Batas sumber daya seperti `work_mem` diterapkan secara individual untuk setiap pekerja, yang berarti total pemanfaatan mungkin jauh lebih tinggi di semua proses daripada biasanya untuk setiap proses tunggal.

Pertimbangkan untuk meningkatkan v CPUs atau menyetel parameter memori jika beban kerja Anda sangat paralel.

Selidiki manajemen koneksi

Jika mengalami kelelahan koneksi, tinjau strategi penyatuan koneksi aplikasi. Pertimbangkan untuk menerapkan penyatuan koneksi di tingkat aplikasi jika belum digunakan.

Meninjau dan mengoptimalkan operasi pemeliharaan

Mengkoordinasikan pembuatan indeks dan tugas pemeliharaan lainnya untuk mencegah pertentangan sumber daya. Pertimbangkan untuk menjadwalkan operasi ini selama jam-jam di luar sibuk. Hindari penjadwalan pemeliharaan berat (misalnya, pembuatan indeks paralel) selama periode beban kueri pengguna yang tinggi. Operasi ini dapat mengkonsumsi pekerja paralel dan memengaruhi kinerja untuk kueri reguler.

Memanfaatkan Query Plan Management (QPM)

Di Aurora PostgreSQL, fitur Query Plan Management (QPM) dirancang untuk memastikan kemampuan beradaptasi dan stabilitas rencana, terlepas dari perubahan lingkungan database

yang dapat menyebabkan regresi rencana kueri. Untuk informasi selengkapnya, lihat Gambaran umum manajemen rencana kueri Aurora PostgreSQL .QPM menyediakan beberapa kontrol atas pengoptimal. Tinjau rencana yang disetujui di QPM untuk memastikan mereka selaras dengan pengaturan paralelisme saat ini. Perbarui atau hapus rencana usang yang mungkin memaksa eksekusi paralel suboptimal.

Anda juga dapat memperbaiki rencana menggunakan pg_hint_plan. Untuk informasi selengkapnya, lihat Memperbaiki rencana menggunakan pg_hint_plan. Anda dapat menggunakan petunjuk bernama `Parallel` untuk menegakkan eksekusi paralel. Untuk informasi selengkapnya, lihat Petunjuk untuk rencana paralel.

## Lock:advisory

Peristiwa `Lock:advisory` terjadi saat aplikasi PostgreSQL menggunakan kunci untuk mengoordinasi aktivitas di beberapa sesi.

Topik

- Versi mesin yang relevan
- Konteks
- Penyebab
- Tindakan

### Versi mesin yang relevan

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

### Konteks

Kunci penasihat PostgreSQL adalah kunci kooperatif tingkat aplikasi yang secara eksplisit dikunci dan dibuka oleh kode aplikasi pengguna. Aplikasi dapat menggunakan kunci penasihat PostgreSQL untuk mengoordinasi aktivitas di beberapa sesi. Tidak seperti kunci biasa tingkat objek atau baris, aplikasi memiliki kontrol penuh atas masa pakai kunci. Untuk informasi selengkapnya, lihat Kunci Penasihat dalam dokumentasi PostgreSQL.

Kunci penasihat dapat dilepaskan sebelum transaksi berakhir atau disimpan oleh sesi di seluruh transaksi. Ini tidak berlaku untuk kunci implisit yang diberlakukan sistem, seperti kunci eksklusif akses pada tabel yang diperoleh dari pernyataan `CREATE INDEX`.

Untuk deskripsi fungsi yang digunakan untuk memperoleh (mengunci) dan melepaskan (membuka kunci) kunci penasihat, lihat Fungsi Kunci Penasihat dalam dokumentasi PostgreSQL.

Kunci penasihat diterapkan di atas sistem penguncian PostgreSQL biasa dan terlihat dalam tampilan sistem `pg_locks`.

## Penyebab

Jenis kunci ini secara eksklusif dikendalikan oleh aplikasi yang menggunakannya secara eksplisit. Kunci penasihat yang diperoleh untuk setiap baris sebagai bagian dari kueri dapat menyebabkan lonjakan kunci atau penumpukan jangka panjang.

Efek ini terjadi saat kueri dijalankan dengan cara yang memperoleh kunci pada lebih banyak baris daripada yang ditampilkan oleh kueri. Aplikasi pada akhirnya harus melepaskan setiap kunci, tetapi jika kunci diperoleh pada baris yang tidak ditampilkan, maka aplikasi tidak dapat menemukan semua kunci.

Contoh berikut berasal dari Kunci Penasihat dalam dokumentasi PostgreSQL.

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

Dalam contoh ini, klausa `LIMIT` hanya dapat menghentikan output kueri setelah baris dipilih secara internal dan nilai ID-nya dikunci. Hal ini dapat terjadi secara tiba-tiba saat volume data yang berkembang menyebabkan perencana memilih rencana eksekusi lain yang tidak diuji selama pengembangan. Penumpukan dalam kasus ini terjadi karena aplikasi secara eksplisit memanggil `pg_advisory_unlock` untuk setiap nilai ID yang terkunci. Namun, dalam kasus ini, aplikasi tidak dapat menemukan set kunci yang diperoleh pada baris yang tidak ditampilkan. Karena diperoleh pada tingkat sesi, kunci tidak dilepaskan secara otomatis pada akhir transaksi.

Kemungkinan penyebab lain untuk lonjakan upaya kunci yang diblokir adalah konflik yang tidak diinginkan. Dalam konflik ini, bagian aplikasi yang tidak terkait berbagi ruang ID kunci yang sama secara tidak sengaja.

## Tindakan

Tinjau penggunaan kunci penasihat oleh aplikasi dan jelaskan secara detail di mana dan kapan dalam alur aplikasi setiap jenis kunci penasihat diperoleh dan dilepaskan.

Ketahui apakah sesi memperoleh terlalu banyak kunci atau sesi yang berjalan lama tidak melepaskan kunci cukup awal, sehingga mengarah ke penumpukan kunci yang lambat. Anda dapat

memperbaiki penumpukan kunci tingkat sesi yang lambat dengan mengakhiri sesi menggunakan `pg_terminate_backend(pid)`.

Klien yang menunggu kunci penasihat muncul dalam `pg_stat_activity` dengan `wait_event_type=Lock` dan `wait_event=advisory`. Anda dapat memperoleh nilai kunci tertentu dengan membuat kueri tampilan sistem `pg_locks` untuk `pid` yang sama, dengan mencari `locktype=advisory` dan `granted=f`.

Anda kemudian dapat mengidentifikasi sesi pemblokiran dengan membuat kueri `pg_locks` untuk kunci penasihat serupa yang memiliki `granted=t`, seperti ditunjukkan dalam contoh berikut.

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
         blocking_activity.query AS blocking_statement
    FROM pg_catalog.pg_locks blocked_locks
    JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
 blocked_locks.pid
    JOIN pg_catalog.pg_locks blocking_locks
        ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM
 blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
        AND blocking_locks.pid != blocked_locks.pid
```

```
    JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
    WHERE NOT blocked_locks.GRANTED;
```

Semua fungsi API kunci penasihat memiliki dua set argumen, baik satu argumen `bigint` atau dua argumen `integer`:

- Untuk fungsi API dengan satu argumen `bigint`, 32 bit atas masuk dalam `pg_locks.classid` dan 32 bit bawah masuk dalam `pg_locks.objid`.
- Untuk fungsi API dengan dua argumen `integer`, argumen pertama adalah `pg_locks.classid` dan argumen kedua adalah `pg_locks.objid`.

Nilai `pg_locks.objsubid` menunjukkan formulir API yang digunakan: 1 berarti satu argumen `bigint`; 2 berarti dua argumen `integer`.

## Lock:extend

Peristiwa `Lock:extend` terjadi saat suatu proses backend sedang menunggu untuk mengunci relasi agar dapat diperluas, sementara proses lain mengunci relasi tersebut untuk tujuan yang sama.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

### Konteks

Peristiwa `Lock:extend` menunjukkan bahwa suatu proses backend sedang menunggu untuk memperluas relasi yang dikunci oleh proses backend lain saat relasi tersebut diperluas. Karena hanya satu proses pada satu waktu yang dapat memperluas relasi, sistem membuat peristiwa tunggu `Lock:extend`. Operasi `INSERT`, `COPY`, dan `UPDATE` dapat membuat peristiwa ini.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `Lock:extend` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Lonjakan sisipan atau pembaruan bersamaan ke tabel yang sama

Mungkin terdapat peningkatan jumlah sesi bersamaan dengan kueri yang disisipkan ke atau memperbarui tabel yang sama.

Bandwidth jaringan tidak cukup

Bandwidth jaringan pada instans DB mungkin tidak cukup untuk kebutuhan komunikasi penyimpanan dari beban kerja saat ini. Hal ini dapat berkontribusi pada latensi penyimpanan yang menyebabkan peningkatan peristiwa `Lock:extend`.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengurangi sisipan dan pembaruan bersamaan ke relasi yang sama
- Meningkatkan bandwidth jaringan

Mengurangi sisipan dan pembaruan bersamaan ke relasi yang sama

Pertama, ketahui apakah terdapat peningkatan pada metrik `tup_inserted` dan `tup_updated` dengan disertai peningkatan pada peristiwa tunggu ini. Jika demikian, periksa relasi yang memiliki pertentangan tinggi untuk operasi penyisipan dan pembaruan. Untuk menentukan hal ini, buat kueri tampilan `pg_stat_all_tables` untuk nilai pada bidang `n_tup_ins` dan `n_tup_upd`. Untuk informasi tentang tampilan `pg_stat_all_tables`, lihat pg_stat_statements pada dokumentasi PostgreSQL.

Untuk mendapatkan informasi selengkapnya tentang pemblokiran dan kueri yang diblokir, buat kueri `pg_stat_activity` seperti contoh berikut:

```
SELECT
    blocked.pid,
```

```
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
            blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-------------------------------------------------------------------
+--------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

Setelah Anda mengidentifikasi relasi yang berkontribusi untuk meningkatkan peristiwa `Lock:extend`, gunakan teknik berikut untuk mengurangi pertentangan:

- Cari tahu apakah Anda dapat menggunakan partisi untuk mengurangi pertentangan pada tabel yang sama. Memisahkan tuple yang disisipkan atau diperbarui ke dalam partisi yang berbeda dapat mengurangi pertentangan. Untuk informasi tentang partisi, lihat Mengelola partisi PostgreSQL dengan ekstensi pg_partman.
- Jika peristiwa tunggu terutama disebabkan oleh aktivitas pembaruan, pertimbangkan untuk mengurangi nilai faktor isi relasi. Ini dapat mengurangi permintaan untuk blok baru selama pembaruan. Faktor isi adalah parameter penyimpanan untuk tabel yang menentukan jumlah maksimum ruang untuk mengemas halaman tabel. Ini dinyatakan sebagai persentase dari total ruang untuk sebuah halaman. Untuk informasi selengkapnya tentang parameter faktor isi, lihat BUAT TABEL pada dokumentasi PostgreSQL.

  > **Important**
  >
  > Kami sangat merekomendasikan untuk menguji sistem jika Anda mengubah faktor isi karena mengubah nilai ini dapat berdampak negatif pada performa, tergantung pada beban kerja Anda.

Meningkatkan bandwidth jaringan

Untuk melihat apakah terdapat peningkatan latensi tulis, periksa metrik `WriteLatency` di CloudWatch. Jika ada, gunakan metrik `WriteThroughput` dan `ReadThroughput` Amazon CloudWatch untuk memantau lalu lintas terkait penyimpanan pada klaster DB. Metrik ini dapat membantu Anda menentukan apakah bandwidth jaringan cukup untuk aktivitas penyimpanan beban kerja Anda.

Jika tidak cukup, tingkatkan bandwidth jaringan Anda. Jika instans DB Anda mencapai batas bandwidth jaringan, satu-satunya cara untuk meningkatkan bandwidth adalah meningkatkan ukuran instans DB Anda.

Untuk informasi selengkapnya tentang melihat metrik CloudWatch, lihat CloudWatch Metrik Amazon untuk Amazon Aurora. Untuk informasi tentang performa jaringan untuk setiap kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

## Lock:Relation

Peristiwa `Lock:Relation` terjadi saat kueri menunggu untuk memperoleh kunci pada tabel atau tampilan (relasi) yang saat ini dikunci oleh transaksi lain.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi acara tunggu ini didukung untuk semua versi Aurora Postgre. SQL

### Konteks

Sebagian besar SQL perintah Postgre secara implisit menggunakan kunci untuk mengontrol akses bersamaan ke data dalam tabel. Anda juga dapat menggunakan kunci ini secara eksplisit dalam kode aplikasi Anda dengan perintah `LOCK`. Banyak mode kunci yang tidak kompatibel satu sama lain, dan mode ini dapat memblokir transaksi saat mencoba mengakses objek yang sama. Ketika ini terjadi,

Aurora Postgre SQL menghasilkan sebuah peristiwa. `Lock:Relation` Berikut adalah beberapa contoh umum:

- Kunci eksklusif seperti `ACCESS EXCLUSIVE` dapat memblokir semua akses konkuren. Operasi bahasa definisi data (DDL) seperti`DROP TABLE`,, `TRUNCATEVACUUM FULL`, dan `CLUSTER` memperoleh `ACCESS EXCLUSIVE` kunci secara implisit. `ACCESS EXCLUSIVE`juga merupakan mode kunci default untuk `LOCK TABLE` pernyataan yang tidak menentukan mode secara eksplisit.
- Menggunakan `CREATE INDEX (without CONCURRENT)` pada tabel bertentangan dengan pernyataan bahasa manipulasi data (DML)`UPDATE`,`DELETE`, dan`INSERT`, yang memperoleh `ROW EXCLUSIVE` kunci.

Untuk informasi selengkapnya tentang kunci tingkat tabel dan mode kunci yang bertentangan, lihat Penguncian Eksplisit dalam dokumentasi Postgre. SQL

Kueri dan transaksi yang memblokir biasanya dapat dibuka blokirnya dengan salah satu cara berikut:

- Kueri yang memblokir – Aplikasi dapat membatalkan kueri atau pengguna dapat mengakhiri proses. Mesin juga dapat memaksa kueri untuk berakhir karena batas waktu pernyataan sesi atau mekanisme deteksi deadlock.
- Transaksi yang memblokir – Transaksi berhenti memblokir saat menjalankan pernyataan `ROLLBACK` atau `COMMIT`. Rollback juga terjadi secara otomatis saat sesi diputus oleh klien atau masalah jaringan, atau diakhiri. Sesi dapat berakhir saat mesin basis data dimatikan, sistem kehabisan memori, dan sebagainya.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `Lock:Relation` terjadi lebih sering dari biasanya, hal tersebut dapat menunjukkan masalah performa. Penyebab umumnya meliputi yang berikut:

Peningkatan sesi konkuren dengan kunci tabel yang bertentangan

Mungkin ada peningkatan jumlah sesi konkuren dengan kueri yang mengunci tabel yang sama dengan mode kunci yang bertentangan.

Operasi pemeliharaan

Operasi pemeliharaan kondisi seperti `VACUUM` dan `ANALYZE` dapat secara signifikan meningkatkan jumlah kunci yang bertentangan. `VACUUM FULL` memperoleh kunci `ACCESS EXCLUSIVE`, dan `ANALYZE` memperoleh kunci `SHARE UPDATE EXCLUSIVE`. Kedua jenis kunci

tersebut dapat menyebabkan peristiwa tunggu `Lock:Relation`. Operasi pemeliharaan data aplikasi seperti menyegarkan tampilan terwujud juga dapat meningkatkan kueri dan transaksi yang diblokir.

Kunci pada instans pembaca

Mungkin ada pertentangan antara kunci relasi yang dipegang oleh penulis dan pembaca. Saat ini, hanya kunci relasi `ACCESS EXCLUSIVE` yang direplikasi ke instans pembaca. Namun, kunci relasi `ACCESS EXCLUSIVE` akan bertentangan dengan kunci relasi `ACCESS SHARE` yang dipegang oleh pembaca. Hal ini dapat menyebabkan peningkatan peristiwa tunggu relasi kunci pada pembaca.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengurangi dampak dari SQL pernyataan pemblokiran
- Minimalkan efek operasi pemeliharaan
- Periksa kunci pembaca

Mengurangi dampak dari SQL pernyataan pemblokiran

Untuk mengurangi dampak pemblokiran SQL pernyataan, ubah kode aplikasi Anda jika memungkinkan. Berikut adalah dua teknik umum untuk mengurangi pemblokiran:

- Gunakan `NOWAIT` opsi — Beberapa SQL perintah, seperti `SELECT` dan `LOCK` pernyataan, mendukung opsi ini. Arahan `NOWAIT` membatalkan kueri permintaan kunci jika kunci tidak dapat segera diperoleh. Teknik ini dapat membantu mencegah sesi yang memblokir menyebabkan penumpukan sesi yang diblokir di belakangnya.

  Misalnya: Transaksi A sedang menunggu kunci yang dipegang oleh transaksi B. Jika B meminta kunci pada tabel yang dikunci oleh transaksi C, transaksi A mungkin diblokir hingga transaksi C selesai. Namun, jika transaksi B menggunakan `NOWAIT` saat meminta kunci pada C, transaksi ini dapat gagal cepat (fail fast) dan memastikan bahwa transaksi A tidak harus menunggu tanpa batas waktu.
- Gunakan `SET lock_timeout` — Tetapkan `lock_timeout` nilai untuk membatasi waktu SQL pernyataan menunggu untuk memperoleh kunci pada relasi. Jika kunci tidak diperoleh dalam batas

waktu yang ditentukan, transaksi yang meminta kunci dibatalkan. Tetapkan nilai ini pada tingkat sesi.

### Minimalkan efek operasi pemeliharaan

Operasi pemeliharaan seperti `VACUUM` dan `ANALYZE` bersifat penting. Sebaiknya jangan dinonaktifkan karena peristiwa tunggu `Lock:Relation` berkaitan dengan operasi pemeliharaan ini. Pendekatan berikut dapat meminimalkan efek operasi ini:

- Jalankan operasi pemeliharaan secara manual di luar jam sibuk.
- Untuk mengurangi peristiwa tunggu `Lock:Relation` yang disebabkan oleh tugas autovacuum, lakukan penyetelan autovacuum yang diperlukan. Untuk informasi tentang menyetel autovacuum, lihat Bekerja dengan Postgre SQL autovacuum di Amazon di Panduan Pengguna Amazon. RDS RDS

### Periksa kunci pembaca

Anda dapat melihat bagaimana sesi bersamaan pada kunci yang mungkin dipegang oleh penulis dan pembaca yang memblokir satu sama lain. Salah satu cara untuk melakukannya adalah dengan menjalankan kueri yang mengembalikan jenis kunci dan relasi. Dalam tabel, Anda dapat menemukan urutan kueri untuk dua sesi bersamaan seperti itu, sesi penulis dan sesi pembaca.

Proses pemutaran ulang menunggu durasi `max_standby_streaming_delay` sebelum membatalkan kueri pembaca. Seperti yang ditunjukkan pada contoh, batas waktu kunci 100 md jauh di bawah default `max_standby_streaming_delay` 30 detik. Waktu kunci habis sebelum menjadi masalah.

| Acara urutan | Sesi | Perintah atau Output |
| --- | --- | --- |
| Menetapkan variabel lingkungan dipanggil READER dengan nilai yang ditentukan dan mencoba untuk terhubung ke instance DB dengan endpoint ini. | Sesi pembaca | CLIperintah:<br>`export READER=au rorapg2.1234567891 0.us-west-1.rds.am azonaws.com`<br><br>`psql -h $READER` |

| Acara urutan | Sesi | Perintah atau Output |
| --- | --- | --- |
| | | Output:<br>`psql (15devel, server 10.14)`<br>`Type "help" for help.` |
| Menetapkan variabel lingkungan dipanggil WRITER dan mencoba untuk terhubung ke instance DB dengan endpoint ini. | Sesi penulis | CLIperintah:<br>`export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com`<br>`psql -h $WRITER`<br>Output:<br>`psql (15devel, server 10.14)`<br>`Type "help" for help.` |
| Sesi penulis membuat tabel t1 pada instance penulis. | Sesi penulis | Kueri postgreSQL:<br>`postgres=> CREATE TABLE t1(b integer);`<br>`CREATE TABLE` |
| Jika tidak ada pertanyaan yang bertentangan pada penulis, ACCESS EXCLUSIVE kunci diperoleh pada penulis segera. | Sesi penulis | `ACCESS EXCLUSIVE` kunci diaktifkan |

| Acara urutan | Sesi | Perintah atau Output |
| --- | --- | --- |
| Sesi pembaca menetapkan interval batas waktu kunci 100 milidetik. | Sesi pembaca | Kueri postgreSQL:<br>`postgres=>  SET lock_timeout=100; SET` |
| Sesi pembaca mencoba membaca data dari tabel t1 pada instance pembaca. | Sesi pembaca | Kueri postgreSQL:<br>`postgres=>  SELECT * FROM t1;`<br>Contoh output:<br>`b --- (0 rows)` |
| Sesi penulis turun t1. | Sesi penulis | Kueri postgreSQL:<br>`postgres=>  BEGIN; BEGIN postgres=>  DROP TABLE t1; DROP TABLE postgres=>` |

| Acara urutan | Sesi | Perintah atau Output |
| --- | --- | --- |
| Waktu kueri habis dan dibatalkan pada pembaca. | Sesi pembaca | Kueri postgreSQL:<br>`postgres=>  SELECT * FROM t1;`<br>Contoh output:<br>`ERROR:  canceling statement due to lock timeout`<br>`LINE 1: SELECT * FROM t1;`<br>`^` |
| Untuk menentukan penyebab kesalahan. kueri sesi pembaca dan `pg_locks` `pg_stat_activity` | Sesi pembaca | Kueri postgreSQL:<br>`postgres=>  SELECT locktype, relation, mode, backend_type`<br>`postgres=>  FROM pg_locks l, pg_stat_activity t1`<br>`postgres=>  WHERE l.pid=t1.pid AND relation = 't1'::regclass::oid;` |

| Acara urutan | Sesi | Perintah atau Output |
| --- | --- | --- |
| Hasilnya menunjukkan bahwa `aurora wal replay` proses memegang `ACCESS EXCLUSIVE` kunci pada tabel t1. | Sesi pembaca | Hasil kueri:<br>`locktype \| relation \| mode \| backend_type`<br>`----------+----------+------------------+---------------`<br>`relation \| 68628525 \| AccessExclusiveLock \| aurora wal replay`<br>`(1 row)` |

## Lock:transactionid

Peristiwa `Lock:transactionid` terjadi saat transaksi sedang menunggu kunci tingkat baris.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

### Konteks

Peristiwa `Lock:transactionid` terjadi saat transaksi mencoba memperoleh kunci tingkat baris yang telah diberikan untuk transaksi yang sedang berlangsung pada saat bersamaan. Sesi yang menunjukkan peristiwa tunggu `Lock:transactionid` diblokir karena kunci ini. Setelah transaksi pemblokiran berakhir dengan pernyataan `COMMIT` atau `ROLLBACK`, transaksi yang diblokir dapat dilanjutkan.

Semantik kontrol bersamaan multiversi dari Aurora PostgreSQL menjamin bahwa pembaca tidak memblokir penulis dan penulis tidak memblokir pembaca. Agar pertentangan tingkat baris terjadi, transaksi yang memblokir dan diblokir harus mengeluarkan pernyataan yang bertentangan dari jenis berikut:

- `UPDATE`
- `SELECT … FOR UPDATE`
- `SELECT … FOR KEY SHARE`

Pernyataan `SELECT … FOR KEY SHARE` adalah kasus khusus. Basis data menggunakan klausa `FOR KEY SHARE` untuk mengoptimalkan performa integritas referensial. Kunci tingkat baris pada baris dapat memblokir perintah `INSERT`, `UPDATE`, dan `DELETE` pada tabel lain yang mereferensikan baris tersebut.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini ditampilkan lebih dari biasanya, penyebabnya biasanya pernyataan `UPDATE`, `SELECT … FOR UPDATE`, atau `SELECT … FOR KEY SHARE` yang dikombinasikan dengan kondisi berikut.

Topik

- Konkurensi tinggi
- Idle pada transaksi
- Transaksi jangka panjang

### Konkurensi tinggi

Aurora PostgreSQL dapat menggunakan semantik penguncian tingkat baris terperinci. Probabilitas pertentangan tingkat baris meningkat saat kondisi berikut terpenuhi:

- Beban kerja bersamaan tingkat tinggi berselisih untuk baris yang sama.
- Konkurensi meningkat.

### Idle pada transaksi

Terkadang kolom `pg_stat_activity.state` menunjukkan nilai `idle in transaction`. Nilai ini ditampilkan untuk sesi yang telah memulai transaksi, tetapi belum mengeluarkan `COMMIT` atau

ROLLBACK. Jika nilai pg_stat_activity.state tidak active, kueri yang ditampilkan pada pg_stat_activity adalah yang terbaru untuk diselesaikan. Sesi pemblokiran tidak secara aktif memproses kueri karena transaksi yang terbuka menahan kunci.

Jika transaksi idle memperoleh kunci tingkat baris, hal tersebut mungkin mencegah sesi lain memperolehnya. Kondisi ini menyebabkan peristiwa tunggu Lock:transactionid yang sering terjadi. Untuk mendiagnosis masalah, periksa output dari pg_stat_activity dan pg_locks.

Transaksi jangka panjang

Transaksi yang berjalan dalam jangka panjang mendapatkan kunci untuk jangka panjang. Kunci yang sudah lama dipegang ini dapat memblokir transaksi lain agar tidak berlangsung.

## Tindakan

Penguncian baris adalah pertentangan antara pernyataan UPDATE, SELECT … FOR UPDATE, atau SELECT … FOR KEY SHARE. Sebelum mencoba solusi, cari tahu kapan pernyataan ini berjalan pada baris yang sama. Gunakan informasi ini untuk memilih strategi yang dijelaskan pada bagian berikut.

Topik

- Menanggapi konkurensi tinggi
- Menanggapi transaksi idle
- Menanggapi transaksi jangka panjang

Menanggapi konkurensi tinggi

Jika masalahnya pada konkurensi, coba salah satu teknik berikut:

- Menurunkan konkurensi pada aplikasi. Misalnya, mengurangi jumlah sesi aktif.
- Menerapkan pool koneksi. Untuk mempelajari cara mengumpulkan koneksi dengan Proksi RDS, lihat Proksi Amazon RDS untuk Aurora.
- Desain aplikasi atau model data untuk menghindari perselisihan pernyataan UPDATE dan SELECT … FOR UPDATE. Anda juga dapat mengurangi jumlah kunci asing yang diakses oleh pernyataan SELECT … FOR KEY SHARE.

Menanggapi transaksi idle

Jika `pg_stat_activity.state` menampilkan `idle in transaction`, gunakan strategi berikut:

- Aktifkan autocommit saat memungkinkan. Pendekatan ini mencegah transaksi memblokir transaksi lain sambil menunggu `COMMIT` atau `ROLLBACK`.
- Cari jalur kode yang tidak memiliki `COMMIT`, `ROLLBACK`, atau `END`.
- Pastikan logika penanganan pengecualian pada aplikasi Anda selalu memiliki jalur ke `end of transaction` yang valid.
- Pastikan bahwa aplikasi Anda memproses hasil kueri setelah mengakhiri transaksi dengan `COMMIT` atau `ROLLBACK`.

Menanggapi transaksi jangka panjang

Jika transaksi jangka panjang menyebabkan `Lock:transactionid` yang sering terjadi, cobalah strategi berikut:

- Jauhkan kunci baris dari transaksi jangka panjang.
- Batasi panjang kueri dengan menerapkan autocommit bila memungkinkan.

## Lock:tuple

Peristiwa `Lock:tuple` terjadi saat proses backend menunggu perolehan kunci pada tuple.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

## Konteks

Peristiwa `Lock:tuple` menunjukkan bahwa backend sedang menunggu untuk memperoleh kunci pada tuple, sementara backend lain memegang kunci yang bertentangan pada tuple yang sama. Tabel berikut menggambarkan skenario saat sesi membuat peristiwa `Lock:tuple`.

| Wakt | Sesi 1 | Sesi 2 | Sesi 3 |
|---|---|---|---|
| t1 | Memulai transaksi. | | |
| t2 | Memperbarui baris 1. | | |
| t3 | | Memperbarui baris 1. Sesi tersebut mendapatk an kunci eksklusif pada tuple, lalu menunggu sesi 1 untuk melepaskan kunci dengan metode commit atau rollback. | |
| t4 | | | Memperbarui baris 1. Sesi tersebut menunggu sesi 2 untuk melepaskan kunci eksklusif pada tuple. |

Atau Anda dapat menyimulasikan peristiwa tunggu ini dengan menggunakan alat benchmarking `pgbench`. Konfigurasikan jumlah sesi bersamaan yang tinggi untuk memperbarui baris yang sama pada tabel dengan file SQL khusus.

Untuk mempelajari selengkapnya tentang mode kunci yang bertentangan, lihat Penguncian Eksplisit pada dokumentasi PostgreSQL. Untuk mempelajari selengkapnya tentang `pgbench`, lihat pgbench pada dokumentasi PostgreSQL.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini ditampilkan lebih dari biasanya, mungkin menunjukkan masalah performa, penyebab umumnya terdiri dari berikut:

- Sesi bersamaan dalam jumlah yang tinggi mencoba mendapatkan kunci yang bertentangan untuk tuple yang sama dengan menjalankan pernyataan `UPDATE` atau `DELETE`.
- Sesi bersamaan yang tinggi menjalankan pernyataan `SELECT` menggunakan mode kunci `FOR UPDATE` atau `FOR NO KEY UPDATE`.
- Berbagai faktor mendorong aplikasi atau pool koneksi untuk membuka lebih banyak sesi agar menjalankan operasi yang sama. Saat sesi baru mencoba memodifikasi baris yang sama, beban DB dapat melonjak, dan `Lock:tuple` dapat ditampilkan.

Untuk informasi selengkapnya, lihat Penguncian Tingkat Baris pada dokumentasi PostgreSQL.

## Tindakan

Kami merekomendasikan berbagai tindakan tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menyelidiki logika aplikasi
- Menemukan sesi pemblokir
- Mengurangi konkurensi saat tinggi
- Memecahkan masalah

### Menyelidiki logika aplikasi

Cari tahu apakah sesi pemblokir telah berada pada status `idle in transaction` dalam waktu yang lama. Jika demikian, coba akhiri sesi pemblokir sebagai solusi jangka pendek. Anda dapat menggunakan fungsi `pg_terminate_backend`. Untuk informasi selengkapnya tentang fungsi ini, lihat Fungsi Pemberi Sinyal Server pada dokumentasi PostgreSQL.

Untuk solusi jangka panjang, lakukan hal berikut:

- Sesuaikan logika aplikasi.
- Gunakan parameter `idle_in_transaction_session_timeout`. Parameter ini mengakhiri sesi apa pun dengan transaksi terbuka yang telah idle lebih lama dari jumlah waktu yang ditentukan. Untuk informasi selengkapnya, lihat Default Koneksi Klien pada dokumentasi PostgreSQL.
- Gunakan autocommit sebanyak mungkin. Untuk informasi selengkapnya, lihat SET AUTOCOMMIT pada dokumentasi PostgreSQL.

Menemukan sesi pemblokir

Saat peristiwa tunggu `Lock:tuple` terjadi, identifikasi pemblokir dan sesi yang diblokir dengan mencari tahu kunci yang bergantung satu sama lain. Untuk informasi selengkapnya, lihat Informasi dependensi kunci pada wiki PostgreSQL. Untuk menganalisis peristiwa `Lock:tuple` yang lampau, gunakan fungsi Aurora `aurora_stat_backend_waits`.

Contoh berikut memberikan kueri semua sesi, memfilter `tuple`, dan mengurutkan berdasarkan `wait_time`.

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     left(query,80) as current_query,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

  pid  | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
```

```
-------+---------+----------+-------------------------------------------------
+------------------+--------------------+--------------+-----------+-----------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

### Mengurangi konkurensi saat tinggi

Peristiwa `Lock:tuple` mungkin terjadi terus-menerus, terutama pada waktu beban kerja yang sibuk. Dalam situasi ini, coba kurangi konkurensi tinggi untuk baris yang sangat sibuk. Sering kali, hanya beberapa baris mengontrol antrean atau logika Boolean, yang membuat baris ini sangat sibuk.

Anda dapat mengurangi konkurensi dengan menggunakan pendekatan yang berbeda berdasarkan ketentuan bisnis, logika aplikasi, dan jenis beban kerja. Misalnya, Anda dapat melakukan hal berikut:

- Mendesain ulang tabel dan logika data untuk mengurangi konkurensi tinggi.
- Mengubah logika aplikasi untuk mengurangi konkurensi tinggi di tingkat baris.
- Memanfaatkan dan mendesain ulang kueri dengan kunci tingkat baris.
- Menggunakan klausa `NOWAIT` dengan operasi coba lagi.
- Mempertimbangkan penggunaan kontrol konkurensi logika penguncian hibrid yang optimis.
- Mempertimbangkan perubahan tingkat isolasi basis data.

### Memecahkan masalah

`Lock:tuple` dapat terjadi dengan masalah seperti kekurangan CPU atau penggunaan maksimum bandwidth Amazon EBS. Untuk mengurangi masalah, pertimbangkan pendekatan berikut:

- Menaikkan skala jenis kelas instans.
- Mengoptimalkan kueri intensif sumber daya.
- Mengubah logika aplikasi.
- Mengarsipkan data yang jarang diakses.

# LWLock:buffer_content (BufferContent)

Peristiwa `LWLock:buffer_content` terjadi saat suatu sesi menunggu untuk membaca atau menulis halaman data di memori, sementara sesi lain mengunci halaman tersebut untuk penulisan. Di Aurora PostgreSQL 13 dan versi yang lebih tinggi, peristiwa tunggu ini disebut `BufferContent`.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

## Konteks

Untuk membaca atau memanipulasi data, PostgreSQL mengaksesnya melalui buffer memori bersama. Untuk membaca dari buffer, suatu proses mendapatkan LWLock (lightweight lock) pada konten buffer dalam mode bersama. Untuk menulis ke buffer, aplikasi ini mendapatkan kunci tersebut dalam mode eksklusif. Kunci bersama memungkinkan proses lain untuk secara bersamaan memperoleh kunci bersama pada konten tersebut. Kunci eksklusif mencegah proses lain menerima jenis kunci apa pun.

Peristiwa `LWLock:buffer_content` (`BufferContent`) menunjukkan bahwa beberapa proses mencoba mendapatkan kunci pada konten buffer tertentu.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `LWLock:buffer_content` (`BufferContent`) muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Peningkatan pembaruan bersamaan ke data yang sama

Mungkin terdapat peningkatan jumlah sesi bersamaan dengan kueri yang memperbarui konten buffer yang sama. Pertentangan ini dapat lebih terasa pada tabel dengan banyak indeks.

Data beban kerja tidak ada dalam memori

Saat data yang diproses oleh beban kerja aktif tidak ada dalam memori, peristiwa tunggu ini dapat meningkat. Efek ini terjadi karena proses yang menahan kunci dapat melakukannya lebih lama saat menjalankan operasi I/O disk.

Penggunaan batasan kunci asing yang berlebihan

Batasan kunci asing dapat meningkatkan jumlah waktu suatu proses menahan kunci konten buffer. Efek ini terjadi karena operasi baca memerlukan kunci konten buffer bersama pada kunci yang direferensikan saat kunci tersebut diperbarui.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda. Anda dapat mengidentifikasi peristiwa `LWLock:buffer_content` (`BufferContent`) dengan menggunakan Wawasan Performa Amazon RDS atau dengan membuat kueri tampilan `pg_stat_activity`.

Topik

- Meningkatkan efisiensi dalam memori
- Mengurangi penggunaan batasan kunci asing
- Menghapus indeks yang tidak digunakan

### Meningkatkan efisiensi dalam memori

Untuk meningkatkan kemungkinan data beban kerja aktif berada dalam memori, buat partisi tabel atau naikkan skala kelas instans Anda. Untuk informasi tentang kelas instans DB, lihat Kelas instans Amazon Aurora DB.

### Mengurangi penggunaan batasan kunci asing

Selidiki beban kerja yang mengalami jumlah peristiwa tunggu `LWLock:buffer_content` (`BufferContent`) yang tinggi untuk penggunaan batasan kunci asing. Hapus batasan kunci asing yang tidak perlu.

### Menghapus indeks yang tidak digunakan

Untuk beban kerja yang mengalami jumlah peristiwa tunggu `LWLock:buffer_content` (`BufferContent`) yang tinggi, identifikasi indeks yang tidak digunakan, lalu hapus.

# LWLock:buffer_mapping

Peristiwa ini terjadi saat sesi menunggu untuk mengaitkan blok data dengan buffer di pool buffer bersama.

> **Note**
>
> Peristiwa ini ditampilkan sebagai `LWLock:buffer_mapping` di Aurora PostgreSQL versi 12 dan yang lebih rendah, serta `LWLock:BufferMapping` di versi 13 dan yang lebih tinggi.

Topik


- Versi mesin yang didukung
- Konteks
- Penyebab
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

## Konteks

Pool buffer bersama adalah area memori Aurora PostgreSQL yang memegang semua halaman yang sedang atau telah digunakan oleh proses. Saat memerlukan halaman, suatu proses membaca halaman ke dalam pool buffer bersama. Parameter `shared_buffers` menetapkan ukuran buffer bersama dan menyimpan area memori untuk menyimpan tabel dan halaman indeks. Jika Anda mengubah parameter ini, pastikan untuk memulai ulang basis data. Untuk informasi selengkapnya, lihat Buffer bersama.

Peristiwa tunggu `LWLock:buffer_mapping` terjadi dalam skenario berikut:

- Suatu proses mencari tabel buffer untuk suatu halaman dan memperoleh kunci pemetaan buffer bersama.
- Suatu proses memuat suatu halaman ke dalam pool buffer dan memperoleh kunci pemetaan buffer eksklusif.
- Suatu proses menghapus suatu halaman dari pool dan memperoleh kunci pemetaan buffer eksklusif.

## Penyebab

Saat peristiwa ini ditampilkan lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, basis data masuk dan keluar dari pool buffer bersama. Penyebab umumnya meliputi yang berikut:

- Kueri yang besar
- Bloat indeks dan tabel
- Pemindaian tabel lengkap
- Ukuran pool bersama yang lebih kecil dari set kerja

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Memantau metrik terkait buffer
- Menilai strategi pengindeksan
- Mengurangi jumlah buffer yang harus dialokasikan dengan cepat

### Memantau metrik terkait buffer

Saat peristiwa tunggu `LWLock:buffer_mapping` melonjak, selidiki rasio hit buffer. Anda dapat menggunakan metrik ini untuk mendapatkan pemahaman yang lebih baik tentang apa yang terjadi pada cache buffer. Periksa metrik berikut:

`BufferCacheHitRatio`

Metrik Amazon CloudWatch ini mengukur persentase permintaan yang dilayani oleh cache buffer instans DB pada klaster DB Anda. Anda mungkin melihat penurunan metrik ini sebelum peristiwa tunggu `LWLock:buffer_mapping`.

`blks_hit`

Metrik penghitung Wawasan Performa ini menunjukkan jumlah blok yang diambil dari pool buffer bersama. Setelah peristiwa tunggu `LWLock:buffer_mapping` ditampilkan, Anda mungkin melihat lonjakan pada `blks_hit`.

`blks_read`

Metrik penghitung Wawasan Performa ini menunjukkan jumlah blok yang memerlukan I/O untuk dibaca ke dalam pool buffer bersama. Anda mungkin melihat lonjakan pada `blks_read` sebelum peristiwa tunggu `LWLock:buffer_mapping`.

Menilai strategi pengindeksan

Untuk mengonfirmasi bahwa strategi pengindeksan tidak menurunkan performa, periksa hal berikut:

Bloat indeks

Pastikan bloat indeks dan tabel tidak menyebabkan halaman yang tidak perlu turut dibaca ke buffer bersama. Jika tabel Anda berisi baris yang tidak digunakan, coba arsipkan data dan hapus baris dari tabel. Anda selanjutnya dapat membangun kembali indeks untuk tabel yang diubah ukurannya.

Indeks untuk kueri yang sering digunakan

Untuk menentukan apakah Anda memiliki indeks optimal, pantau metrik mesin DB di Wawasan Performa. Metrik `tup_returned` menunjukkan jumlah baris yang dibaca. Metrik `tup_fetched` menunjukkan jumlah baris yang dikembalikan ke klien. Jika `tup_returned` secara signifikan lebih besar dari `tup_fetched`, data mungkin tidak diindeks dengan benar. Selain itu, statistik tabel Anda mungkin tidak terkini.

Mengurangi jumlah buffer yang harus dialokasikan dengan cepat

Untuk mengurangi peristiwa tunggu `LWLock:buffer_mapping`, coba kurangi jumlah buffer yang harus dialokasikan dengan cepat. Salah satu strateginya adalah dengan melakukan operasi batch yang lebih kecil. Anda mungkin dapat mencapai batch yang lebih kecil dengan partisi tabel.

## LWLock:BufferIO (IPC: Bufferio)

`LWLock:BufferIO`Peristiwa ini terjadi ketika Aurora PostgreSQL atau RDS untuk PostgreSQL sedang menunggu proses lain untuk menyelesaikan operasi mereka) ketika secara bersamaan mencoba mengakses halaman. input/output (I/O Tujuannya adalah agar halaman yang sama dibacakan ke buffer bersama.

Topik

- Versi mesin yang relevan

- Konteks
- Penyebab
- Tindakan


## Versi mesin yang relevan

Informasi peristiwa tunggu ini relevan untuk semua versi Aurora PostgreSQL. Untuk Aurora PostgreSQL 12 dan versi sebelumnya, peristiwa tunggu ini disebut lwlock:buffer_io, sedangkan pada Aurora PostgreSQL versi 13 disebut lwlock:bufferio. Dari Aurora PostgreSQL versi 14, peristiwa tunggu BufferIO dipindahkan dari jenis peristiwa tunggu `LWLock` ke `IPC` (IPC:BufferIO).

## Konteks

Setiap buffer bersama memiliki kunci I/O yang terkait dengan peristiwa tunggu `LWLock:BufferIO`, setiap kali blok (atau halaman) harus diambil di luar pool buffer bersama.

Kunci ini digunakan untuk menangani beberapa sesi yang semuanya memerlukan akses ke blok yang sama. Blok ini harus dibaca dari luar pool buffer bersama, yang ditentukan oleh parameter `shared_buffers`.

Segera setelah halaman dibaca di dalam kumpulan buffer bersama, kunci `LWLock:BufferIO` akan dilepaskan.

> **Note**
>
> Peristiwa tunggu `LWLock:BufferIO` mendahului peristiwa tunggu IO: DataFileRead. Peristiwa tunggu `IO:DataFileRead` terjadi saat data sedang dibaca dari penyimpanan.

Untuk informasi selengkapnya tentang kunci ringan, lihat Gambaran Umum Penguncian.

## Penyebab

Penyebab umum peristiwa `LWLock:BufferIO` muncul dalam peristiwa tunggu teratas mencakup yang berikut:

- Beberapa backend atau koneksi mencoba mengakses halaman yang sama yang juga menunggu operasi I/O

- Rasio antara ukuran pool buffer bersama (ditentukan oleh parameter `shared_buffers`) dan jumlah buffer yang dibutuhkan oleh beban kerja saat ini
- Ukuran pool buffer bersama tidak seimbang dengan jumlah halaman yang dikosongkan oleh operasi lain
- Indeks besar atau bloat yang mengharuskan mesin membacakan lebih banyak halaman daripada yang diperlukan ke dalam pool buffer bersama
- Kurangnya indeks yang memaksa mesin DB untuk membaca lebih banyak halaman dari tabel daripada yang diperlukan
- Lonjakan mendadak untuk koneksi basis data yang mencoba melakukan operasi pada halaman yang sama

### Tindakan

Kami merekomendasikan berbagai tindakan tergantung pada penyebab peristiwa tunggu Anda:

- Amati CloudWatch metrik Amazon untuk korelasi antara penurunan tajam dalam peristiwa `BufferCacheHitRatio` dan `LWLock:BufferIO` menunggu. Efek ini dapat berarti bahwa Anda memiliki pengaturan buffer bersama kecil. Anda mungkin perlu meningkatkan atau menaikkan skala kelas instans DB Anda. Anda dapat membagi beban kerja Anda menjadi lebih banyak simpul pembaca.
- Verifikasi apakah Anda memiliki indeks yang tidak digunakan, lalu hapus.
- Gunakan tabel yang dipartisi (yang juga memiliki indeks yang dipartisi). Dengan melakukan hal ini, Anda dapat menjaga penyusunan ulang indeks tetap rendah dan mengurangi dampaknya.
- Hindari kolom pengindeksan yang tidak perlu.
- Cegah lonjakan koneksi basis data yang tiba-tiba dengan menggunakan pool koneksi.
- Batasi jumlah maksimum koneksi ke basis data sebagai praktik terbaik.

## LWLock:lock_manager

Peristiwa ini terjadi saat mesin Aurora PostgreSQL mempertahankan area memori kunci bersama untuk mengalokasikan, memeriksa, dan mendealokasikan kunci saat kunci jalur cepat tidak memungkinkan.

Topik

- Versi mesin yang didukung

- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

## Konteks

Saat Anda mengeluarkan pernyataan SQL, Aurora PostgreSQL mencatat kunci untuk melindungi struktur, data, dan integritas basis data Anda selama operasi bersamaan. Mesin dapat mencapai tujuan ini menggunakan kunci jalur cepat atau kunci jalur yang tidak cepat. Kunci jalur yang tidak cepat lebih mahal dan membuat lebih banyak overhead daripada kunci jalur cepat.

Penguncian jalur cepat

Untuk mengurangi overhead kunci yang sering diambil dan dilepaskan, tetapi jarang terjadi konflik, proses backend dapat menggunakan penguncian jalur cepat. Basis data menggunakan mekanisme ini untuk kunci yang memenuhi kriteria berikut:

- Menggunakan metode kunci DEFAULT.
- Mewakili kunci pada relasi basis data, bukan relasi bersama.
- Merupakan kunci lemah yang tidak mungkin bertentangan.
- Mesin dapat dengan cepat memverifikasi bahwa tidak ada kunci yang saling bertentangan.

Mesin tidak dapat menggunakan penguncian jalur cepat jika salah satu kondisi berikut ini benar:

- Kunci tidak memenuhi kriteria sebelumnya.
- Tidak ada lagi slot yang tersedia untuk proses backend.

Untuk informasi selengkapnya tentang penguncian jalur cepat, lihat jalur cepat di README manajer kunci PostgreSQL dan pg-locks pada dokumentasi PostgreSQL.

Contoh masalah penskalaan untuk manajer kunci

Pada contoh ini, tabel `purchases` menyimpan data selama lima tahun, yang dipartisi berdasarkan hari. Setiap partisi memiliki dua indeks. Urutan peristiwa berikut terjadi:

1. Anda membuat kueri data bernilai beberapa hari, yang memerlukan basis data untuk membaca banyak partisi.
2. Basis data membuat entri kunci untuk setiap partisi. Jika indeks partisi adalah bagian dari jalur akses pengoptimal, basis data akan membuat entri kunci untuk mereka juga.
3. Saat jumlah entri kunci yang diminta untuk proses backend yang sama lebih tinggi dari 16, yang merupakan nilai `FP_LOCK_SLOTS_PER_BACKEND`, manajer kunci menggunakan metode kunci jalur yang tidak cepat.

Aplikasi modern mungkin memiliki ratusan sesi. Jika sesi bersamaan membuat kueri induk tanpa pemangkasan partisi yang tepat, basis data mungkin membuat ratusan atau bahkan ribuan kunci jalur yang tidak cepat. Biasanya, saat konkurensi ini lebih tinggi dari jumlah vCPU, peristiwa tunggu `LWLock:lock_manager` akan ditampilkan.

> **Note**
>
> Peristiwa tunggu `LWLock:lock_manager` tidak terkait dengan jumlah partisi atau indeks dalam skema basis data. Sebaliknya, peristiwa tunggu terkait dengan jumlah kunci jalur yang tidak cepat yang harus dikontrol oleh basis data.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa tunggu `LWLock:lock_manager` terjadi lebih dari biasanya, mungkin menunjukkan masalah performa, kemungkinan penyebab lonjakan mendadak adalah berikut:

- Sesi aktif bersamaan menjalankan kueri yang tidak menggunakan kunci jalur cepat. Sesi ini juga melebihi vCPU maksimum.
- Sesi aktif bersamaan dalam jumlah besar mengakses tabel yang banyak dipartisi. Setiap partisi memiliki beberapa indeks.
- Basis data mengalami storm koneksi. Secara default, beberapa aplikasi dan perangkat lunak pool koneksi membuat lebih banyak koneksi saat basis data lambat. Hal ini memperburuk masalah. Sesuaikan perangkat lunak pool koneksi Anda sehingga storm koneksi tidak terjadi.
- Jumlah sesi yang besar membuat kueri tabel induk tanpa memangkas partisi.
- Data definition language (DDL), data manipulation language (DML), atau perintah pemeliharaan secara eksklusif mengunci relasi yang sibuk atau tuple yang sering diakses atau dimodifikasi.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menggunakan pemangkasan partisi
- Menghapus indeks yang tidak diperlukan
- Menyetel kueri untuk penguncian jalur cepat
- Menyetel untuk peristiwa tunggu lainnya
- Mengurangi masalah perangkat keras
- Menggunakan pooler koneksi
- Meningkatkan versi Aurora PostgreSQL Anda

### Menggunakan pemangkasan partisi

Pemangkasan partisi adalah strategi pengoptimalan kueri yang mengecualikan partisi yang tidak dibutuhkan dari pemindaian tabel, sehingga meningkatkan performa. Pemangkasan partisi diaktifkan secara default. Jika tidak aktif, aktifkan dengan cara berikut:

```
SET enable_partition_pruning = on;
```

Kueri dapat memanfaatkan pemangkasan partisi saat klausa `WHERE` mereka berisi kolom yang digunakan untuk partisi. Untuk informasi selengkapnya, lihat Pemangkasan Partisi pada dokumentasi PostgreSQL.

### Menghapus indeks yang tidak diperlukan

Basis data Anda mungkin berisi indeks yang tidak digunakan atau jarang digunakan. Jika ya, pertimbangkanlah untuk menghapusnya. Lakukan salah satu dari langkah berikut:

- Mempelajari cara menemukan indeks yang tidak perlu dengan membaca Indeks yang Tidak Digunakan di wiki PostgreSQL.
- Menjalankan PG Collector. Skrip SQL ini mengumpulkan informasi basis data dan menyajikannya dalam laporan HTML terkonsolidasi. Memeriksa bagian “Indeks yang tidak digunakan”. Untuk informasi selengkapnya, lihat pg-collector dalam repositori GitHub Lab AWS.

Menyetel kueri untuk penguncian jalur cepat

Untuk mengetahui apakah kueri Anda menggunakan penguncian jalur cepat, buat kueri pada kolom `fastpath` dalam tabel `pg_locks`. Jika kueri Anda tidak menggunakan penguncian jalur cepat, cobalah kurangi jumlah relasi per kueri menjadi kurang dari 16.

Menyetel untuk peristiwa tunggu lainnya

Jika `LWLock:lock_manager` adalah yang pertama atau kedua dalam daftar peristiwa tunggu teratas, periksa apakah peristiwa tunggu berikut juga ditampilkan dalam daftar:

- `Lock:Relation`
- `Lock:transactionid`
- `Lock:tuple`

Jika peristiwa sebelumnya ditampilkan pada posisi tinggi dalam daftar, coba setel peristiwa tunggu ini terlebih dahulu. Peristiwa ini dapat menjadi pendorong untuk `LWLock:lock_manager`.

Mengurangi masalah perangkat keras

Anda mungkin memiliki masalah perangkat keras, seperti kekurangan CPU atau penggunaan maksimum bandwidth Amazon EBS. Jika demikian, coba kurangi masalah perangkat keras. Pertimbangkan tindakan berikut:

- Menaikkan skala kelas instans.
- Mengoptimalkan kueri yang mengonsumsi CPU dan memori dalam jumlah besar.
- Mengubah logika aplikasi.
- Mengarsipkan data.

Untuk informasi selengkapnya tentang CPU, memori, dan bandwidth jaringan EBS, lihat Jenis Instans Amazon RDS.

Menggunakan pooler koneksi

Jika jumlah total koneksi aktif Anda melebihi vCPU maksimum, lebih banyak proses OS memerlukan CPU daripada yang dapat didukung oleh jenis instans Anda. Jika demikian, pertimbangkanlah untuk menggunakan atau menyetel pool koneksi. Untuk informasi selengkapnya tentang vCPU untuk jenis instans Anda, lihat Jenis Instans Amazon RDS.

Untuk informasi selengkapnya tentang pengumpulan koneksi, lihat sumber daya berikut:

- Proksi Amazon RDS untuk Aurora
- pgbouncer
- Pool Koneksi dan Sumber Data pada Dokumentasi PostgreSQL

Meningkatkan versi Aurora PostgreSQL Anda

Jika versi Aurora PostgreSQL Anda saat ini lebih rendah dari 12, tingkatkan ke versi 12 atau yang lebih tinggi. PostgreSQL versi 12 dan 13 memiliki mekanisme partisi yang ditingkatkan. Untuk informasi selengkapnya tentang versi 12, lihat Catatan Rilis PostgreSQL 12.0. Untuk informasi selengkapnya tentang meningkatkan Aurora PostgreSQL, lihat Pembaruan mesin database untuk Amazon Aurora Postgre SQL.

## LWLock:MultiXact

Peristiwa tunggu `LWLock:MultiXactMemberBuffer`, `LWLock:MultiXactOffsetBuffer`, `LWLock:MultiXactMemberSLRU`, dan `LWLock:MultiXactOffsetSLRU` menunjukkan bahwa sesi sedang menunggu pengambilan daftar transaksi yang memodifikasi baris yang sama pada tabel tertentu.

- `LWLock:MultiXactMemberBuffer` – Suatu proses sedang menunggu I/O pada buffer simple least-recently used (SLRU) untuk anggota multixact.
- `LWLock:MultiXactMemberSLRU` – Suatu proses sedang menunggu akses cache simple least-recently used (SLRU) untuk anggota multixact.
- `LWLock:MultiXactOffsetBuffer` – Suatu proses sedang menunggu I/O pada buffer simple least-recently used (SLRU) untuk offset multixact.
- `LWLock:MultiXactOffsetSLRU` – Suatu proses sedang menunggu akses cache simple least-recently used (SLRU) untuk offset multixact.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

## Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

## Konteks

Multixact adalah struktur data yang menyimpan daftar transaksi IDs (XIDs) yang memodifikasi baris tabel yang sama. Saat satu transaksi merujuk suatu baris dalam tabel, ID transaksi disimpan pada baris header tabel. Ketika beberapa transaksi mereferensikan baris yang sama dalam tabel, daftar transaksi IDs disimpan dalam struktur data multixact. Peristiwa tunggu multixact menunjukkan bahwa sesi mengambil dari struktur data daftar transaksi yang merujuk ke baris tertentu dalam tabel.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Tiga penyebab umum penggunaan multixact adalah sebagai berikut:

- Subtransaksi dari titik simpan eksplisit – Secara eksplisit membuat titik simpan pada transaksi Anda yang memunculkan transaksi baru untuk baris yang sama. Misalnya, menggunakan `SELECT FOR UPDATE`, lalu `SAVEPOINT`, dan `UPDATE`.

  Beberapa driver, pemetaan relasional objek (ORMs), dan lapisan abstraksi memiliki opsi konfigurasi untuk secara otomatis membungkus semua operasi dengan savepoint. Hal ini dapat membuat banyak peristiwa tunggu multixact di beberapa beban kerja. Opsi `autosave` Driver JDBC PostgreSQL adalah contohnya. Untuk informasi selengkapnya, lihat pgJDBC pada dokumentasi JDBC PostgreSQL. Contoh lainnya adalah driver ODBC PostgreSQL dan opsi `protocol`. Untuk informasi selengkapnya, lihat Opsi Konfigurasi psqlODBC pada dokumentasi driver ODBC PostgreSQL.
- Subtransaksi dari klausa PL/pgSQL EXCEPTION – Setiap klausa `EXCEPTION` yang Anda tulis dalam fungsi atau prosedur PL/pgSQL membuat `SAVEPOINT` secara internal.
- Kunci asing – Beberapa transaksi memperoleh kunci bersama pada catatan induk.

Ketika baris tertentu disertakan dalam operasi transaksi ganda, memproses baris memerlukan pengambilan transaksi IDs dari `multixact` daftar. Jika pencarian tidak mendapatkan multixact dari cache memori, struktur data harus dibaca dari lapisan penyimpanan Aurora. I/O dari penyimpanan ini berarti kueri SQL dapat memerlukan waktu lebih lama. Kesalahan cache memori dapat mulai terjadi dengan penggunaan yang berat karena jumlah transaksi ganda yang besar. Semua faktor ini berkontribusi pada peningkatan peristiwa tunggu ini.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda. Beberapa tindakan ini dapat membantu untuk segera mengurangi peristiwa tunggu. Namun, yang lainnya mungkin memerlukan penyelidikan dan koreksi untuk menskalakan beban kerja Anda.

Topik

- Lakukan pembekuan vakum di atas tabel dengan peristiwa tunggu ini
- Tingkatkan frekuensi autovacuum pada tabel dengan peristiwa tunggu ini
- Meningkatkan parameter memori
- Mengurangi transaksi jangka panjang
- Tindakan jangka panjang

### Lakukan pembekuan vakum di atas tabel dengan peristiwa tunggu ini

Jika peristiwa tunggu ini melonjak mendadak dan memengaruhi lingkungan produksi, Anda dapat menggunakan salah satu metode sementara berikut untuk mengurangi penghitungannya.

- Gunakan VACUUM FREEZE pada tabel atau partisi tabel yang terpengaruh untuk segera menyelesaikan masalah. Untuk informasi selengkapnya, lihat VACUUM.
- Gunakan klausa VACUUM (FREEZE, INDEX_CLEANUP FALSE) untuk melakukan vakum cepat dengan melewati indeks. Untuk informasi selengkapnya, lihat Memvakum tabel secepat mungkin.

### Tingkatkan frekuensi autovacuum pada tabel dengan peristiwa tunggu ini

Setelah memindai semua tabel di semua basis data, VACUUM akan menghapus multixact, dan nilai multixact tertua akan dimajukan. Untuk informasi selengkapnya, lihat Multixact dan Wraparound. Untuk menjaga LWLock: MultiXact tunggu acara seminimal mungkin, Anda harus menjalankan VACUUM sesering yang diperlukan. Untuk melakukannya, pastikan bahwa VACUUM di klaster DB Aurora PostgreSQL Anda dikonfigurasi secara optimal.

Jika menggunakan VACUUM FREEZE pada tabel atau partisi tabel yang terpengaruh menyelesaikan masalah peristiwa tunggu, sebaiknya gunakan penjadwal, seperti `pg_cron`, untuk melakukan VACUUM daripada menyesuaikan autovacuum pada tingkat instans.

Agar autovacuum terjadi lebih sering, Anda dapat mengurangi nilai parameter penyimpanan `autovacuum_multixact_freeze_max_age` pada tabel yang terpengaruh. Untuk informasi selengkapnya, lihat autovacuum_multixact_freeze_max_age.

Meningkatkan parameter memori

Anda dapat mengoptimalkan penggunaan memori untuk cache multixact dengan menyesuaikan parameter berikut. Pengaturan ini mengontrol berapa banyak memori yang disediakan untuk cache ini, yang dapat membantu mengurangi peristiwa menunggu multixact di beban kerja Anda. Kami merekomendasikan memulai dengan nilai-nilai berikut:

Untuk Aurora PostgreSQL 17 dan yang lebih baru:

- `multixact_offset_buffers`= 128
- `multixact_member_buffers`= 256

Untuk Aurora PostgreSQL 16 dan sebelumnya:

- `multixact_offsets_cache_size`= 128
- `multixact_members_cache_size`= 256

> **Note**
>
> Di Aurora PostgreSQL 17, nama parameter diubah dari `multixact_offsets_cache_size` ke `multixact_offset_buffers` dan dari menjadi selaras dengan komunitas PostgreSQL 17. `multixact_members_cache_size` `multixact_member_buffers`

Anda dapat mengatur parameter ini di tingkat cluster sehingga semua instance di cluster Anda tetap konsisten. Kami menyarankan Anda untuk menguji dan menyesuaikan nilai agar sesuai dengan persyaratan beban kerja spesifik dan kelas instance Anda. Anda harus me-reboot instance writer agar perubahan parameter diterapkan.

Parameter dinyatakan dalam hal entri cache multixact. Setiap entri cache menggunakan `8 KB` memori. Untuk menghitung total memori yang dicadangkan, kalikan setiap nilai parameter dengan`8 KB`. Misalnya, jika Anda menyetel parameter ke 128, total memori cadangan akan menjadi`128 * 8 KB = 1 MB`.

Mengurangi transaksi jangka panjang

Transaksi jangka panjang menyebabkan vakum mempertahankan informasinya hingga transaksi tersebut dilakukan atau hingga transaksi hanya dibaca ditutup. Kami merekomendasikan Anda untuk secara proaktif memantau dan mengelola transaksi jangka panjang. Untuk informasi selengkapnya, lihat Basis data telah lama berjalan idle dalam koneksi transaksi. Cobalah untuk memodifikasi aplikasi Anda untuk menghindari atau meminimalkan penggunaan transaksi jangka panjang Anda.

Tindakan jangka panjang

Periksa beban kerja Anda untuk menemukan penyebab limpahan multixact. Anda harus memperbaiki masalah tersebut untuk menskalakan beban kerja Anda dan mengurangi peristiwa tunggu.

- Anda harus menganalisis DDL (data definition language) yang digunakan untuk membuat tabel Anda. Pastikan bahwa struktur dan indeks tabel dirancang dengan baik.
- Saat tabel yang terpengaruh memiliki kunci asing, tentukan apakah diperlukan atau jika ada cara lain untuk memberlakukan integritas referensial.
- Saat sebuah tabel memiliki indeks besar yang tidak digunakan, hal tersebut dapat menyebabkan autovacuum tidak sesuai dengan beban kerja Anda dan mungkin memblokirnya agar tidak berjalan. Untuk menghindari hal ini, periksa indeks yang tidak digunakan dan hapus sepenuhnya. Untuk informasi selengkapnya, lihat Mengelola autovacuum dengan indeks besar.
- Kurangi penggunaan titik simpan dalam transaksi Anda.

## LWLock:pg_stat_statement

Peristiwa LWLock tunggu: PG_STAT_Statements terjadi ketika ekstensi `pg_stat_statements` mengambil kunci eksklusif pada tabel hash yang melacak pernyataan SQL. Ini terjadi dalam skenario berikut:

- Ketika jumlah pernyataan yang dilacak mencapai nilai `pg_stat_statements.max` parameter yang dikonfigurasi dan ada kebutuhan untuk memberi ruang untuk lebih banyak entri, ekstensi melakukan pengurutan pada jumlah panggilan, menghapus 5% dari pernyataan yang paling tidak dieksekusi, dan mengisi ulang hash dengan entri yang tersisa.
- Ketika `pg_stat_statements` melakukan `garbage collection` operasi ke `pgss_query_texts.stat` file pada disk dan menulis ulang file.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan


## Versi mesin yang didukung

Informasi acara tunggu ini didukung untuk semua versi .

## Konteks

Memahami ekstensi pg_stat_statement - Ekstensi pg_stat_statement melacak statistik eksekusi pernyataan SQL dalam tabel hash. Ekstensi melacak pernyataan SQL hingga batas yang ditentukan oleh `pg_stat_statements.max` parameter. Parameter ini menentukan jumlah maksimum pernyataan yang dapat dilacak yang sesuai dengan jumlah maksimum baris dalam tampilan pg_stat_statement.

Persistensi statistik pernyataan — Ekstensi mempertahankan statistik pernyataan di seluruh instans dimulai ulang dengan:

- Menulis data ke file bernama pg_stat_statements.stat
- Menggunakan parameter pg_stat_statements.save untuk mengontrol perilaku persistensi

Ketika pg_stat_statements.save diatur ke:

- aktif (default): Statistik disimpan saat shutdown dan dimuat ulang saat server dimulai
- off: Statistik tidak disimpan saat shutdown atau dimuat ulang saat server dimulai

Penyimpanan teks kueri — Ekstensi menyimpan teks kueri yang dilacak dalam file bernama. `pgss_query_texts.stat` File ini dapat tumbuh menjadi dua kali lipat ukuran rata-rata semua pernyataan SQL yang dilacak sebelum pengumpulan sampah terjadi. Ekstensi memerlukan kunci eksklusif pada tabel hash selama operasi pembersihan dan menulis ulang `pgss_query_texts.stat` file.

Proses deallocation pernyataan — Ketika jumlah pernyataan yang dilacak mencapai `pg_stat_statements.max` batas dan pernyataan baru perlu dilacak, ekstensi:

- Mengambil kunci eksklusif: pg_stat_statement) LWLock pada tabel hash.
- Memuat data yang ada ke dalam memori lokal.
- Melakukan quicksort berdasarkan jumlah panggilan.
- Menghapus pernyataan yang paling tidak disebut (5% bawah).
- Mengisi ulang tabel hash dengan entri yang tersisa.

Monitoring statement deallocation — Di PostgreSQL 14 dan yang lebih baru, Anda dapat memantau deallocation pernyataan menggunakan tampilan pg_stat_statements_info. Tampilan ini mencakup kolom dealloc yang menunjukkan berapa kali pernyataan dialokasikan untuk memberi ruang bagi yang baru

Jika deallocation pernyataan sering terjadi, itu akan menyebabkan pengumpulan sampah `pgss_query_texts.stat` file yang lebih sering pada disk.

## Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab khas peningkatan `LWLock:pg_stat_statements` menunggu meliputi:

- Peningkatan jumlah kueri unik yang digunakan oleh aplikasi.
- Nilai `pg_stat_statements.max` parameter menjadi kecil dibandingkan dengan jumlah query unik yang digunakan.

## Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda. Anda dapat mengidentifikasi `LWLock:pg_stat_statements` peristiwa dengan menggunakan Amazon RDS Performance Insights atau dengan menanyakan tampilan. `pg_stat_activity`

Sesuaikan `pg_stat_statements` parameter berikut untuk mengontrol perilaku pelacakan dan kurangi: LWLock pg_stat_ statement wait events.

Topik

- Nonaktifkan parameter pg_stat_statements.track
- Meningkatkan parameter pg_stat_statements.max
- Nonaktifkan parameter pg_stat_statements.track_utility

Nonaktifkan parameter pg_stat_statements.track

Jika peristiwa:pg_stat_statement LWLock wait berdampak buruk pada kinerja database, dan solusi cepat diperlukan sebelum analisis lebih lanjut dari tampilan `pg_stat_statements` untuk mengidentifikasi akar penyebab, parameter dapat dinonaktifkan dengan menyetelnya ke. `pg_stat_statements.track` `none` Ini akan menonaktifkan pengumpulan statistik pernyataan.

Meningkatkan parameter pg_stat_statements.max

Untuk mengurangi deallocation dan meminimalkan pengumpulan sampah `pgss_query_texts.stat` file pada disk, tingkatkan nilai `pg_stat_statements.max` parameter. Nilai default-nya adalah `5,000`.

> **Note**
>
> `pg_stat_statements.max`Parameternya statis. Anda harus memulai ulang instans DB Anda untuk menerapkan perubahan apa pun pada parameter ini.

Nonaktifkan parameter pg_stat_statements.track_utility

Anda dapat menganalisis tampilan pg_stat_statement untuk menentukan perintah utilitas mana yang paling banyak menggunakan sumber daya yang dilacak. `pg_stat_statements`

`pg_stat_statements.track_utility`Parameter mengontrol apakah modul melacak perintah utilitas, yang mencakup semua perintah kecuali SELECT, INSERT, UPDATE, DELETE, dan MERGE. Secara default, parameter ini diatur keon.

Misalnya, ketika aplikasi Anda menggunakan banyak kueri savepoint, yang secara inheren unik, itu dapat meningkatkan deallocation pernyataan. Untuk mengatasinya, Anda dapat menonaktifkan `pg_stat_statements.track_utility` parameter untuk berhenti melacak `pg_stat_statements` kueri savepoint.

> **Note**
>
> `pg_stat_statements.track_utility`Parameternya adalah parameter dinamis. Anda dapat mengubah nilainya tanpa memulai ulang instance database Anda.

Example Contoh kueri titik simpan unik di pg_stat_statement

```
                    query                         |       queryid
--------------------------------------------------+---------------------
 SAVEPOINT JDBC_SAVEPOINT_495701                  | -7249565344517699703
 SAVEPOINT JDBC_SAVEPOINT_1320                    | -1572997038849006629
 SAVEPOINT JDBC_SAVEPOINT_26739                   |  54791337410474486
 SAVEPOINT JDBC_SAVEPOINT_1294466                 |  8170064357463507593
 ROLLBACK TO SAVEPOINT JDBC_SAVEPOINT_65016       | -33608214779996400
 SAVEPOINT JDBC_SAVEPOINT_14185                   | -2175035613806809562
 SAVEPOINT JDBC_SAVEPOINT_45837                   | -6201592986750645383
 SAVEPOINT JDBC_SAVEPOINT_1324                    |  6388797791882029332
```

PostgreSQL 17 memperkenalkan beberapa penyempurnaan untuk pelacakan perintah utilitas:

- Nama Savepoint sekarang ditampilkan sebagai konstanta.
- Global Transaction IDs (GIDs) dari perintah komit dua fase sekarang ditampilkan sebagai konstanta.
- Nama pernyataan DEALLOCATE ditampilkan sebagai konstanta.
- Parameter CALL sekarang ditampilkan sebagai konstanta.

## Timeout:PgSleep

Peristiwa `Timeout:PgSleep` terjadi saat proses server telah memanggil fungsi `pg_sleep` dan menunggu batas waktu tidur berakhir.

Topik

- Versi mesin yang didukung
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

### Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

### Kemungkinan penyebab peningkatan peristiwa tunggu

Peristiwa tunggu ini terjadi saat aplikasi, fungsi yang tersimpan, atau pengguna mengeluarkan pernyataan SQL yang memanggil salah satu fungsi berikut:

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

Fungsi sebelumnya menunda eksekusi hingga waktu yang ditentukan dalam detik telah berlalu. Misalnya, `SELECT pg_sleep(1)` dijeda selama 1 detik. Untuk informasi selengkapnya, lihat Menunda Eksekusi pada dokumentasi PostgreSQL.

### Tindakan

Identifikasi pernyataan yang menjalankan fungsi `pg_sleep`. Tentukan apakah penggunaan fungsi tersebut sesuai.

## Menyetel dengan wawasan proaktif Amazon Guru DevOps

DevOpsWawasan proaktif Guru mendeteksi kondisi pada Aurora SQL Postgre yang dapat menyebabkan masalah, dan memberi tahu Anda tentang mereka sebelum terjadi. Wawasan proaktif dapat mengingatkan Anda tentang idle yang berjalan lama dalam koneksi transaksi. Untuk informasi selengkapnya tentang pemecahan masalah idle yang berjalan lama dalam koneksi transaksi, lihat Basis data telah lama berjalan idle dalam koneksi transaksi

DevOpsGuru dapat melakukan hal berikut:

- Mencegah banyak masalah umum pada basis data dengan memeriksa silang konfigurasi basis data terhadap pengaturan umum yang direkomendasikan.
- Memberi tahu Anda tentang masalah kritis dalam armada yang, jika dibiarkan tanpa diperiksa, dapat menyebabkan masalah yang lebih besar di kemudian hari.
- Memberi tahu Anda tentang masalah yang baru ditemukan.

Setiap wawasan proaktif berisi analisis penyebab masalah dan rekomendasi untuk tindakan korektif.

Untuk informasi selengkapnya tentang Amazon DevOps Guru untuk AmazonRDS, lihatMenganalisis anomali kinerja Aurora dengan Amazon Guru untuk Amazon DevOps RDS.

# Basis data telah lama berjalan idle dalam koneksi transaksi

Koneksi ke basis berada dalam status `idle in transaction` selama lebih dari 1.800 detik.

Topik


- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab masalah ini
- Tindakan
- Metrik terkait


## Versi mesin yang didukung

Informasi wawasan ini didukung untuk semua versi Postgre . RDS SQL

## Konteks

Transaksi dalam status `idle in transaction` dapat menahan kunci yang memblokir kueri lain. Ini juga dapat mencegah `VACUUM` (termasuk autovacuum) membersihkan baris mati, yang menyebabkan penggelembungan indeks atau tabel atau ID transaksi tumpang tindih.

## Kemungkinan penyebab masalah ini

Transaksi dimulai dalam sesi interaktif dengan BEGIN atau START TRANSACTION belum berakhir dengan menggunakanCOMMIT,ROLLBACK, atau END perintah. Hal ini menyebabkan status transaksi berubah ke `idle in transaction`.

## Tindakan

Anda dapat menemukan transaksi idle dengan mengueri `pg_stat_activity`.

Di SQL klien Anda, jalankan kueri berikut untuk mencantumkan semua koneksi dalam `idle in transaction` status dan mengurutkannya berdasarkan durasi:

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
```

```
AND   xact_start is not null
ORDER BY 1 DESC;
```

Kami merekomendasikan tindakan yang berbeda bergantung pada penyebab wawasan Anda.

Topik

- Transaksi akhir
- Mengakhiri koneksi
- Konfigurasikan parameter idle_in_transaction_session_timeout
- Periksa AUTOCOMMIT statusnya
- Memeriksa logika transaksi dalam kode aplikasi Anda

Transaksi akhir

Saat Anda memulai transaksi dalam sesi interaktif dengan BEGIN atau STARTTRANSACTION, transaksi tersebut bergerak ke `idle in transaction` status. Itu tetap dalam keadaan ini sampai Anda mengakhiri transaksi dengan mengeluarkanCOMMIT,ROLLBACK, END perintah atau memutuskan koneksi sepenuhnya untuk memutar kembali transaksi.

Mengakhiri koneksi

Akhiri koneksi dengan transaksi idle menggunakan kueri berikut:

```
SELECT pg_terminate_backend(pid);
```

pid adalah ID proses koneksi.

Konfigurasikan parameter idle_in_transaction_session_timeout

Konfigurasikan parameter `idle_in_transaction_session_timeout` di grup parameter baru. Dengan mengonfigurasi parameter ini, intervensi manual tidak diperlukan untuk mengakhiri status idle yang lama dalam transaksi. Untuk informasi lebih lanjut tentang parameter ini, lihat dokumentasi Postgre SQL.

Pesan berikut akan dilaporkan dalam file SQL log Postgre setelah koneksi dihentikan, ketika transaksi dalam keadaan idle_in_transaction lebih lama dari waktu yang ditentukan.

```
FATAL: terminating connection due to idle in transaction timeout
```

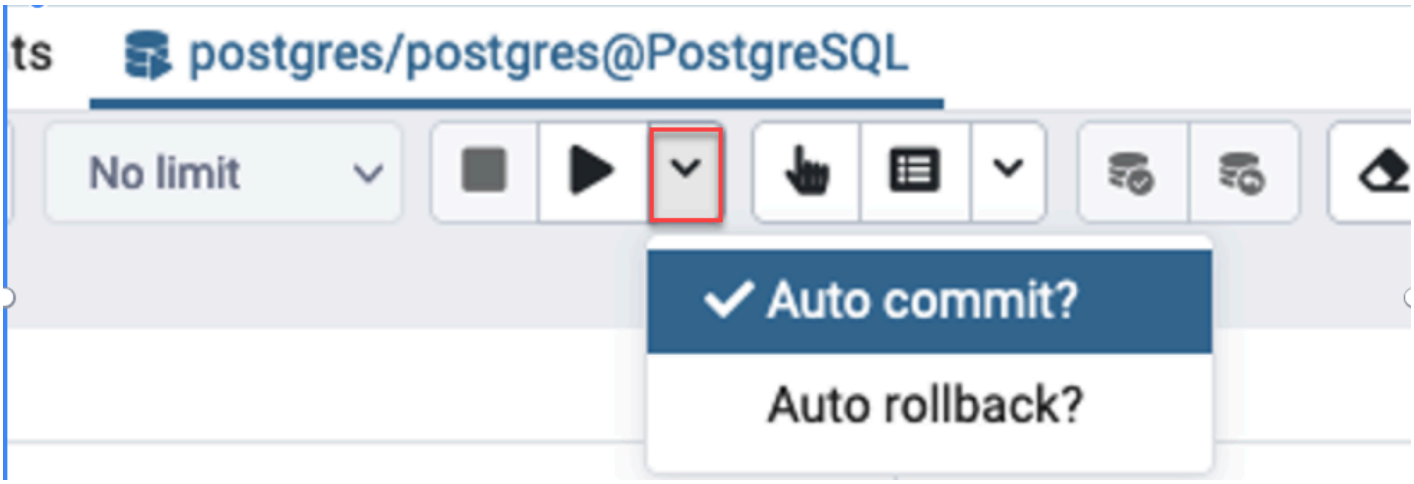
Periksa AUTOCOMMIT statusnya

AUTOCOMMITdihidupkan secara default. Namun, jika dinonaktifkan secara tidak sengaja di klien, pastikan Anda mengaktifkannya kembali.

- Di klien psql Anda, jalankan perintah berikut:

```
postgres=> \set AUTOCOMMIT on
```

- Di pgadmin, nyalakan dengan memilih AUTOCOMMIT opsi dari panah bawah.




Memeriksa logika transaksi dalam kode aplikasi Anda

Selidiki logika aplikasi Anda untuk kemungkinan masalah. Pertimbangkan tindakan berikut:

- Periksa apakah komit JDBC otomatis disetel true dalam aplikasi Anda. Pertimbangkan juga untuk menggunakan perintah `COMMIT` eksplisit dalam kode Anda.
- Periksa logika penanganan kesalahan untuk mengetahui apakah transaksi ditutup setelah kesalahan terjadi.
- Periksa apakah aplikasi Anda membutuhkan waktu lama untuk memproses baris yang ditampilkan oleh kueri saat transaksi terbuka. Jika demikian, pertimbangkan untuk mengodekan aplikasi guna menutup transaksi sebelum memproses baris.
- Periksa apakah transaksi berisi banyak operasi yang berjalan lama. Jika demikian, bagi satu transaksi menjadi beberapa transaksi.

### Metrik terkait

Metrik PI berikut terkait dengan wawasan ini:

- idle_in_transaction_count - Jumlah sesi dalam status `idle in transaction`.
- idle_in_transaction_max_time - Durasi transaksi yang berjalan paling lama dalam status `idle in transaction`.

## Praktik terbaik dengan Amazon Aurora PostgreSQL

Di bagian berikut ini, Anda dapat menemukan beberapa praktik terbaik untuk mengelola klaster DB Amazon Aurora PostgreSQL. Pastikan juga untuk meninjau tugas pemeliharaan dasar. Untuk informasi selengkapnya, lihat Kinerja dan penskalaan untuk Amazon Aurora PostgreSQL.

Topik

- Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans DB Aurora PostgreSQL
- Mendiagnosis bloat tabel dan indeks
- Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL
- Failover cepat dengan Amazon Aurora PostgreSQL
- Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL
- Mengelola churn koneksi Aurora PostgreSQL dengan pooling
- Penanganan koneksi mati di PostgreSQL
- 
- Menggunakan CloudWatch metrik Amazon untuk menganalisis penggunaan sumber daya untuk Aurora PostgreSQL
- Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL
- Memecahkan masalah penyimpanan di Aurora PostgreSQL

## Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans DB Aurora PostgreSQL

Jika Anda menjalankan beban kerja berat atau beban kerja yang melonjak melampaui sumber daya yang dialokasikan untuk instans DB Anda, Anda dapat menghabiskan sumber daya yang Anda

gunakan untuk menjalankan aplikasi dan basis data Aurora. Untuk mendapatkan metrik pada instans database Anda seperti pemanfaatan CPU, penggunaan memori, dan jumlah koneksi database yang digunakan, Anda dapat merujuk ke metrik yang disediakan oleh Amazon, Performance CloudWatch Insights, dan Enhanced Monitoring. Untuk informasi selengkapnya tentang pemantauan instans DB, lihat Memantau metrik di klaster Amazon Aurora.

Jika beban kerja Anda menghabiskan sumber daya yang Anda gunakan, instans DB Anda mungkin akan menjadi lambat, diaktifkan ulang, atau bahkan melakukan failover ke instans DB lain. Untuk menghindari hal ini, pantau pemanfaatan sumber daya Anda, periksa beban kerja yang berjalan pada instans DB Anda, dan buat pengoptimalan jika diperlukan. Jika pengoptimalan tidak meningkatkan metrik instans dan mengurangi kehabisan sumber daya, pertimbangkan untuk menaikkan skala instans DB Anda sebelum mencapai batasnya. Untuk informasi selengkapnya tentang kelas instans DB yang tersedia dan spesifikasinya, lihat Kelas instans Amazon Aurora DB.

## Mendiagnosis bloat tabel dan indeks

Anda dapat menggunakan Multiversion Concurrency Control (MVCC) PostgreSQL untuk membantu menjaga integritas data. MVCC PostgreSQL beroperasi dengan menyimpan salinan internal baris yang diperbarui atau dihapus (juga disebut tuple) sampai transaksi dikomit atau di-rollback. Salinan internal yang disimpan ini tidak terlihat oleh pengguna. Namun, bloat tabel dapat terjadi ketika salinan yang tidak terlihat tersebut tidak dibersihkan secara teratur oleh utilitas VACUUM atau AUTOVACUUM. Bloat tabel yang tidak terpantau dapat menimbulkan peningkatan biaya penyimpanan dan memperlambat kecepatan pemrosesan Anda.

Dalam banyak kasus, pengaturan default untuk VACUUM atau AUTOVACUUM di Aurora cukup untuk menangani bloat tabel yang tidak diinginkan. Namun, Anda sebaiknya memeriksa bloat jika aplikasi Anda mengalami kondisi berikut:

- Memproses sejumlah besar transaksi dalam waktu yang relatif singkat di antara proses VACUUM.
- Beperforma buruk dan kehabisan penyimpanan.

Untuk memulai, kumpulkan informasi paling akurat tentang jumlah ruang yang digunakan oleh tuple nonaktif dan jumlah yang dapat Anda pulihkan dengan membersihkan bloat tabel dan indeks. Untuk melakukannya, gunakan ekstensi `pgstattuple` untuk mengumpulkan statistik tentang klaster Aurora Anda. Untuk informasi selengkapnya, lihat pgstattuple. Hak akses untuk menggunakan ekstensi `pgstattuple` dibatasi pada peran `pg_stat_scan_tables` dan superuser basis data.

Untuk membuat ekstensi `pgstattuple` di Aurora, hubungkan sesi klien ke klaster, misalnya, psql atau pgAdmin, lalu gunakan perintah berikut:

```
CREATE EXTENSION pgstattuple;
```

Buat ekstensi di setiap basis data yang ingin Anda buat profilnya. Setelah membuat ekstensi, gunakan antarmuka baris perintah (CLI) untuk mengukur jumlah ruang yang tidak dapat digunakan yang dapat Anda klaim kembali. Sebelum mengumpulkan statistik, ubah grup parameter klaster dengan menetapkan AUTOVACUUM ke 0. Pengaturan 0 akan mencegah Aurora membersihkan tuple nonaktif secara otomatis yang ditinggalkan oleh aplikasi Anda, yang dapat memengaruhi keakuratan hasil. Masukkan perintah berikut untuk membuat tabel sederhana:

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

Dalam contoh berikut, kita menjalankan kueri dengan AUTOVACUUM diaktifkan untuk klaster DB. `dead_tuple_count` adalah 0, yang menunjukkan bahwa AUTOVACUUM telah menghapus data atau tuple usang dari basis data PostgreSQL.

Untuk menggunakan `pgstattuple` dalam mengumpulkan informasi tentang tabel, tentukan nama tabel atau pengidentifikasi objek (OID) dalam kueri:

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

Dalam kueri berikut, kita menonaktifkan AUTOVACUUM dan memasukkan perintah yang menghapus 25.000 baris dari tabel. Akibatnya, `dead_tuple_count` meningkat menjadi 25000.

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;

DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

Untuk mengklaim kembali tuple nonaktif, mulailah proses VACUUM.

## Mengamati bloat tanpa menghentikan aplikasi Anda

Pengaturan pada klaster Aurora dioptimalkan untuk memberikan praktik terbaik untuk sebagian besar beban kerja. Namun, Anda sebaiknya mengoptimalkan klaster agar lebih sesuai dengan aplikasi dan pola penggunaan Anda. Dalam hal ini, Anda dapat menggunakan ekstensi `pgstattuple` tanpa menghentikan aplikasi yang sibuk. Untuk melakukannya, lakukan langkah-langkah berikut ini:

1. Kloning instans Aurora Anda.
2. Ubah file parameter untuk menonaktifkan AUTOVACUUM di klon.
3. Lakukan kueri `pgstattuple` saat menguji klon dengan beban kerja sampel atau dengan pgbench, yang merupakan program untuk menjalankan pengujian tolok ukur di PostgreSQL. Untuk informasi selengkapnya, lihat pgbench.

Setelah menjalankan aplikasi Anda dan melihat hasilnya, gunakan pg_repack atau VACUUM FULL pada salinan yang dipulihkan dan bandingkan perbedaannya. Jika Anda melihat penurunan yang signifikan pada dead_tuple_count, dead_tuple_len, atau dead_tuple_percent, sesuaikan jadwal vacuum pada klaster produksi Anda untuk meminimalkan bloat.

## Menghindari bloat di tabel sementara

Jika aplikasi Anda membuat tabel sementara, pastikan bahwa aplikasi Anda menghapus tabel sementara ketika sudah tidak lagi diperlukan. Proses autovacuum tidak menemukan tabel sementara. Jika tidak terpantau, tabel sementara dapat dengan cepat membuat bloat basis data. Selain itu, bloat dapat meluas ke tabel sistem, yang merupakan tabel internal yang melacak objek dan atribut PostgreSQL, seperti pg_attribute dan pg_depend.

Ketika tabel sementara tidak lagi diperlukan, Anda dapat menggunakan pernyataan TRUNCATE untuk mengosongkan tabel dan membebaskan ruang. Kemudian, lakukan vacuum manual pada tabel pg_attribute dan pg_depend. Dengan melakukan vacuum pada tabel ini, akan dipastikan bahwa pembuatan dan pemotongan/penghapusan tabel sementara secara terus-menerus tidak akan menambahkan tuple dan berkontribusi pada bloat sistem.

Anda dapat menghindari masalah ini saat membuat tabel sementara dengan menyertakan sintaksis berikut yang menghapus baris baru saat konten dikomit:

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

Klausa `ON COMMIT DELETE ROWS` memotong tabel sementara ketika transaksi dikomit.

## Menghindari bloat dalam indeks

Saat Anda mengubah bidang yang diindeks dalam tabel, pembaruan indeks akan menghasilkan satu atau beberapa tuple nonaktif dalam indeks tersebut. Secara default, proses autovacuum membersihkan bloat dalam indeks, tetapi pembersihan tersebut menggunakan sejumlah besar waktu dan sumber daya. Untuk menentukan preferensi pembersihan indeks saat Anda membuat tabel, sertakan klausa vacuum_index_cleanup. Secara default, pada waktu pembuatan tabel, klausa ini diatur ke AUTO, yang berarti bahwa server akan memutuskan apakah indeks Anda memerlukan pembersihan saat melakukan vaccum pada tabel. Anda dapat mengatur klausa ini ke ON untuk mengaktifkan pembersihan indeks untuk tabel tertentu, atau OFF untuk menonaktifkan pembersihan indeks untuk tabel tersebut. Ingat, menonaktifkan pembersihan indeks mungkin menghemat waktu, tetapi berpotensi menyebabkan indeks mengalami bloat.

Anda dapat mengontrol pembersihan indeks secara manual saat Anda melakukan VACUUM pada tabel di baris perintah. Untuk melakukan vacuum pada tabel dan menghapus tuple nonaktif dari indeks, sertakan klausa INDEX_CLEANUP dengan nilai ON dan nama tabel:

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;
```

```
INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Untuk melakukan vacuum pada tabel tanpa membersihkan indeks, tentukan nilai OFF:

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

## Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL

Aurora PostgreSQL sekarang mencakup kemampuan manajemen memori tingkat lanjut untuk mengoptimalkan kinerja dan ketahanan database di bawah berbagai beban kerja. Peningkatan ini membantu Aurora PostgreSQL mempertahankan ketersediaan dan daya tanggap yang konsisten, bahkan selama periode permintaan memori yang tinggi.

Fitur ini tersedia dan diaktifkan secara default dalam versi PostgreSQL Aurora berikut untuk instance yang disediakan:

- 15.3 dan semua versi minor yang lebih tinggi
- 14.8 dan versi minor yang lebih tinggi
- 13.11 dan versi minor yang lebih tinggi
- 12.15 dan versi minor yang lebih tinggi
- 11.20 dan versi minor yang lebih tinggi

Fitur ini tersedia dan diaktifkan secara default di PostgreSQL Aurora berikut untuk Aurora Serverless contoh:

- 16.3 dan semua versi minor yang lebih tinggi
- 15.7 dan semua versi minor yang lebih tinggi
- 14.12 dan versi minor yang lebih tinggi
- 13.15 dan versi minor yang lebih tinggi

Ketika beban kerja pelanggan menggunakan semua memori kosong yang tersedia, sistem operasi dapat me-restart database untuk melindungi sumber daya, yang menyebabkan tidak

tersedianya sementara. Peningkatan manajemen memori baru di Aurora PostgreSQL secara proaktif membatalkan transaksi tertentu ketika sistem mengalami tekanan memori tinggi, membantu menjaga stabilitas database.

Fitur utama dari manajemen memori yang ditingkatkan adalah sebagai berikut:

- Membatalkan transaksi basis data yang meminta lebih banyak memori ketika sistem mendekati tekanan memori kritis.
- Sistem dikatakan berada di bawah tekanan memori kritis ketika menghabiskan semua memori fisik dan akan menghabiskan swap. Dalam keadaan ini, setiap transaksi yang meminta memori akan dibatalkan agar dapat segera mengurangi tekanan memori dalam instans DB.
- Peluncur PostgreSQL penting dan pekerja latar belakang seperti pekerja autovacuum selalu dilindungi.

## Menangani parameter manajemen memori

Untuk mengaktifkan manajemen memori

Fitur ini diaktifkan secara default. Pesan kesalahan ditampilkan ketika transaksi dibatalkan karena memori tidak mencukupi seperti yang ditunjukkan pada contoh berikut:

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

Untuk mematikan manajemen memori

Untuk mematikan fitur ini, sambungkan ke cluster Aurora PostgreSQL DB dengan psql dan gunakan pernyataan SET untuk nilai parameter seperti yang disebutkan di bawah ini.

> **Note**
>
> Kami menyarankan agar Anda tetap mengaktifkan manajemen memori. Ini membantu mencegah potensi out-of-memory kesalahan yang dapat menyebabkan restart basis data yang diinduksi beban kerja karena kelelahan memori.

Tabel berikut menunjukkan cara mematikan fitur manajemen memori untuk versi PostgreSQL Aurora yang berbeda:

| Versi Aurora PostgreSQL | Parameter | Default | Perintah untuk mematikan manajemen memori di tingkat sesi |
|---|---|---|---|
| 11.20, 11.21, 12.15, 12.16, 13.11, 13.12, 14.8, 14.9, 15.3, 15.4 | `rds.memory_allocation_guard` | `false` | `SET rds.memory_allocation_guard = true;` |
| 12.17, 13.13, 14.10, 15.5, dan versi yang lebih tinggi | `rds.enable_memory_management` | `true` | `SET rds.enable_memory_management = false;` |

> **Note**
>
> `rds.memory_allocation_guard`Parameter telah usang di Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5, dan versi yang lebih tinggi.

Menyetel nilai parameter ini dalam grup parameter cluster DB mencegah kueri dibatalkan. Untuk informasi selengkapnya tentang grup parameter cluster DB, lihat.

### Batasan

- Fitur ini tidak didukung di db.t3.medium kelas instance.

## Failover cepat dengan Amazon Aurora PostgreSQL

Di bagian berikut ini, Anda dapat mempelajari cara memastikan failover terjadi secepat mungkin. Untuk memulihkan dengan cepat setelah failover, Anda dapat menggunakan manajemen cache klaster untuk klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL.

Beberapa langkah yang dapat Anda ambil untuk membuat failover beperforma cepat antara lain sebagai berikut:

- Atur keepalive Protokol Kontrol Transmisi (TCP) dengan rentang waktu singkat, untuk menghentikan kueri yang berjalan lebih lama sebelum batas waktu baca berakhir jika ada kegagalan.
- Atur batas waktu untuk caching Java Sistem Nama Domain (DNS) secara agresif. Tindakan ini akan membantu memastikan titik akhir hanya baca Aurora dapat disikluskan dengan benar melalui simpul hanya baca pada percobaan koneksi selanjutnya.
- Atur variabel batas waktu yang digunakan dalam string koneksi JDBC serendah mungkin. Gunakan objek koneksi terpisah untuk kueri berjalan singkat dan berjalan lama.
- Gunakan titik akhir baca dan tulis Aurora yang diberikan untuk terhubung ke klaster.
- Gunakan operasi API RDS untuk menguji respons aplikasi saat terjadi kegagalan sisi server. Selain itu, gunakan alat penghapusan paket untuk menguji respons aplikasi terhadap kegagalan sisi klien.
- Gunakan Driver AWS JDBC untuk memanfaatkan sepenuhnya kemampuan failover Aurora PostgreSQL. Untuk informasi selengkapnya tentang Driver AWS JDBC dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) JDBC Driver. GitHub

Hal ini dibahas secara lebih terperinci sebagai berikut.

Topik


- Mengatur parameter keepalive TCP
- Mengonfigurasi aplikasi Anda untuk failover cepat
- Menguji failover
- Contoh failover cepat di Java


## Mengatur parameter keepalive TCP

Saat Anda mengatur koneksi TCP, serangkaian pengatur waktu dikaitkan dengan koneksi. Saat pengatur waktu keepalive mencapai nol, paket probe keepalive dikirim ke titik akhir koneksi. Jika probe menerima balasan, Anda dapat mengasumsikan bahwa koneksi masih aktif dan berjalan.

Dengan mengaktifkan parameter keepalive TCP dan mengaturnya secara agresif, akan dipastikan bahwa jika klien Anda tidak dapat terhubung ke basis data, koneksi aktif apa pun akan ditutup dengan cepat. Aplikasi kemudian dapat terhubung ke titik akhir baru.

Pastikan untuk mengatur parameter keepalive TCP berikut:

- `tcp_keepalives_idle`mengontrol waktu, dalam hitungan detik, setelah itu paket keepalive dikirim ketika tidak ada data yang dikirim oleh soket. ACKs tidak dianggap sebagai data. Kami menyarankan pengaturan berikut:

  ```
  tcp_keepalives_idle = 1
  ```

- `tcp_keepalives_interval` mengontrol waktu, dalam hitungan detik, di antara waktu pengiriman paket keepalive berikutnya setelah paket awal dikirim. Atur waktu ini dengan menggunakan parameter `tcp_keepalives_idle`. Kami menyarankan pengaturan berikut:

  ```
  tcp_keepalives_interval = 1
  ```

- `tcp_keepalives_count` adalah jumlah probe keepalive yang tidak diakui yang terjadi sebelum aplikasi diberi tahu. Kami menyarankan pengaturan berikut:

  ```
  tcp_keepalives_count = 5
  ```

Pengaturan ini harus memberi tahu aplikasi dalam waktu lima detik ketika basis data berhenti merespons. Jika paket keepalive sering dihapus di dalam jaringan aplikasi, Anda dapat mengatur nilai `tcp_keepalives_count` yang lebih tinggi. Tindakan ini memungkinkan lebih banyak buffer di jaringan yang kurang andal, meskipun akan menambah waktu yang dibutuhkan untuk mendeteksi kegagalan sebenarnya.

Untuk mengatur parameter keepalive TCP di Linux

1. Uji konfigurasi parameter TCP keepalive Anda.

   Untuk melakukannya, kami merekomendasikan penggunaan baris perintah dengan perintah berikut. Konfigurasi yang disarankan ini ditujukan untuk seluruh sistem. Dengan kata lain, konfigurasi ini juga memengaruhi semua aplikasi lain yang membuat soket dengan opsi `SO_KEEPALIVE` aktif.

   ```
   sudo sysctl net.ipv4.tcp_keepalive_time=1
   sudo sysctl net.ipv4.tcp_keepalive_intvl=1
   sudo sysctl net.ipv4.tcp_keepalive_probes=5
   ```

2. Setelah Anda menemukan konfigurasi yang cocok untuk aplikasi Anda, pertahankan pengaturan ini dengan menambahkan baris berikut ke `/etc/sysctl.conf`, termasuk setiap perubahan yang Anda buat:

   ```
   tcp_keepalive_time = 1
   ```

```
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

## Mengonfigurasi aplikasi Anda untuk failover cepat

Di bagian berikut ini, Anda dapat menemukan diskusi tentang beberapa perubahan konfigurasi untuk Aurora PostgreSQL yang dapat Anda buat untuk failover cepat. Untuk mempelajari selengkapnya tentang penyiapan dan konfigurasi driver JDBC PostgreSQL, lihat dokumentasi Driver JDBC PostgreSQL.

Topik

- Mengurangi batas waktu cache DNS
- Mengatur string koneksi Aurora PostgreSQL untuk failover cepat
- Opsi lain untuk mendapatkan string host

### Mengurangi batas waktu cache DNS

Saat aplikasi Anda mencoba membuat koneksi setelah failover, penulis Aurora PostgreSQL baru akan menjadi pembaca sebelumnya. Anda dapat menemukannya dengan menggunakan titik akhir hanya baca Aurora sebelum pembaruan DNS disebarkan sepenuhnya. Mengatur time to live (TTL) DNS java ke nilai rendah, seperti kurang dari 30 detik, akan membantu siklus di antara simpul pembaca pada percobaan koneksi selanjutnya.

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

### Mengatur string koneksi Aurora PostgreSQL untuk failover cepat

Untuk menggunakan failover cepat Aurora PostgreSQL, pastikan bahwa string koneksi aplikasi Anda memiliki daftar host, bukan hanya satu host. Berikut ini adalah contoh string koneksi yang dapat Anda gunakan untuk terhubung ke klaster Aurora PostgreSQL. Dalam contoh ini, host dicetak tebal.

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

```
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=primary
```

Untuk ketersediaan terbaik dan untuk menghindari ketergantungan pada API RDS, kami sarankan Anda mempertahankan file yang digunakan untuk terhubung. File ini berisi string host yang dibaca aplikasi sejak Anda membuat koneksi ke basis data. String host ini memiliki semua titik akhir Aurora yang tersedia untuk klaster. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Koneksi titik akhir Amazon Aurora.

Misalnya, Anda dapat menyimpan titik akhir Anda dalam file lokal seperti yang ditunjukkan berikut ini.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

Aplikasi Anda akan membaca dari file ini untuk mengisi bagian host dalam string koneksi JDBC. Mengganti nama klaster DB akan menyebabkan perubahan pada titik akhir ini. Pastikan bahwa aplikasi Anda menangani peristiwa ini jika terjadi.

Opsi lainnya adalah menggunakan daftar simpul instans DB sebagai berikut.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

Kelebihan dari pendekatan ini adalah driver koneksi JDBC PostgreSQL akan melewati semua simpul pada daftar ini untuk menemukan koneksi yang valid. Sebaliknya, ketika Anda menggunakan titik akhir Aurora, hanya dua simpul yang akan dicoba pada setiap percobaan koneksi. Namun, ada kekurangan pada penggunaan simpul instans DB. Jika Anda menambahkan atau menghapus simpul dari klaster dan daftar instans titik akhir menjadi kedaluwarsa, driver koneksi mungkin tidak akan menemukan host yang tepat untuk terhubung.

Untuk membantu memastikan bahwa aplikasi Anda tidak menunggu terlalu lama untuk terhubung ke satu host, atur parameter berikut secara agresif:

- `targetServerType` – Mengontrol apakah driver terhubung ke simpul tulis atau simpul baca. Untuk memastikan bahwa aplikasi Anda hanya terhubung kembali ke simpul tulis, atur nilai `targetServerType` ke `primary`.

Nilai untuk parameter `targetServerType` mencakup `primary`, `secondary`, `any`, dan `preferSecondary`. Nilai `preferSecondary` akan mencoba membuat koneksi ke pembaca terlebih dahulu. Koneksi dibuat ke penulis jika tidak ada koneksi pembaca yang dapat dibuat.

- `loginTimeout` – Mengontrol waktu tunggu aplikasi untuk login ke basis data setelah koneksi soket dibuat.
- `connectTimeout` – Mengontrol waktu tunggu soket untuk membuat koneksi ke basis data.

Anda dapat mengubah parameter aplikasi lain untuk mempercepat proses koneksi, tergantung pada seberapa agresif perilaku aplikasi yang Anda inginkan.

- `cancelSignalTimeout` – Di beberapa aplikasi, Anda mungkin ingin mengirim sinyal pembatalan “percobaan terbaik” pada kueri yang telah habis batas waktunya. Jika sinyal pembatalan ini berada di jalur failover Anda, pertimbangkan untuk mengaturnya secara agresif agar sinyal ini tidak dikirim ke host yang nonaktif.
- `socketTimeout` – Parameter ini mengontrol waktu tunggu soket untuk operasi baca. Parameter ini dapat digunakan sebagai "batas waktu kueri" global untuk memastikan tidak ada kueri yang menunggu lebih lama dari nilai ini. Praktik yang baik adalah memiliki dua handler koneksi. Satu handler koneksi menjalankan kueri yang bermasa aktif singkat dan menetapkan nilai ini lebih rendah. Handler koneksi lain, untuk kueri yang berjalan lama, mengatur nilai ini jauh lebih tinggi. Dengan pendekatan ini, Anda dapat mengandalkan parameter keepalive TCP untuk menghentikan kueri yang berjalan lama jika server mengalami gangguan.
- `tcpKeepAlive` – Aktifkan parameter ini untuk memastikan parameter keepalive TCP yang Anda tetapkan dipatuhi.
- `loadBalanceHosts` – Saat ditetapkan ke `true`, parameter ini akan membuat aplikasi terhubung ke host acak yang dipilih dari daftar host kandidat.

Opsi lain untuk mendapatkan string host

Anda bisa mendapatkan string host dari beberapa sumber, termasuk fungsi `aurora_replica_status` dan dengan menggunakan API Amazon RDS.

Dalam banyak kasus, Anda perlu menentukan penulis klaster atau untuk menemukan simpul pembaca lainnya di klaster. Untuk melakukannya, aplikasi Anda dapat terhubung ke instans DB apa pun di klaster DB dan mengueri fungsi `aurora_replica_status`. Anda dapat menggunakan fungsi ini untuk mengurangi waktu yang dibutuhkan untuk menemukan host untuk terhubung.

Namun, dalam skenario kegagalan jaringan tertentu `aurora_replica_status` fungsi mungkin menampilkan out-of-date atau informasi yang tidak lengkap.

Cara yang tepat untuk memastikan bahwa aplikasi Anda dapat menemukan simpul untuk terhubung adalah dengan mencoba terhubung ke titik akhir penulis klaster lalu titik akhir pembaca klaster. Anda melakukannya sampai Anda dapat membuat koneksi yang dapat dibaca. Titik akhir ini tidak berubah kecuali jika Anda mengganti nama klaster DB Anda. Dengan demikian, Anda umumnya dapat membiarkannya sebagai anggota statis aplikasi Anda atau menyimpannya dalam file sumber daya yang dibaca oleh aplikasi Anda.

Setelah membuat koneksi menggunakan salah satu titik akhir ini, Anda dapat memperoleh informasi tentang klaster lainnya. Untuk melakukannya, panggil fungsi `aurora_replica_status`. Misalnya, perintah berikut mengambil informasi dengan `aurora_replica_status`.

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+-----------------
+---------------------------+-------------------------------+-----------------------
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

Misalnya, bagian host dalam string koneksi Anda mungkin dimulai dengan titik akhir klaster penulis dan pembaca, seperti yang ditunjukkan berikut.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

Dalam skenario ini, aplikasi Anda mencoba membuat koneksi ke jenis simpul apa pun, primer atau sekunder. Saat aplikasi Anda terhubung, praktik yang baik adalah memeriksa terlebih dahulu status baca/tulis simpul tersebut. Untuk melakukannya, jalankan kueri untuk hasil perintah `SHOW transaction_read_only`.

Jika nilai yang dihasilkan dari kueri adalah `OFF`, berarti Anda berhasil terhubung ke simpul primer. Namun, misalkan nilai yang dihasilkan adalah `ON` dan aplikasi Anda memerlukan koneksi baca/tulis. Dalam hal ini, Anda dapat memanggil fungsi `aurora_replica_status` untuk menentukan `server_id` yang memiliki `session_id='MASTER_SESSION_ID'`. Fungsi ini akan memberikan nama simpul primer. Anda dapat menggunakannya dengan `endpointPostfix` yang dijelaskan berikut.

Pastikan Anda mengetahui saat Anda terhubung ke replika yang memiliki data yang kedaluwarsa (stale). Ketika ini terjadi, `aurora_replica_status` fungsi mungkin menampilkan out-of-date informasi. Anda dapat menetapkan ambang batas untuk staleness di tingkat aplikasi. Untuk memeriksanya, Anda dapat melihat perbedaan antara waktu server dan nilai `last_update_timestamp`. Secara umum, aplikasi Anda harus menghindari peralihan di antara dua host karena informasi yang bertentangan yang dihasilkan oleh fungsi `aurora_replica_status`. Aplikasi Anda harus mencoba semua host yang diketahui terlebih dahulu, bukan mengikuti data yang dihasilkan oleh `aurora_replica_status`.

Daftar instance menggunakan operasi Describe DBClusters API, contoh di Java

Anda dapat menemukan daftar instance secara terprogram dengan menggunakan AWS SDK untuk Java, khususnya operasi DBClustersDeskripsikan API.

Berikut ini adalah contoh sederhana tentang cara melakukannya di Java 8.

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);

DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
   .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
 singleClusterResult.getPort()))
   .collect(Collectors.joining(","));
```

Di sini, `pgJDBCEndpointStr` berisi daftar titik akhir berformat, seperti yang ditunjukkan berikut ini.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

Variabel `endpointPostfix` dapat berupa konstanta yang ditetapkan aplikasi Anda. Atau, aplikasi Anda bisa mendapatkannya dengan mengueri operasi API `DescribeDBInstances` untuk satu instans di klaster Anda. Nilai ini tetap konstan dalam Wilayah AWS dan untuk pelanggan individu. Jadi, variabel tersebut menyimpan panggilan API hanya untuk mempertahankan konstanta ini dalam file sumber daya yang dibaca aplikasi Anda. Dalam contoh sebelumnya, variabel tersebut diatur sebagai berikut.

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

Untuk tujuan ketersediaan, praktik yang baik adalah menggunakan titik akhir Aurora pada klaster DB Anda secara default jika API ini tidak merespons atau memerlukan waktu terlalu lama untuk merespons. Titik akhir dijamin akan diperbarui selama waktu yang diperlukan untuk memperbarui catatan DNS. Pembaruan catatan DNS dengan titik akhir biasanya membutuhkan waktu kurang dari 30 detik. Anda dapat menyimpan titik akhir dalam file sumber daya yang digunakan aplikasi Anda.

## Menguji failover

Dalam semua kasus, Anda harus memiliki klaster DB yang berisi dua atau beberapa instans DB.

Dari sisi server, operasi API tertentu dapat menyebabkan pemadaman yang dapat digunakan untuk menguji respons aplikasi Anda:

- Failover DBCluster — Operasi ini mencoba mempromosikan instans DB baru di cluster DB Anda ke penulis.

  Contoh kode berikut menunjukkan cara Anda dapat menggunakan `failoverDBCluster` untuk menyebabkan pemadaman. Untuk detail selengkapnya tentang menyiapkan klien Amazon RDS, lihat Menggunakan AWS SDK for Java.

  ```
  public void causeFailover() {

      final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

      FailoverDBClusterRequest request = new FailoverDBClusterRequest();
      request.setDBClusterIdentifier("cluster-identifier");
  ```

```
    rdsClient.failoverDBCluster(request);
}
```

- Reboot DBInstance - Failover tidak dijamin dengan operasi API ini. Namun, operasi ini menonaktifkan basis data pada penulis. Anda dapat menggunakannya untuk menguji respons aplikasi Anda terhadap koneksi yang terputus. Parameter `ForceFailover` tidak berlaku untuk mesin Aurora. Sebagai gantinya, gunakan operasi API `FailoverDBCluster`.
- Modifikasi DBCluster - Memodifikasi `Port` parameter menyebabkan pemadaman ketika node di cluster mulai mendengarkan pada port baru. Secara umum, aplikasi Anda dapat merespons kegagalan ini terlebih dahulu dengan memastikan bahwa hanya aplikasi Anda yang mengontrol perubahan port. Selan itu, pastikan bahwa aplikasi Anda dapat dengan tepat memperbarui titik akhir yang diandalkannya. Anda dapat melakukannya dengan meminta seseorang memperbarui port secara manual saat melakukan perubahan di tingkat API. Atau, Anda dapat melakukannya dengan menggunakan API RDS di aplikasi Anda untuk menentukan apakah port telah berubah.
- Memodifikasi DBInstance - Memodifikasi `DBInstanceClass` parameter menyebabkan pemadaman.
- Hapus DBInstance - Menghapus primer (penulis) menyebabkan instans DB baru dipromosikan menjadi penulis di cluster DB Anda.

Dari sisi aplikasi atau klien, jika Anda menggunakan Linux, Anda dapat menguji respons aplikasi terhadap penghapusan paket yang tiba-tiba. Anda dapat melakukannya berdasarkan apakah port atau host telah berubah, atau apakah paket keepalive TCP dikirim atau diterima dengan menggunakan perintah iptables.

## Contoh failover cepat di Java

Contoh kode berikut menunjukkan cara aplikasi dapat mengatur pengelola driver Aurora PostgreSQL.

Aplikasi memanggil fungsi `getConnection` saat memerlukan koneksi. Panggilan ke `getConnection` dapat gagal menemukan host yang valid. Contohnya adalah ketika tidak ada penulis yang ditemukan, tetapi parameter `targetServerType` diatur ke `primary`. Dalam hal ini, aplikasi yang memanggil harus mencoba memanggil lagi fungsi tersebut.

Untuk menghindari mendorong perilaku coba lagi ke aplikasi, Anda dapat menggabungkan panggilan coba lagi ini ke pooler koneksi. Dengan sebagian besar pooler koneksi, Anda dapat menentukan string koneksi JDBC. Jadi, aplikasi Anda dapat memanggil `getJdbcConnectionString` dan meneruskannya ke pooler koneksi. Tindakan ini memungkinkan Anda menggunakan failover lebih cepat dengan Aurora PostgreSQL.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }

       /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
 aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different
 RO endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
       // If the lookup fails, default to something like small to retry
       java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
   }

   public Connection getConnection(String targetServerType) throws SQLException {
       return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
   }

   public Connection getConnection(String targetServerType, Duration queryTimeout)
 throws SQLException {
```

```
       Connection conn =
DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

      /*
        * A good practice is to set socket and statement timeout to be the same thing
since both
        * the client AND server will stop the query at the same time, leaving no
running queries
        * on the backend
        */
       Statement st = conn.createStatement();
       st.execute("set statement_timeout to " + queryTimeout.getMillis());
       st.close();

      return conn;
  }

  private static String urlFormat = "jdbc:postgresql://%s"
          + "/postgres"
          + "?user=%s"
          + "&password=%s"
          + "&loginTimeout=%d"
          + "&connectTimeout=%d"
          + "&cancelSignalTimeout=%d"
          + "&socketTimeout=%d"
          + "&targetServerType=%s"
          + "&tcpKeepAlive=true"
          + "&ssl=true"
          + "&loadBalanceHosts=true";
  public String getJdbcConnectionString(String targetServerType, Duration
queryTimeout) {
      return String.format(urlFormat,
                getFormattedEndpointList(getLocalEndpointList()),
                CredentialManager.getUsername(),
                CredentialManager.getPassword(),
                LOGIN_TIMEOUT.getStandardSeconds(),
                CONNECT_TIMEOUT.getStandardSeconds(),
                CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
                queryTimeout.getStandardSeconds(),
                targetServerType
      );
  }

  private List<String> getLocalEndpointList() {
```

```
        /*
          * As mentioned in the best practices doc, a good idea is to read a local
 resource file and parse the cluster endpoints.
          * For illustration purposes, the endpoint list is hardcoded here
          */
         List<String> newEndpointList = new ArrayList<>();
         newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
         newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

        return newEndpointList;
    }

    private static String getFormattedEndpointList(List<String> endpoints) {
        return IntStream.range(0, endpoints.size())
                .mapToObj(i -> endpoints.get(i).toString())
                .collect(Collectors.joining(","));
    }
}
```

## Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL

Untuk pemulihan cepat instans DB penulis di klaster Aurora PostgreSQL Anda jika terjadi failover, gunakan manajemen cache klaster untuk Amazon Aurora PostgreSQL. Manajemen cache klaster memastikan agar performa aplikasi dipertahankan jika terjadi failover.

Dalam situasi failover yang umum, Anda mungkin mendapati penurunan performa yang besar tetapi sementara setelah failover. Penurunan performa ini terjadi karena ketika instans DB failover dimulai, cache buffer kosong. Cache kosong juga dikenal sebagai cold cache. Cold cache menurunkan performa karena instans DB harus membaca dari disk yang lebih lambat, bukan memanfaatkan nilai yang tersimpan dalam cache buffer.

Dengan manajemen cache klaster, Anda menetapkan instans DB pembaca tertentu sebagai target failover. Manajemen cache klaster memastikan bahwa data dalam cache pembaca yang ditetapkan masih tersinkronisasi dengan data dalam cache instans DB penulis. Cache pembaca yang ditetapkan dengan nilai yang sudah diisi dikenal sebagai warm cache. Jika terjadi failover, pembaca yang ditetapkan akan langsung menggunakan nilai yang ada di warm cache-nya saat dipromosikan ke instans DB penulis baru. Pendekatan ini memberikan performa pemulihan yang lebih baik pada aplikasi Anda.

Manajemen cache klaster mengharuskan instans pembaca yang ditetapkan memiliki jenis kelas dan ukuran instans yang sama (contohnya, `db.r5.2xlarge` atau `db.r5.xlarge`) seperti instans penulis. Ingatlah hal ini ketika Anda membuat klaster DB Aurora PostgreSQL, sehingga klaster Anda dapat dipulihkan selama failover. Untuk daftar jenis kelas dan ukuran instans, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

> **Note**
>
> Manajemen cache cluster tidak didukung untuk cluster DB sekunder Aurora PostgreSQL yang merupakan bagian dari database global Aurora. Untuk klaster utama di mana fitur ini didukung, disarankan agar tidak ada beban kerja yang dijalankan pada pembaca tier-0 yang ditunjuk.

Daftar Isi


- Mengonfigurasi manajemen cache klaster
  - Mengaktifkan manajemen cache klaster
  - Mengatur prioritas tingkat promosi untuk instans DB penulis
  - Mengatur prioritas tingkat promosi untuk instans DB pembaca
- Memantau cache buffer
- Memecahkan masalah konfigurasi CCM


## Mengonfigurasi manajemen cache klaster

Untuk mengonfigurasi manajemen cache klaster, lakukan proses berikut ini secara berurutan.

Topik


- Mengaktifkan manajemen cache klaster
- Mengatur prioritas tingkat promosi untuk instans DB penulis
- Mengatur prioritas tingkat promosi untuk instans DB pembaca


> **Note**
>
> Berikan waktu setidaknya 1 menit setelah menyelesaikan langkah ini agar manajemen cache klaster dapat beroperasi sepenuhnya.

### Mengaktifkan manajemen cache klaster

Untuk mengaktifkan manajemen cache klaster, lakukan langkah-langkah yang dijelaskan sebagai berikut.

#### Konsol

Untuk mengaktifkan manajemen cache klaster

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter untuk klaster DB Aurora PostgreSQL Anda.

   Klaster DB harus menggunakan grup parameter selain default karena Anda tidak dapat mengubah nilai di grup parameter default.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Tetapkan nilai parameter klaster `apg_ccm_enabled` ke 1.
6. Pilih Simpan perubahan.

#### AWS CLI

Untuk mengaktifkan manajemen cache cluster untuk cluster Aurora PostgreSQL DB, gunakan perintah dengan parameter yang diperlukan berikut: AWS CLI `modify-db-cluster-parameter-group`

- `--db-cluster-parameter-group-name`
- `--parameters`

#### Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Mengatur prioritas tingkat promosi untuk instans DB penulis

Untuk manajemen cache klaster, pastikan bahwa prioritas promosi adalah tingkat-0 untuk instans DB penulis pada klaster DB Aurora PostgreSQL. Prioritas tingkat promosi adalah nilai yang menentukan urutan promosi pembaca Aurora ke instans DB penulis setelah kegagalan. Nilai yang valid adalah 0–15, dengan 0 adalah prioritas pertama dan 15 adalah prioritas terakhir. Untuk informasi selengkapnya tentang tingkat promosi, lihat Toleransi kesalahan untuk klaster DB Aurora.

Konsol

Untuk mengatur prioritas promosi instans DB penulis ke tingkat-0

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih instans DB Penulis pada klaster DB Aurora PostgreSQL.
4. Pilih Ubah. Halaman Modifikasi Instans DB akan muncul.
5. Di panel Konfigurasi tambahan, pilih tingkat-0 untuk Prioritas failover.
6. Pilih Lanjutkan dan periksa ringkasan modifikasi.
7. Untuk segera menerapkan perubahan setelah Anda menyimpannya, pilih Terapkan segera.
8. Pilih Modifikasi Instans DB untuk menyimpan perubahan Anda.

AWS CLI

Untuk menetapkan prioritas tingkat promosi ke 0 untuk instans DB penulis menggunakan AWS CLI, panggil modify-db-instanceperintah dengan parameter wajib berikut:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

Mengatur prioritas tingkat promosi untuk instans DB pembaca

Anda harus menetapkan hanya satu instans DB pembaca untuk manajemen cache cluster. Untuk melakukannya, pilih pembaca dari klaster Aurora PostgreSQL yang memiliki kelas dan ukuran instans yang sama dengan instans DB penulis. Misalnya, jika penulis menggunakan `db.r5.xlarge`, pilih pembaca yang menggunakan jenis kelas dan ukuran instans yang sama ini. Kemudian, atur prioritas tingkat promosinya ke 0.

Prioritas tingkat promosi adalah nilai yang menentukan urutan promosi pembaca Aurora ke instans DB penulis setelah kegagalan. Nilai yang valid adalah 0–15, dengan 0 adalah prioritas pertama dan 15 adalah prioritas terakhir.

Konsol

Untuk menetapkan prioritas promosi instans DB pembaca ke tingkat-0

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih instans DB Pembaca pada klaster DB Aurora PostgreSQL yang memiliki kelas instans yang sama dengan instans DB penulis.
4. Pilih Ubah. Halaman Modifikasi Instans DB akan muncul.

5. Di panel Konfigurasi tambahan, pilih tingkat-0 untuk Prioritas failover.
6. Pilih Lanjutkan dan periksa ringkasan modifikasi.
7. Untuk segera menerapkan perubahan setelah Anda menyimpannya, pilih Terapkan segera.
8. Pilih Modifikasi Instans DB untuk menyimpan perubahan Anda.

AWS CLI

Untuk menetapkan prioritas tingkat promosi ke 0 untuk instans DB pembaca menggunakan AWS CLI, panggil modify-db-instanceperintah dengan parameter wajib berikut:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

## Memantau cache buffer

Setelah mengatur manajemen cache klaster, Anda dapat memantau status sinkronisasi antara cache buffer instans DB penulis dan warm cache buffer pembaca yang ditetapkan. Untuk memeriksa konten cache buffer di instans DB penulis dan instans DB pembaca yang ditetapkan, gunakan

modul pg_buffercache PostgreSQL. Untuk informasi selengkapnya, lihat Dokumentasi pg_buffercache PostgreSQL.

Menggunakan Fungsi **aurora_ccm_status**

Manajemen cache klaster juga menyediakan fungsi aurora_ccm_status. Gunakan fungsi aurora_ccm_status pada instans DB penulis untuk mendapatkan informasi berikut tentang progres cache warming pada pembaca yang ditetapkan:

- buffers_sent_last_minute – Jumlah buffer yang telah dikirim ke pembaca yang ditetapkan dalam satu menit terakhir.
- buffers_found_last_minute – Jumlah buffer yang sering diakses yang diidentifikasi selama beberapa menit terakhir.
- buffers_sent_last_scan – Jumlah buffer yang telah dikirim ke pembaca yang ditetapkan selama pemindaian lengkap cache buffer.
- buffers_found_last_scan – Jumlah buffer yang telah diidentifikasi sering diakses dan perlu dikirim selama pemindaian lengkap terakhir cache buffer. Buffer yang sudah di-cache pada pembaca yang ditetapkan tidak akan dikirimkan.
- buffers_sent_current_scan – Jumlah buffer yang telah dikirim sejauh ini selama pemindaian saat ini.
- buffers_found_current_scan – Jumlah buffer yang telah diidentifikasi sering diakses dalam pemindaian saat ini.
- current_scan_progress – Jumlah buffer yang telah diakses sejauh ini selama pemindaian saat ini.

Contoh berikut menunjukkan cara menggunakan fungsi aurora_ccm_status untuk mengonversi sebagian output-nya menjadi warm rate dan warm percentage.

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

## Memecahkan masalah konfigurasi CCM

Ketika Anda mengaktifkan parameter apg_ccm_enabled cluster, manajemen cache cluster secara otomatis diaktifkan pada tingkat instans pada instans DB penulis dan satu instance DB pembaca

pada cluster Aurora PostgreSQL DB. Instance writer dan reader harus menggunakan tipe dan ukuran kelas instance yang sama. Prioritas tingkat promosi mereka diatur ke`0`. Instans pembaca lain di cluster DB harus memiliki tingkat promosi bukan nol dan manajemen cache cluster dinonaktifkan untuk instance tersebut.

Alasan berikut dapat menyebabkan masalah dalam konfigurasi dan menonaktifkan manajemen cache cluster:

- Ketika tidak ada instans DB pembaca tunggal yang disetel ke tingkat promosi 0.
- Ketika instance DB penulis tidak disetel ke tingkat promosi 0.
- Ketika lebih dari satu instans DB pembaca diatur ke tingkat promosi 0.
- Ketika penulis dan satu pembaca, instans DB dengan tingkat promosi 0 tidak memiliki ukuran instans yang sama.

## Mengelola churn koneksi Aurora PostgreSQL dengan pooling

Ketika aplikasi klien terhubung dan terputus begitu sering sehingga waktu respons klaster DB Aurora PostgreSQL melambat, klaster dikatakan mengalami churn koneksi. Setiap koneksi baru ke titik akhir klaster DB Aurora PostgreSQL menghabiskan sumber daya, sehingga akan mengurangi sumber daya yang dapat digunakan untuk memproses beban kerja yang sebenarnya. Churn koneksi adalah masalah yang kami sarankan agar Anda kelola dengan mengikuti beberapa praktik terbaik yang dibahas berikut.

Sebagai permulaan, Anda dapat mempersingkat waktu respons pada klaster DB Aurora PostgreSQL yang memiliki tingkat churn koneksi yang tinggi. Untuk melakukannya, Anda dapat menggunakan pooler koneksi, seperti Proksi RDS. Pooler koneksi menyediakan cache koneksi siap pakai untuk klien. Hampir semua versi Aurora PostgreSQL mendukung Proksi RDS. Untuk informasi selengkapnya, lihat Proksi Amazon RDS dengan Aurora PostgreSQL.

Jika versi spesifik Aurora PostgreSQL Anda tidak mendukung Proxy RDS, Anda dapat menggunakan pooler koneksi lain yang kompatibel dengan PostgreSQL, seperti. PgBouncer Untuk mempelajari lebih lanjut, lihat situs PgBouncerweb.

Untuk melihat apakah klaster DB Aurora PostgreSQL Anda dapat memperoleh manfaat dari pooling koneksi, Anda dapat memeriksa file `postgresql.log` untuk mengetahui apakah ada koneksi dan pemutusan koneksi. Anda juga dapat menggunakan Wawasan Performa untuk mengetahui jumlah churn koneksi yang dialami klaster DB Aurora PostgreSQL Anda. Di bagian berikut ini, Anda dapat menemukan informasi tentang kedua topik tersebut.

## Pencatatan log koneksi dan pemutusan koneksi

Parameter `log_connections` dan `log_disconnections` PostgreSQL dapat menangkap koneksi dan pemutusan koneksi ke instans penulis klaster DB Aurora PostgreSQL. Secara default, parameter ini dinonaktifkan. Untuk mengaktifkan parameter ini, gunakan grup parameter kustom dan aktifkan dengan mengubah nilainya menjadi 1. Untuk informasi selengkapnya tentang grup parameter kustom, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB. Untuk memeriksa pengaturan, hubungkan ke titik akhir klaster DB Anda untuk Aurora PostgreSQL dengan menggunakan psql dan kueri sebagai berikut.

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

Dengan kedua parameter ini diaktifkan, log akan menangkap semua koneksi dan pemutusan koneksi baru. Anda melihat pengguna dan basis data untuk setiap koneksi baru yang diotorisasi. Pada waktu pemutusan koneksi, durasi sesi juga dicatat dalam log, seperti yang ditunjukkan pada contoh berikut.

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

Untuk memeriksa churn koneksi aplikasi Anda, aktifkan parameter ini jika belum aktif. Kemudian kumpulkan data di log PostgreSQL untuk analisis dengan menjalankan aplikasi Anda dengan beban kerja dan periode waktu yang realistis. Anda dapat melihat file log di konsol RDS. Pilih instans penulis klaster DB Aurora PostgreSQL Anda, lalu pilih tab Log & peristiwa. Untuk informasi selengkapnya, lihat Melihat dan mencantumkan file log basis data.

Atau, Anda dapat mengunduh file log dari konsol dan menggunakan urutan perintah berikut. Urutan ini menemukan jumlah total koneksi yang diotorisasi dan diputus per menit.

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

Dalam contoh output ini, Anda dapat melihat lonjakan koneksi yang diotorisasi diikuti dengan pemutusan koneksi mulai dari 16:12:10.

```
.....
,.......
.........
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,.......
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
```

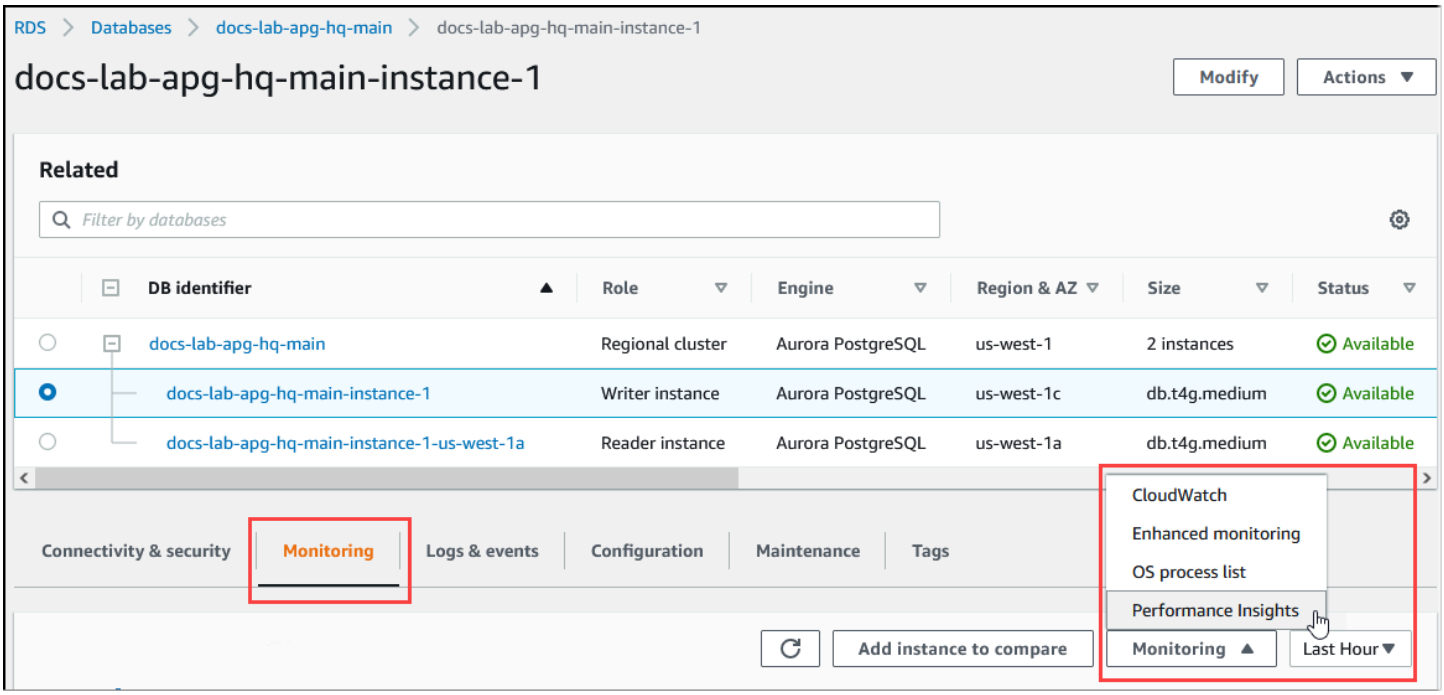
```
1 2022-03-21 16:16:00 connection authorized:
```

Dengan informasi ini, Anda dapat memutuskan apakah beban kerja Anda dapat memperoleh manfaat dari pooler koneksi. Untuk analisis yang lebih mendetail, Anda dapat menggunakan Wawasan Performa.

## Mendeteksi churn koneksi dengan Wawasan Performa

Anda dapat menggunakan Wawasan Performa untuk menilai jumlah churn koneksi pada klaster DB Aurora Edisi Kompatibel PostgreSQL. Saat Anda membuat klaster DB Aurora PostgreSQL, pengaturan untuk Wawasan Performa diaktifkan secara default. Jika Anda menghapus pilihan ini saat membuat klaster DB, ubah klaster Anda agar mengaktifkan fitur tersebut. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

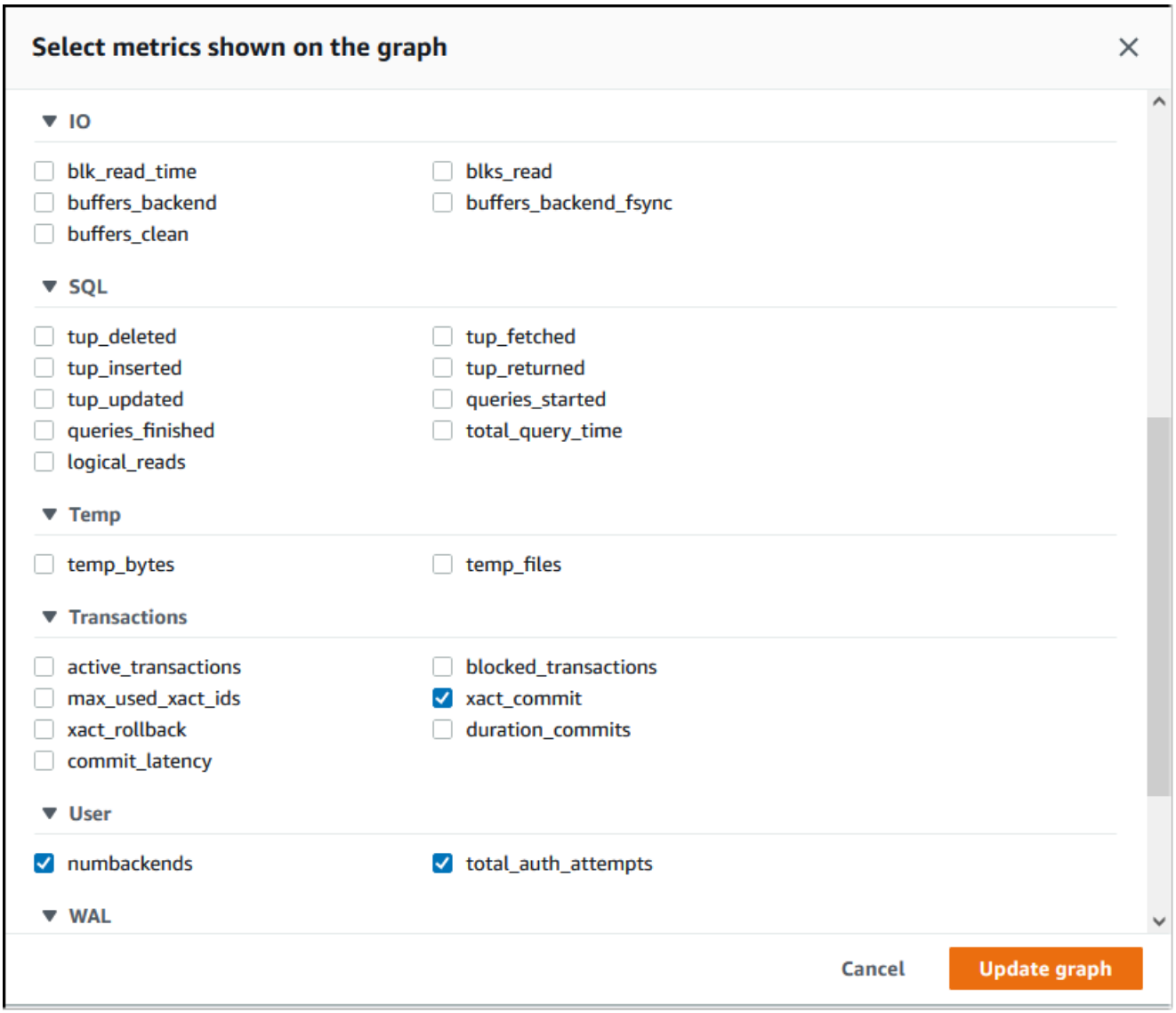
Dengan Wawasan Performa yang berjalan di klaster DB Aurora PostgreSQL, Anda dapat memilih metrik yang ingin dipantau. Anda dapat mengakses Wawasan Performa dari panel navigasi di konsol. Anda juga dapat mengakses Wawasan Performa dari tab Pemantauan untuk instans penulis klaster DB Aurora PostgreSQL Anda, seperti yang ditunjukkan pada gambar berikut.




Dari konsol Wawasan Performa, pilih Kelola metrik. Untuk menganalisis aktivitas koneksi dan pemutusan koneksi klaster DB Aurora PostgreSQL Anda, pilih metrik berikut. Ini semua adalah metrik dari PostgreSQL.

- `xact_commit` – Jumlah transaksi yang dikomit.

- `total_auth_attempts` – Jumlah percobaan koneksi pengguna yang diautentikasi per menit.
- `numbackends` – Jumlah backend yang saat ini terhubung ke basis data.

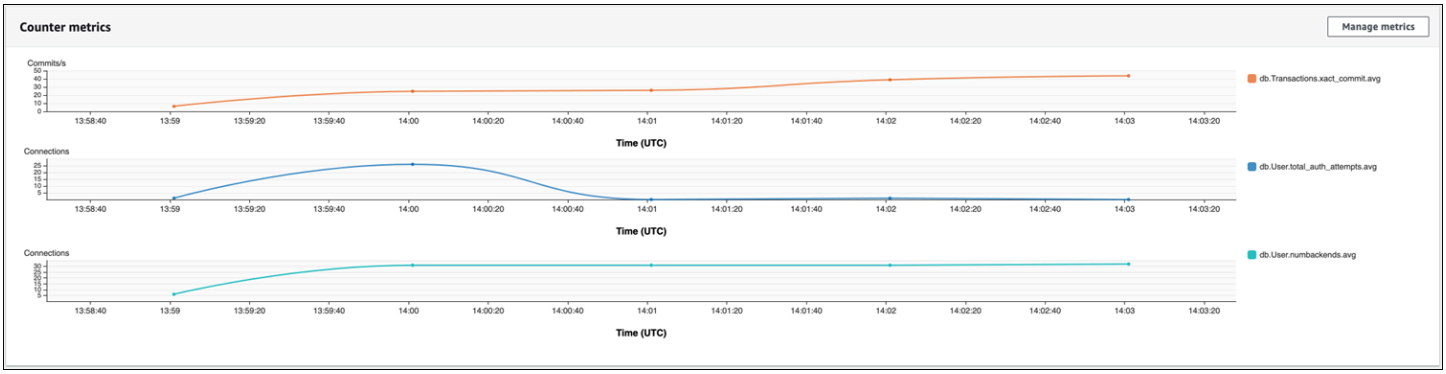



Untuk menyimpan pengaturan dan menampilkan aktivitas koneksi, pilih Perbarui grafik.

Pada gambar berikut, Anda dapat melihat dampak dari pgbench yang dijalankan dengan 100 pengguna. Garis yang menunjukkan koneksi berada pada kemiringan ke atas yang konsisten. Untuk mempelajari selengkapnya tentang pgbench dan cara menggunakannya, lihat pgbench dalam dokumentasi PostgreSQL.

Gambar ini menunjukkan bahwa menjalankan beban kerja dengan sedikitnya 100 pengguna tanpa pooler koneksi dapat menyebabkan peningkatan yang signifikan dalam jumlah `total_auth_attempts` selama durasi pemrosesan beban kerja. Perhatikan bahwa yang terbaik adalah tetap `total_auth_attempts` sedekat mungkin dengan nol.

Dengan pooling koneksi Proksi RDS, percobaan koneksi meningkat pada awal beban kerja. Setelah mengatur pool koneksi, rata-ratanya menurun. Sumber daya yang digunakan oleh transaksi dan penggunaan backend tetap konsisten selama pemrosesan beban kerja.




Untuk informasi selengkapnya tentang cara menggunakan Wawasan Performa dengan klaster DB Aurora PostgreSQL, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. Untuk menganalisis metrik, lihat Menganalisis metrik dengan dasbor Wawasan Performa.

## Menunjukkan manfaat dari pooling koneksi

Seperti disebutkan sebelumnya, jika Anda menentukan bahwa klaster DB Aurora PostgreSQL Anda memiliki masalah churn koneksi, Anda dapat menggunakan Proksi RDS untuk meningkatkan performa. Di bagian berikut ini, Anda dapat menemukan contoh yang menunjukkan perbedaan dalam memproses beban kerja saat koneksi di-pool dan saat tidak. Contoh ini menggunakan pgbench untuk memodelkan beban kerja transaksi.

Seperti psql, pgbench adalah aplikasi klien PostgreSQL yang dapat Anda instal dan jalankan dari mesin klien lokal Anda. Anda juga dapat menginstal dan menjalankannya dari EC2 instans Amazon yang Anda gunakan untuk mengelola klaster Aurora PostgreSQL DB Anda. Untuk informasi selengkapnya, lihat pgbench dalam dokumentasi PostgreSQL.

Untuk mempelajari contoh ini, pertama-tama Anda perlu membuat lingkungan pgbench dalam basis data Anda. Perintah berikut adalah templat dasar untuk menginisialisasi tabel pgbench dalam basis data yang ditentukan. Contoh ini menggunakan akun pengguna utama default, `postgres`, untuk login. Ubah akunnya sesuai kebutuhan untuk klaster DB Aurora PostgreSQL Anda. Anda membuat lingkungan pgbench dalam basis data pada instans penulis klaster Anda.

> **Note**
>
> Proses inisialisasi pgbench menghapus dan membuat ulang tabel bernama `pgbench_accounts`, `pgbench_branches`, `pgbench_history`, dan `pgbench_tellers`. Pastikan basis data yang Anda pilih untuk *dbname* ketika Anda menginisialisasi pgbench tidak menggunakan nama-nama ini.

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

Untuk pgbench, tentukan parameter berikut.

-d

Menghasilkan laporan debugging saat pgbench berjalan.

-h

Menentukan titik akhir instans penulis klaster DB Aurora PostgreSQL.

-i

Menginisialisasi lingkungan pgbench dalam basis data untuk pengujian tolok ukur.

-p

Mengidentifikasi port yang digunakan untuk koneksi basis data. Port default untuk Aurora PostgreSQL biasanya adalah 5432 atau 5433.

-s

Menentukan faktor penskalaan yang akan digunakan untuk mengisi tabel dengan baris. Faktor penskalaan default adalah 1, yang menghasilkan 1 baris dalam tabel `pgbench_branches`, 10 baris dalam tabel `pgbench_tellers`, dan 100.000 baris dalam tabel `pgbench_accounts`.

-U

Menentukan akun pengguna untuk instans penulis klaster DB Aurora PostgreSQL.

Setelah lingkungan pgbench diatur, Anda kemudian dapat menjalankan pengujian tolok ukur dengan dan tanpa pooling koneksi. Pengujian default terdiri dari serangkaian lima perintah SELECT, UPDATE, dan INSERT per transaksi yang berjalan berulang kali selama waktu yang ditentukan. Anda dapat menentukan faktor penskalaan, jumlah klien, dan detail lainnya untuk memodelkan kasus penggunaan Anda sendiri.

Sebagai contoh, perintah yang ditampilkan berikutnya akan menjalankan tolok ukur selama 60 detik (opsi -T, untuk waktu) dengan 20 koneksi konkuren (opsi -c). Opsi -C membuat pengujian menggunakan koneksi baru setiap kali dijalankan, bukan sekali per sesi klien. Pengaturan ini memberi Anda indikasi overhead koneksi.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
  transaction type: <builtin: TPC-B (sort of)>
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

Dengan menjalankan pgbench pada instans penulis klaster DB Aurora PostgreSQL tanpa menggunakan kembali koneksi, akan terlihat bahwa hanya sekitar 8 transaksi yang diproses setiap detik. Artinya, ada total 495 transaksi selama pengujian 1 menit.

Jika Anda menggunakan kembali koneksi, respons dari klaster DB Aurora PostgreSQL untuk jumlah pengguna tersebut hampir 20 kali lebih cepat. Dengan penggunaan kembali koneksi, total 9.042 transaksi diproses dibandingkan dengan 495 dalam jumlah waktu yang sama dan untuk jumlah koneksi pengguna yang sama. Perbedaannya adalah bahwa dalam contoh berikut ini, setiap koneksi digunakan kembali.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

Contoh ini menunjukkan kepada Anda bahwa pooling koneksi dapat secara signifikan mempersingkat waktu respons. Untuk informasi tentang cara menyiapkan Proksi RDS untuk klaster DB Aurora PostgreSQL Anda, lihat Proksi Amazon RDS untuk Aurora.

## Penanganan koneksi mati di PostgreSQL

Koneksi mati terjadi ketika sesi database tetap aktif di server meskipun aplikasi klien telah ditinggalkan atau dihentikan secara tidak normal. Situasi ini biasanya muncul ketika proses klien macet atau berhenti secara tak terduga tanpa menutup koneksi database mereka dengan benar atau membatalkan permintaan yang sedang berlangsung.

PostgreSQL secara efisien mengidentifikasi dan membersihkan koneksi mati saat proses server menganggur atau mencoba mengirim data ke klien. Namun, deteksi menantang untuk sesi yang menganggur, menunggu masukan klien, atau menjalankan kueri secara aktif. Untuk menangani skenario ini, PostgreSQL `tcp_keepalives_*` menyediakan,, dan parameter. `tcp_user_timeout` `client_connection_check_interval`

Topik

- Memahami TCP keepalive
- Parameter kunci TCP keepalive dalam RDS untuk PostgreSQL Aurora
- Kasus penggunaan untuk pengaturan TCP keepalive
- Praktik terbaik


## Memahami TCP keepalive

TCP Keepalive adalah mekanisme tingkat protokol yang membantu menjaga dan memverifikasi integritas koneksi. Setiap koneksi TCP mempertahankan pengaturan tingkat kernel yang mengatur perilaku keepalive. Ketika timer keepalive kedaluwarsa, sistem melakukan hal berikut:

- Mengirim paket probe tanpa data dan set flag ACK.
- Mengharapkan respons dari titik akhir jarak jauh sesuai dengan spesifikasi. TCP/IP
- Mengelola status koneksi berdasarkan respons atau kekurangannya.

## Parameter kunci TCP keepalive dalam RDS untuk PostgreSQL Aurora

| Parameter | Deskripsi | Nilai default |
|---|---|---|
| `tcp_keepalives_idle` | Menentukan jumlah detik tidak aktif sebelum mengirim pesan keepalive. | 300 |
| `tcp_keepalives_interval` | Menentukan jumlah detik antara transmisi ulang pesan keepalive yang tidak diakui. | 30 |
| `tcp_keepalives_count` | Pesan keepalive maksimum yang hilang sebelum menyatakan koneksi mati | 2 |
| `tcp_user_timeout` | Menentukan berapa lama (dalam Millidetik) data yang tidak diakui dapat tetap sebelum menutup sambungan secara paksa. | 0 |

| Parameter | Deskripsi | Nilai default |
| --- | --- | --- |
| `client_connection_check_interval` | Mengatur interval (dalam Millidetik) untuk memeriksa status koneksi klien selama kueri yang berjalan lama. Ini memastikan deteksi koneksi tertutup yang lebih cepat. | 0 |

## Kasus penggunaan untuk pengaturan TCP keepalive

Menjaga sesi idle tetap hidup

Untuk mencegah koneksi idle dihentikan oleh firewall atau router karena tidak aktif:

- Konfigurasikan `tcp_keepalives_idle` untuk mengirim paket keepalive secara berkala.

Mendeteksi koneksi mati

Untuk mendeteksi koneksi mati dengan segera:

- Sesuaikan`tcp_keepalives_idle`,`tcp_keepalives_interval`, dan`tcp_keepalives_count`. Misalnya, dengan default Aurora PostgreSQL, dibutuhkan sekitar satu menit (2 probe × 30 detik) untuk mendeteksi koneksi mati. Menurunkan nilai-nilai ini dapat mempercepat deteksi.
- Gunakan `tcp_user_timeout` untuk menentukan waktu tunggu maksimum untuk pengakuan.

Pengaturan TCP keepalive membantu kernel mendeteksi koneksi mati, tetapi PostgreSQL mungkin tidak bertindak sampai soket digunakan. Jika sesi menjalankan kueri panjang, koneksi mati mungkin hanya terdeteksi setelah kueri selesai. Di PostgreSQL 14 dan versi yang lebih tinggi`client_connection_check_interval`, dapat mempercepat deteksi koneksi mati dengan melakukan polling soket secara berkala selama eksekusi kueri.

## Praktik terbaik

- Tetapkan interval keepalive yang wajar: Tune`tcp_user_timeout`,`tcp_keepalives_idle`, `tcp_keepalives_count` dan `tcp_keepalives_interval` untuk menyeimbangkan kecepatan deteksi dan penggunaan sumber daya.
- Optimalkan untuk lingkungan Anda: Sejajarkan pengaturan dengan perilaku jaringan, kebijakan firewall, dan kebutuhan sesi.
- Memanfaatkan fitur PostgreSQL: Gunakan **`client_connection_check_interval`** di PostgreSQL 14 dan versi yang lebih tinggi untuk pemeriksaan koneksi yang efisien.

Di Amazon Aurora PostgreSQL, Anda dapat menggunakan beberapa parameter yang mengontrol jumlah memori yang digunakan untuk berbagai tugas pemrosesan. Jika tugas membutuhkan lebih banyak memori daripada jumlah yang ditetapkan untuk parameter tertentu, Aurora Postgre SQL menggunakan sumber daya lain untuk diproses, seperti dengan menulis ke disk. Ini dapat menyebabkan cluster Aurora Postgre SQL DB Anda melambat atau berpotensi berhenti, dengan kesalahan. out-of-memory

Pengaturan default untuk setiap parameter memori biasanya dapat menangani tugas pemrosesan yang dimaksudkan. Namun, Anda juga dapat menyetel cluster Aurora Postgre SQL DB Anda RDS untuk parameter terkait memori instans DB. Anda melakukan penyetelan ini untuk memastikan bahwa memori yang cukup dialokasikan untuk memproses beban kerja spesifik Anda.

Di bagian berikut ini, Anda dapat menemukan informasi tentang parameter yang mengontrol manajemen memori. Anda juga dapat mempelajari cara menilai pemanfaatan memori.

## Memeriksa dan mengatur nilai parameter

Parameter yang dapat Anda atur untuk mengelola memori dan menilai penggunaan memori cluster Aurora Postgre SQL DB Anda meliputi:

- `work_mem`— Menentukan jumlah memori yang digunakan cluster Aurora SQL Postgre DB untuk operasi pengurutan internal dan tabel hash sebelum menulis ke file disk sementara.
- `log_temp_files` – Mencatat log pembuatan file sementara, nama file, dan ukuran. Ketika parameter ini diaktifkan, entri log disimpan untuk setiap file sementara yang dibuat. Nyalakan ini untuk melihat seberapa sering cluster Aurora Postgre SQL DB Anda perlu menulis ke disk. Matikan lagi setelah Anda mengumpulkan informasi tentang pembuatan file sementara cluster Aurora Postgre SQL DB Anda, untuk menghindari logging yang berlebihan.

- `logical_decoding_work_mem`— Menentukan jumlah memori (dalam kilobyte) yang akan digunakan oleh setiap buffer penyusun ulang internal sebelum tumpah ke disk. Memori ini digunakan untuk decoding logis, yang merupakan proses untuk membuat replika. Hal ini dilakukan dengan mengkonversi data dari file write-ahead log (WAL) ke output streaming logis yang dibutuhkan oleh target.

  Nilai parameter ini membuat buffer tunggal dari ukuran yang ditentukan untuk setiap koneksi replikasi. Secara default, nilainya adalah 65536 KB. Setelah buffer ini diisi, kelebihannya akan ditulis ke disk sebagai file. Untuk meminimalkan aktivitas disk, Anda dapat mengatur nilai parameter ini ke nilai yang jauh lebih tinggi daripada `work_mem`.

Ini semua adalah parameter dinamis, sehingga Anda dapat mengubahnya untuk sesi saat ini. Untuk melakukan ini, sambungkan ke cluster Aurora Postgre SQL DB RDS untuk instance Postgre SQL dengan psql dan menggunakan pernyataan, seperti yang ditunjukkan berikut. `SET`

```
SET parameter_name TO parameter_value;
```

Pengaturan sesi hanya berlaku selama sesi berlangsung. Ketika sesi berakhir, parameter akan kembali ke pengaturannya dalam grup parameter klaster DB. Sebelum mengubah parameter apa pun, periksa terlebih dahulu nilai saat ini dengan mengueri tabel `pg_settings` sebagai berikut.

```
SELECT unit, setting, max_val
   FROM pg_settings WHERE name='parameter_name';
```

Misalnya, untuk menemukan nilai `work_mem` parameter, sambungkan ke instance penulis cluster Aurora Postgre SQL DB dan jalankan kueri berikut.

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val   | pg_size_pretty
------+----------+-----------+----------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

Untuk mengubah pengaturan parameter agar dipersistensi, grup parameter klaster DB kustom harus digunakan. Setelah menjalankan cluster Aurora Postgre SQL DB Anda untuk instance Postgre DB Anda. SQL Untuk informasi selengkapnya, lihat .

## Memahami parameter memori kerja

Parameter memori kerja (`work_mem`) menentukan jumlah maksimum memori yang dapat digunakan Aurora SQL Postgre untuk memproses kueri kompleks. Kueri kompleks mencakup kueri yang memerlukan operasi pengurutan atau pengelompokan, dengan kata lain, kueri yang menggunakan klausa berikut:

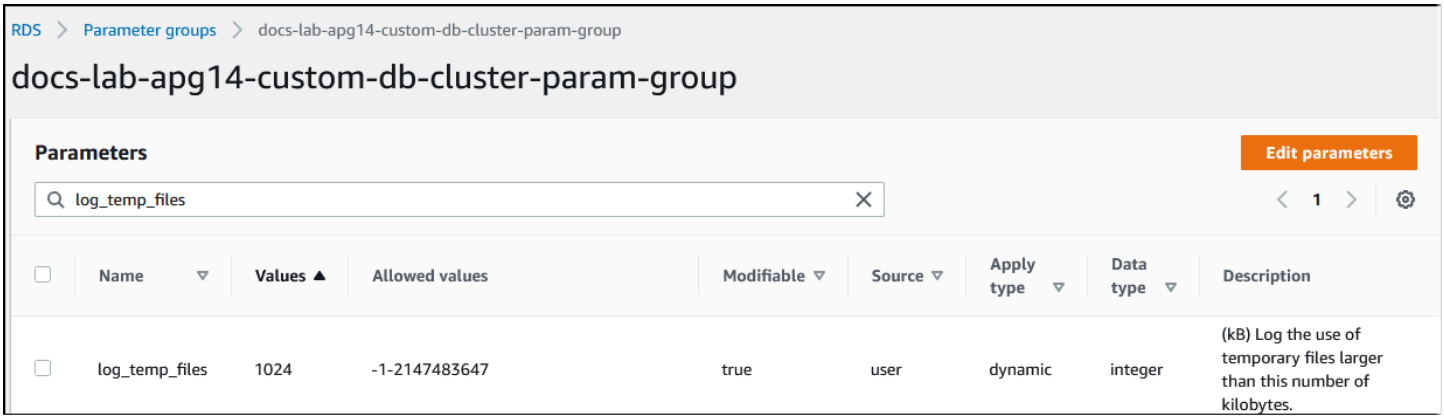
- ORDEROLEH
- DISTINCT
- GROUPOLEH
- JOIN(MERGEdanHASH)

Perencana kueri secara tidak langsung memengaruhi bagaimana cluster Aurora Postgre SQL DB Anda menggunakan memori kerja. Perencana kueri menghasilkan rencana eksekusi untuk memproses SQL pernyataan. Rencana tertentu dapat memecah kueri kompleks menjadi beberapa unit kerja yang dapat dijalankan secara paralel. Jika memungkinkan, Aurora Postgre SQL menggunakan jumlah memori yang ditentukan dalam `work_mem` parameter untuk setiap sesi sebelum menulis ke disk untuk setiap proses paralel.

Beberapa pengguna database yang menjalankan beberapa operasi secara bersamaan dan menghasilkan beberapa unit kerja secara paralel dapat menghabiskan memori kerja yang dialokasikan cluster Aurora SQL Postgre DB Anda. Hal ini dapat menyebabkan pembuatan file sementara yang berlebihan dan disk I/O, atau lebih buruk lagi, dapat menyebabkan out-of-memory kesalahan.

### Mengidentifikasi penggunaan file sementara

Setiap kali memori yang diperlukan untuk memproses kueri melebihi nilai yang ditentukan dalam parameter `work_mem`, data kerja akan dialihkan ke disk dalam file sementara. Anda dapat melihat seberapa sering ini terjadi dengan mengaktifkan parameter `log_temp_files`. Secara default, parameter ini nonaktif (diatur ke -1). Untuk menangkap semua informasi file sementara, atur parameter ini ke 0. Atur `log_temp_files` ke bilangan bulat positif lainnya untuk menangkap informasi file sementara untuk file yang sama dengan atau lebih besar dari jumlah data tersebut (dalam kilobyte). Pada gambar berikut, Anda dapat melihat contoh dari AWS Management Console.




Setelah mengonfigurasi pencatatan log file sementara, Anda dapat menguji dengan beban kerja Anda sendiri untuk melihat apakah pengaturan memori kerja Anda cukup. Anda juga dapat mensimulasikan beban kerja dengan menggunakan pgbench, aplikasi benchmarking sederhana dari komunitas Postgre. SQL

Contoh berikut menginisialisasi (`-i`) `pgbench` dengan membuat tabel dan baris yang diperlukan untuk menjalankan pengujian. Dalam contoh ini, faktor penskalaan (`-s` 50) membuat 50 baris dalam tabel `pgbench_branches`, 500 baris dalam `pgbench_tellers`, dan 5.000.000 baris dalam tabel `pgbench_accounts` di basis data `labdb`.

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

Setelah menginisialisasi lingkungan, Anda dapat menjalankan tolok ukur selama waktu (`-T`) dan jumlah klien (`-c`) tertentu. Contoh ini juga menggunakan `-d` opsi untuk mengeluarkan informasi debugging karena transaksi diproses oleh cluster Aurora SQL Postgre DB.

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

Untuk informasi lebih lanjut tentang pgbench, lihat pgbench di dokumentasi Postgre. SQL

Anda dapat menggunakan perintah metacommand psql (\d) untuk membuat daftar relasi seperti tabel, tampilan, dan indeks yang dibuat oleh pgbench.

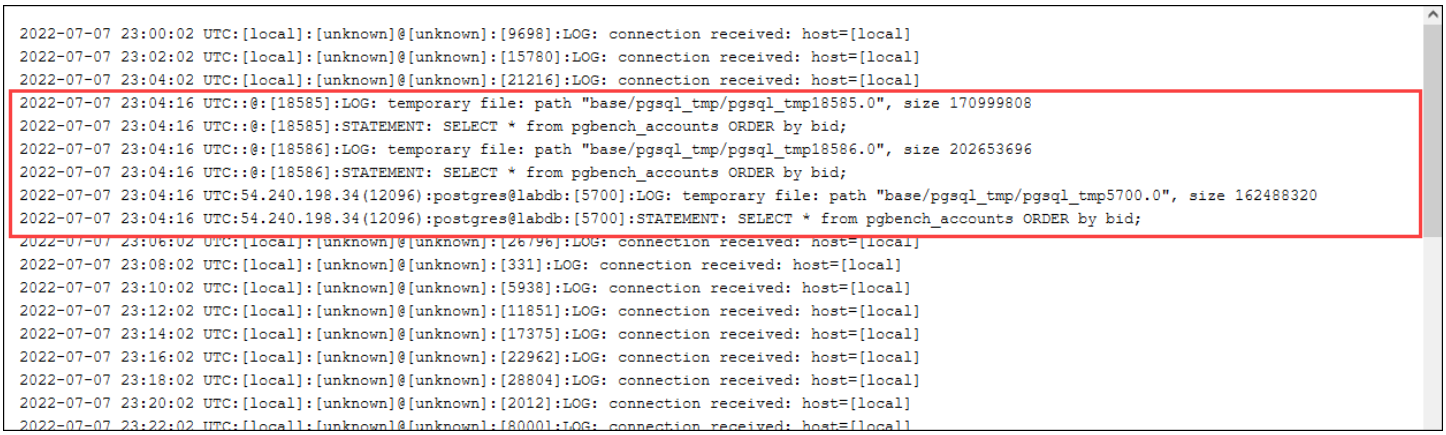
```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

Seperti yang ditunjukkan pada output, tabel `pgbench_accounts` diindeks pada kolom `aid`. Untuk memastikan bahwa kueri berikutnya menggunakan memori kerja, jalankan kueri terhadap kolom yang tidak diindeks, seperti yang ditunjukkan pada contoh berikut.

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

Periksa log apakah ada file sementara. Untuk melakukannya, buka AWS Management Console, pilih instance cluster Aurora Postgre SQL DB, lalu pilih tab Logs & Events. Lihat log di konsol atau unduh untuk analisis lebih lanjut. Seperti yang ditunjukkan pada gambar berikut, ukuran file sementara

yang diperlukan untuk memproses kueri menunjukkan bahwa Anda harus mempertimbangkan untuk meningkatkan jumlah yang ditentukan untuk parameter `work_mem`.




Anda dapat mengonfigurasi parameter ini secara berbeda untuk individu dan grup, berdasarkan kebutuhan operasional Anda. Misalnya, Anda dapat mengatur parameter `work_mem` ke 8 GB untuk peran bernama`dev_team`.

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

Dengan pengaturan untuk `work_mem` ini, peran apa pun yang merupakan anggota peran `dev_team` akan diberi alokasi hingga 8 GB memori kerja.

## Menggunakan indeks untuk waktu respons yang lebih cepat

Jika kueri Anda terlalu lama untuk memberikan hasil, Anda dapat memverifikasi bahwa indeks Anda digunakan seperti yang diharapkan. Pertama, aktifkan `\timing`, metacommand psql, sebagai berikut.

```
postgres=>  \timing on
```

Setelah menyalakan waktu, gunakan SELECT pernyataan sederhana.

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
  AS accounts;
count
-------
5000000
(1 row)
```

```
Time: 3119.049 ms (00:03.119)
```

Seperti yang ditunjukkan pada output, kueri ini membutuhkan waktu lebih dari 3 detik untuk diselesaikan. Untuk mempersingkat waktu respons, buat indeks di `pgbench_accounts` sebagai berikut.

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

Jalankan kembali kueri, dan perhatikan waktu respons yang lebih cepat. Dalam contoh ini, kueri selesai sekitar 5 kali lebih cepat, dalam waktu sekitar setengah detik.

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

## Menyesuaikan memori kerja untuk pendekodean logis

Replikasi logis telah tersedia di semua versi Aurora SQL RDS sejak diperkenalkan di SQL Postgre versi 10. SQL Saat Anda mengonfigurasi replikasi logis, Anda juga dapat mengatur parameter `logical_decoding_work_mem` untuk menentukan jumlah memori yang dapat digunakan proses pendekodean logis untuk proses pendekodean dan streaming.

Selama decoding logis, catatan write-ahead log (WAL) diubah menjadi SQL pernyataan yang kemudian dikirim ke target lain untuk replikasi logis atau tugas lain. Ketika transaksi ditulis ke WAL dan kemudian dikonversi, seluruh transaksi harus sesuai dengan nilai yang ditentukan untuk`logical_decoding_work_mem`. Secara default, parameter ini diatur ke 65536 KB. Setiap kelebihan akan ditulis ke disk. Artinya data tersebut harus dibaca ulang dari disk sebelum dapat dikirim ke tujuannya, sehingga memperlambat prosesnya secara keseluruhan.

Anda dapat menilai jumlah kelebihan transaksi dalam beban kerja Anda saat ini pada titik waktu tertentu dengan menggunakan fungsi `aurora_stat_file` seperti yang ditunjukkan pada contoh berikut.

```
SELECT split_part (filename, '/', 2)
```

```
   AS slot_name, count(1) AS num_spill_files,
   sum(used_bytes) AS slot_total_bytes,
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

Kueri ini mengembalikan jumlah dan ukuran file tumpahan pada cluster DB Aurora SQL Postgre Anda saat kueri dipanggil. Beban kerja yang berjalan lebih lama mungkin belum memiliki spill file pada disk. Untuk membuat profil beban kerja yang berjalan lama, sebaiknya Anda membuat tabel untuk menangkap informasi spill file saat beban kerja berjalan. Anda dapat membuat tabel sebagai berikut.

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

Untuk melihat bagaimana spill file digunakan selama replikasi logis, siapkan penerbit dan pelanggan lalu mulai replikasi sederhana. Untuk informasi selengkapnya, lihat Menyiapkan replikasi logis untuk cluster DB Aurora Postgre SQL Anda. Dengan replikasi yang sedang berlangsung, Anda dapat membuat pekerjaan yang menangkap hasil yang ditetapkan dari fungsi spill file `aurora_stat_file()` sebagai berikut.

```
INSERT INTO spill_file_tracking
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

Gunakan perintah psql berikut untuk menjalankan pekerjaan sekali per detik.

```
\watch 0.5
```

Saat pekerjaan sedang berjalan, hubungkan ke instans penulis dari sesi psql lain. Gunakan rangkaian pernyataan berikut untuk menjalankan beban kerja yang melebihi konfigurasi memori dan menyebabkan Aurora SQL Postgre membuat file tumpahan.

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
UPDATE 10000001
```

Pernyataan ini membutuhkan waktu beberapa menit untuk selesai. Setelah selesai, tekan tombol Ctrl dan tombol C bersama-sama untuk menghentikan fungsi pemantauan. Kemudian gunakan perintah berikut untuk membuat tabel untuk menyimpan informasi tentang penggunaan file tumpahan cluster Aurora Postgre SQL DB.

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                  spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
-------------------------------+-----------------------+--------+------------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

Output ini menunjukkan bahwa contoh yang dijalankan telah membuat lima spill file yang menggunakan memori 611 MB. Untuk menghindari penulisan ke disk, kami menyarankan agar mengatur parameter `logical_decoding_work_mem` ke ukuran memori tertinggi berikutnya, 1024.

# Menggunakan CloudWatch metrik Amazon untuk menganalisis penggunaan sumber daya untuk Aurora PostgreSQL

Aurora secara otomatis mengirimkan data metrik ke CloudWatch dalam periode 1 menit. Anda dapat menganalisis penggunaan sumber daya untuk Aurora PostgreSQL menggunakan metrik. CloudWatch Anda dapat mengevaluasi throughput jaringan dan penggunaan jaringan dengan metrik tersebut.

## Mengevaluasi throughput jaringan dengan CloudWatch

Ketika penggunaan sistem Anda mendekati batas sumber daya untuk jenis instans Anda, pemrosesan dapat melambat. Anda dapat menggunakan Wawasan CloudWatch Log untuk memantau penggunaan sumber daya penyimpanan Anda dan memastikan bahwa sumber daya yang cukup tersedia. Bila diperlukan, Anda dapat mengubah instans DB ke kelas instans yang lebih besar.

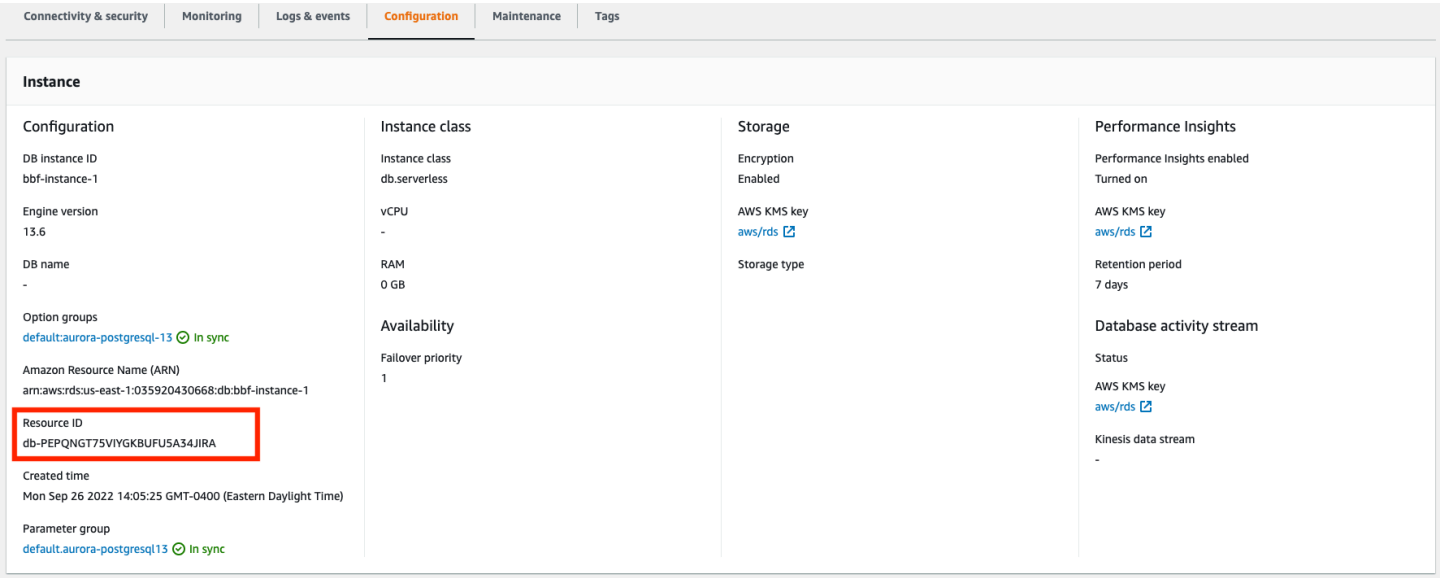
Pemrosesan penyimpanan Aurora mungkin lambat karena:

- Bandwidth jaringan tidak mencukupi antara instans DB dan klien.
- Bandwidth jaringan tidak mencukupi untuk subsistem penyimpanan.
- Beban kerja yang besar untuk jenis instans Anda.

Anda dapat meminta Wawasan CloudWatch Log untuk menghasilkan grafik penggunaan sumber daya penyimpanan Aurora untuk memantau sumber daya. Grafik tersebut menampilkan metrik dan pemanfaatan CPU untuk membantu Anda memutuskan apakah akan menaikkan skala menjadi instans yang lebih besar. Untuk informasi tentang sintaks kueri untuk Wawasan CloudWatch Log, lihat sintaks kueri Wawasan CloudWatch Log

Untuk menggunakannya CloudWatch, Anda perlu mengekspor file log Aurora PostgreSQL Anda ke file log. CloudWatch Anda juga dapat memodifikasi klaster yang ada untuk mengekspor log ke CloudWatch. Untuk informasi tentang mengekspor log ke CloudWatch, lihatMengaktifkan opsi untuk menerbitkan log ke Amazon CloudWatch.

Anda memerlukan ID Sumber Daya instans DB Anda untuk menanyakan Wawasan CloudWatch Log. ID Sumber Daya tersedia di tab Konfigurasi di konsol Anda:




Untuk mengueri file log Anda untuk metrik penyimpanan sumber daya:

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.

   Halaman beranda CloudWatch ikhtisar muncul.

2. Jika perlu, ubah Wilayah AWS. Di bilah navigasi, pilih Wilayah AWS tempat AWS sumber daya Anda berada. Untuk informasi selengkapnya, lihat Wilayah dan titik akhir.

3. Di panel navigasi, pilih Log, lalu Wawasan Log.

   Halaman Wawasan Log muncul.

4. Pilih file log dari daftar tarik-turun untuk menganalisis.

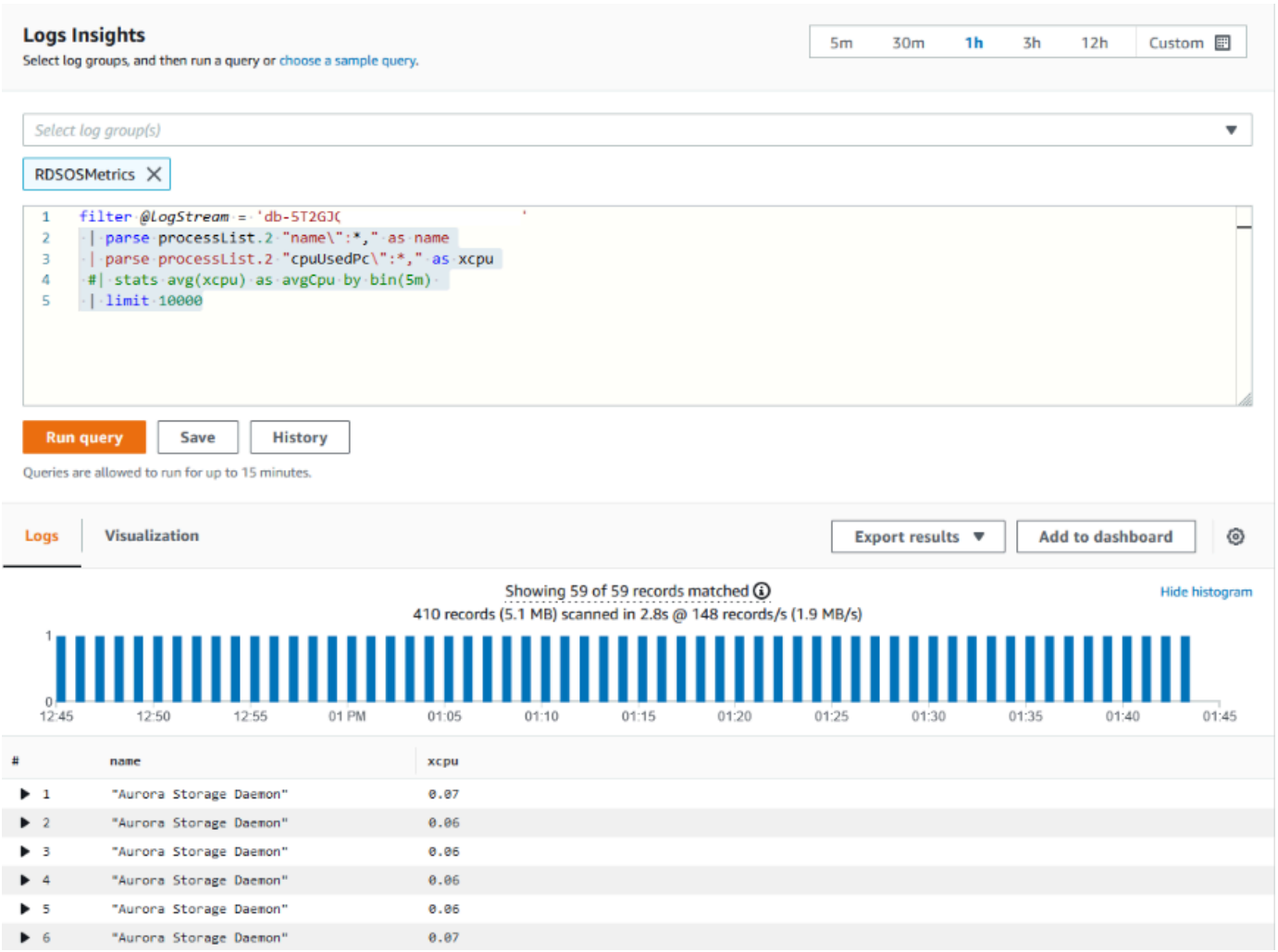
5. Masukkan kueri berikut di bidang, ganti `<resource ID>` dengan ID sumber daya klaster DB Anda:

   ```
   filter @logStream = <resource ID> | parse @message "\"Aurora Storage Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc | display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by bin(5m) | limit 10000
   ```

6. Klik Jalankan kueri.

   Grafik pemanfaatan penyimpanan ditampilkan.

   Gambar berikut menyediakan halaman Wawasan Log dan tampilan grafik.




## Mengevaluasi penggunaan instans DB untuk Aurora CloudWatch PostgreSQL dengan metrik

Anda dapat menggunakan CloudWatch metrik untuk melihat throughput instans dan mengetahui apakah kelas instans menyediakan sumber daya yang memadai untuk aplikasi Anda. Untuk informasi tentang batas kelas instans DB Anda, buka Spesifikasi perangkat keras kelas instans DB untuk Aurora dan cari spesifikasi untuk kelas instans DB Anda guna menemukan performa jaringan.

Jika penggunaan instans DB Anda mendekati batas kelas instans, maka performa mungkin mulai melambat. CloudWatch Metrik dapat mengonfirmasi situasi ini sehingga Anda dapat merencanakan untuk meningkatkan skala secara manual ke kelas instance yang lebih besar.

Gabungkan nilai CloudWatch metrik berikut untuk mengetahui apakah Anda mendekati batas kelas instance:

- NetworkThroughput— Jumlah throughput jaringan yang diterima dan ditransmisikan oleh klien untuk setiap instance di cluster Aurora DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- StorageNetworkThroughput— Jumlah throughput jaringan yang diterima dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.

Tambahkan ke NetworkThroughputStorageNetworkThroughputuntuk menemukan throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB Anda. Batas kelas instans untuk instans Anda harus lebih besar dari jumlah kedua metrik gabungan ini.

Anda dapat menggunakan metrik berikut untuk meninjau detail tambahan lalu lintas jaringan dari aplikasi klien Anda saat mengirim dan menerima:

- NetworkReceiveThroughput— Jumlah throughput jaringan yang diterima dari klien oleh setiap instance di cluster DB PostgreSQL Aurora. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- NetworkTransmitThroughput— Jumlah throughput jaringan yang dikirim ke klien oleh setiap instance di cluster Aurora DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- StorageNetworkReceiveThroughput— Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instance di cluster DB.
- StorageNetworkTransmitThroughput— Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster DB.

Tambahkan semua metrik ini bersama-sama untuk mengevaluasi bagaimana penggunaan jaringan Anda dibandingkan dengan batas kelas instans. Batas kelas instans harus lebih besar dari jumlah metrik gabungan ini.

Batas jaringan dan pemanfaatan CPU untuk penyimpanan saling berkaitan. Ketika throughput jaringan meningkat, maka pemanfaatan CPU juga meningkat. Pemantauan penggunaan CPU dan jaringan memberikan informasi tentang bagaimana dan mengapa sumber daya habis.

Untuk membantu meminimalkan penggunaan jaringan, Anda dapat mempertimbangkan:

- Menggunakan kelas instans yang lebih besar.
- Menggunakan strategi partisi `pg_partman`.

- Membagi permintaan tulis dalam batch untuk mengurangi keseluruhan transaksi.
- Mengarahkan beban kerja hanya-baca ke instans hanya-baca.
- Menghapus indeks yang tidak digunakan.
- Memeriksa objek bloat dan VACUUM. Dalam kasus bloat parah, gunakan ekstensi `pg_repack` PostgreSQL. Untuk informasi selengkapnya tentang `pg_repack`, lihat Menata ulang tabel di basis data PostgreSQL dengan kunci minimal.

## Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL

Menggunakan replikasi logis dan kloning cepat Aurora, Anda dapat melakukan upgrade versi mayor yang menggunakan versi basis data Aurora PostgreSQL saat ini sambil secara bertahap memigrasikan data yang berubah ke basis data versi mayor yang baru. Proses upgrade dengan waktu henti rendah ini disebut sebagai blue/green upgrade. Versi basis data saat ini disebut sebagai lingkungan “blue” dan versi basis data baru disebut sebagai lingkungan “green”.

Kloning cepat Aurora sepenuhnya memuat data yang ada dengan mengambil snapshot basis data sumber. Kloning cepat menggunakan copy-on-write protokol yang dibangun di atas lapisan penyimpanan Aurora, yang memungkinkan Anda membuat tiruan database dalam waktu singkat. Metode ini sangat efektif saat meng-upgrade ke basis data besar.

Replikasi logis di PostgreSQL melacak dan mentransfer perubahan data Anda dari instans awal ke instans baru yang berjalan secara paralel hingga Anda pindah ke versi PostgreSQL yang lebih baru. Replikasi logis menggunakan model "publish and subscribe". Untuk informasi selengkapnya tentang replikasi logis Aurora PostgreSQL, lihat Replikasi dengan Amazon Aurora PostgreSQL.

> **Tip**
>
> Anda dapat meminimalkan waktu henti yang diperlukan untuk upgrade versi mayor dengan menggunakan fitur Deployment Blue/Green Amazon RDS terkelola. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

Topik

- Persyaratan

- Batasan
- Mengatur dan memeriksa nilai parameter
- Meng-upgrade Aurora PostgreSQL ke versi mayor yang baru
- Melakukan tugas pasca-upgrade

## Persyaratan

Anda harus memenuhi persyaratan berikut untuk melakukan proses upgrade dengan waktu henti rendah ini:

- Anda harus memiliki izin rds_superuser.
- Klaster DB Aurora PostgreSQL yang ingin Anda upgrade harus menjalankan versi yang didukung yang dapat melakukan upgrade versi mayor menggunakan replikasi logis. Pastikan untuk menerapkan pembaruan dan patch versi minor apa pun ke klaster DB Anda. Fungsi `aurora_volume_logical_start_lsn` yang digunakan dalam teknik ini didukung dalam versi Aurora PostgreSQL berikut:
  - 15.2 dan versi 15 yang lebih tinggi
  - Versi 14.3 dan versi 14 yang lebih tinggi
  - Versi 13.6 dan versi 13 yang lebih tinggi
  - Versi 12.10 dan versi 12 yang lebih tinggi
  - Versi 11.15 dan versi 11 yang lebih tinggi
  - 10.20 dan versi 10 yang lebih tinggi

  Untuk informasi selengkapnya tentang fungsi `aurora_volume_logical_start_lsn`, lihat aurora_volume_logical_start_lsn.
- Semua tabel Anda harus memiliki kunci primer atau menyertakan kolom identitas PostgreSQL.
- Konfigurasikan grup keamanan untuk VPC Anda agar mengizinkan akses masuk dan keluar antara dua klaster DB Aurora PostgreSQL, baik yang lama maupun yang baru. Anda dapat memberikan akses ke rentang perutean antar domain tanpa kelas (CIDR) tertentu atau ke grup keamanan lain di VPC Anda atau VPC peer. (VPC peer membutuhkan koneksi peering VPC.)

> **Note**
>
> Untuk informasi terperinci tentang izin yang diperlukan untuk mengonfigurasi dan mengelola skenario replikasi logis yang berjalan, lihat Dokumentasi inti PostgreSQL.

## Batasan

Saat Anda melakukan upgrade dengan waktu henti rendah pada klaster DB Aurora PostgreSQL Anda untuk meng-upgrade-nya ke versi mayor baru, Anda menggunakan fitur replikasi logis PostgreSQL native. Fitur ini memiliki kemampuan dan batasan yang sama dengan replikasi logis PostgreSQL. Untuk informasi selengkapnya, lihat Replikasi logis PostgreSQL.

- Perintah bahasa definisi data (DDL) tidak direplikasi.
- Replikasi tidak mendukung perubahan skema dalam basis data aktif. Skema dibuat kembali dalam bentuk aslinya selama proses kloning. Jika Anda mengubah skema setelah kloning, tetapi sebelum menyelesaikan upgrade, perubahan tersebut tidak akan tercermin dalam instans yang di-upgrade.
- Objek besar tidak direplikasi, tetapi Anda dapat menyimpan data dalam tabel normal.
- Replikasi hanya didukung oleh tabel, termasuk tabel yang dipartisi. Replikasi ke jenis relasi lain, seperti tampilan, tampilan terwujud, atau tabel asing, tidak didukung.
- Data urutan tidak direplikasi dan memerlukan pembaruan manual pasca-failover.

> **Note**
>
> Upgrade ini tidak mendukung auto-scripting. Anda harus melakukan semua langkah secara manual.

## Mengatur dan memeriksa nilai parameter

Sebelum meng-upgrade, konfigurasikan instans penulis klaster DB Aurora PostgreSQL Anda untuk bertindak sebagai server penerbitan. Instans harus menggunakan grup parameter klaster DB kustom dengan pengaturan berikut:

- `rds.logical_replication` – Atur parameter ini ke 1. Parameter`rds.logical_replication` memiliki tujuan yang sama dengan parameter `wal_level` server PostgreSQL mandiri dan parameter lain yang mengontrol manajemen file write-ahead log.

- `max_replication_slots` – Atur parameter ini ke jumlah total langganan yang berencana Anda buat. Jika Anda menggunakan AWS DMS, atur parameter ini ke jumlah AWS DMS tugas yang Anda rencanakan untuk digunakan untuk pengambilan data yang diubah dari cluster DB ini.
- `max_wal_senders` – Atur ke jumlah koneksi konkuren, plus beberapa tambahan, untuk mengakomodasi tugas manajemen dan sesi baru. Jika Anda menggunakan AWS DMS, jumlah max_wal_senders harus sama dengan jumlah sesi bersamaan ditambah jumlah AWS DMS tugas yang mungkin bekerja pada waktu tertentu.
- `max_logical_replication_workers` – Atur ke jumlah pekerja replikasi logis dan pekerja sinkronisasi tabel yang Anda inginkan. Umumnya aman untuk mengatur jumlah pekerja replikasi ke nilai yang sama yang digunakan untuk max_wal_senders. Pekerja diambil dari kumpulan proses latar belakang (max_worker_processes) yang dialokasikan untuk server.
- `max_worker_processes` – Atur ke jumlah proses latar belakang untuk server. Jumlah ini harus cukup besar untuk mengalokasikan pekerja untuk replikasi, proses autovacuum, dan proses pemeliharaan lainnya yang mungkin terjadi secara bersamaan.

Saat Anda meng-upgrade ke versi Aurora PostgreSQL yang lebih baru, Anda perlu menduplikasi parameter apa pun yang Anda modifikasi di grup parameter versi sebelumnya. Parameter ini diterapkan ke versi yang di-upgrade. Anda dapat mengkueri tabel `pg_settings` untuk mendapatkan daftar pengaturan parameter sehingga Anda dapat membuatnya kembali di klaster DB Aurora PostgreSQL baru Anda.

Misalnya, untuk mendapatkan pengaturan untuk parameter replikasi, jalankan kueri berikut:

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

## Meng-upgrade Aurora PostgreSQL ke versi mayor yang baru

Untuk mempersiapkan penerbit (blue)

1. Dalam contoh berikut, instans penulis sumber (blue) adalah klaster DB Aurora PostgreSQL yang menjalankan PostgreSQL versi 11.15. Ini adalah simpul penerbitan dalam skenario replikasi kita. Untuk demonstrasi ini, instans penulis sumber kita menghosting tabel sampel yang menyimpan serangkaian nilai:

```
CREATE TABLE my_table (a int PRIMARY KEY);
INSERT INTO my_table VALUES (generate_series(1,100));
```

2. Untuk membuat penerbitan pada instans sumber, hubungkan ke simpul penulis instans dengan psql (CLI untuk PostgreSQL) atau dengan klien pilihan Anda). Masukkan perintah berikut di setiap basis data:

```
CREATE PUBLICATION publication_name FOR ALL TABLES;
```

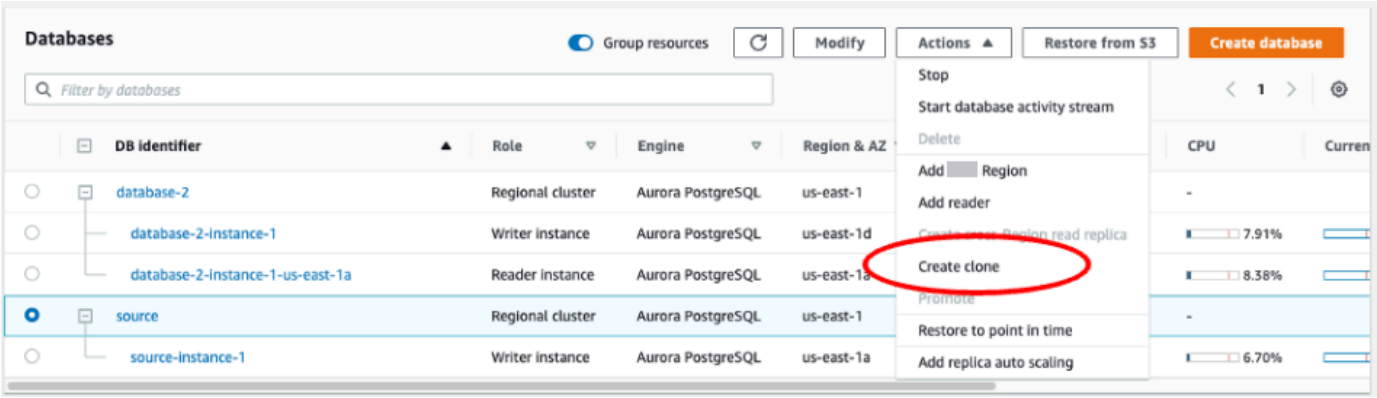
   Publication_name menentukan nama penerbitan.

3. Anda juga perlu membuat slot replikasi pada instans. Perintah berikut membuat slot replikasi dan memuat plug-in pendekodean logis `pgoutput`. Plug-in ini mengubah pembacaan konten dari write-ahead logging (WAL) ke protokol replikasi logis, dan memfilter data sesuai dengan spesifikasi penerbitan.

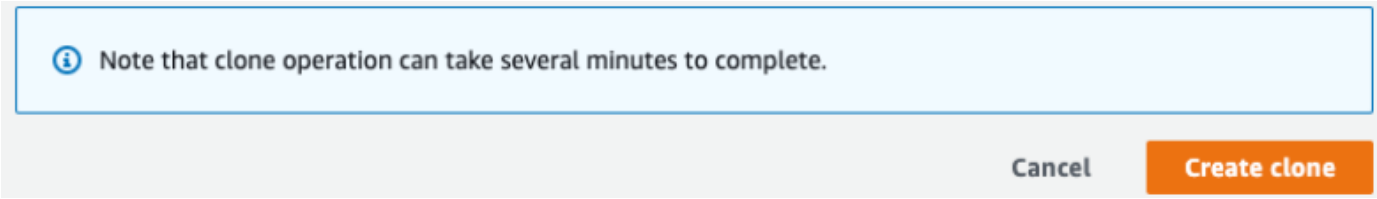
```
SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
```

Untuk mengkloning penerbit

1. Gunakan Konsol Amazon RDS untuk membuat klon dari instans sumber. Sorot nama instans di Konsol Amazon RDS, lalu pilih Buat klona di menu Tindakan.




2. Berikan nama yang unik untuk instans. Sebagian besar pengaturan adalah default dari instans sumber. Ketika Anda telah membuat perubahan yang diperlukan untuk instans baru, pilih Buat klona.




3. Saat instans target dimulai, kolom Status simpul penulis akan menampilkan Membuat di kolom Status. Saat instans siap, statusnya berubah menjadi Tersedia.

Untuk mempersiapkan klon untuk upgrade

1. Klon adalah instans 'green' dalam model deployment. Ini adalah host dari simpul langganan replikasi. Ketika simpul tersedia, hubungkan dengan psql dan kueri simpul penulis baru untuk mendapatkan nomor urutan log (LSN). LSN mengidentifikasi awal catatan di stream WAL.

```
SELECT aurora_volume_logical_start_lsn();
```

2. Dalam respons dari kueri, Anda akan menemukan nomor LSN. Anda memerlukan nomor ini nanti dalam prosesnya, jadi catatlah.

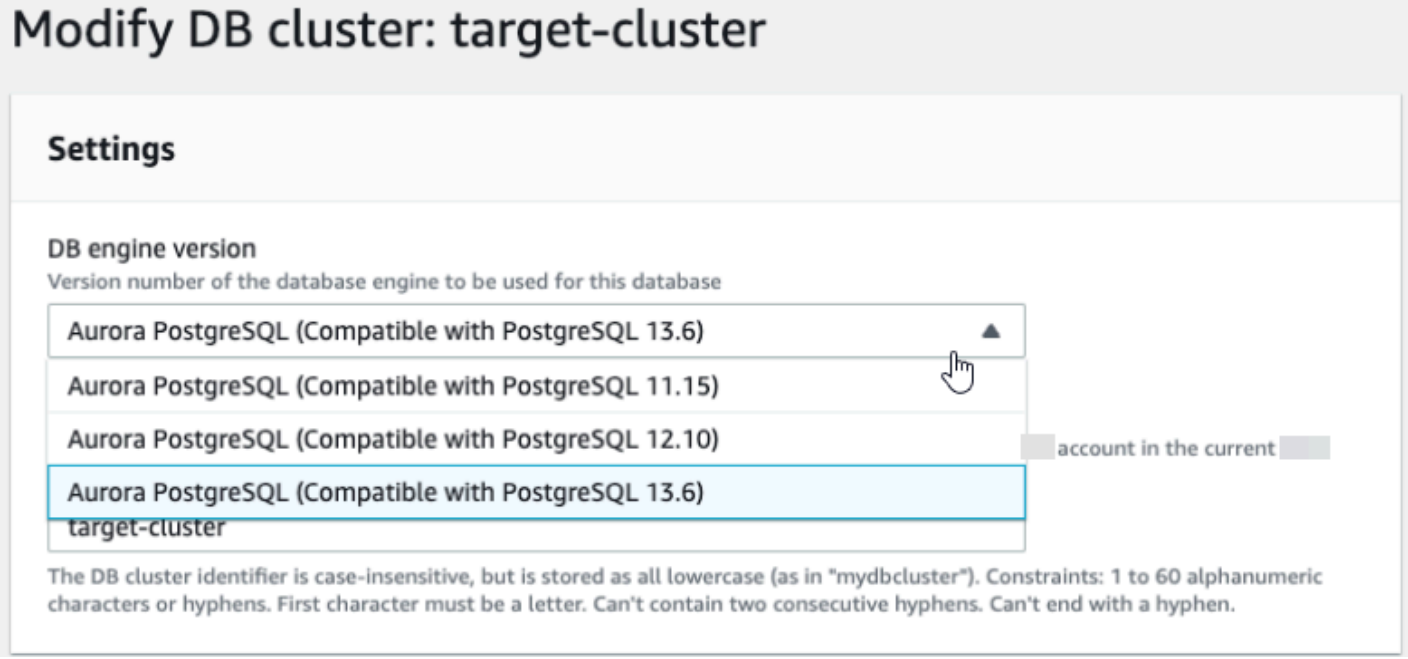
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. Sebelum meng-upgrade klon, hapus slot replikasi klon.

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

Untuk meng-upgrade klaster ke versi mayor yang baru

- Setelah mengkloning simpul penyedia, gunakan Konsol Amazon RDS untuk memulai upgrade versi mayor pada simpul langganan. Sorot nama instans di konsol RDS, dan pilih tombol Modifikasi. Pilih versi yang diperbarui dan grup parameter Anda yang diperbarui, lalu segera terapkan pengaturan untuk meng-upgrade instans target.




- Anda juga dapat menggunakan CLI untuk melakukan upgrade:

```
aws rds modify-db-cluster –db-cluster-identifier $TARGET_Aurora_ID –engine-version
 13.6 –allow-major-version-upgrade –apply-immediately
```

Untuk mempersiapkan pelanggan (hijau)

1. Saat klon tersedia setelah upgrade, hubungkan dengan psql dan tentukan langganan. Untuk melakukannya, Anda perlu menentukan opsi berikut dalam perintah `CREATE SUBSCRIPTION`:

   - `subscription_name` – Nama langganan.
   - `admin_user_name` – Nama pengguna administratif dengan izin rds_superuser.
   - `admin_user_password` – Kata sandi yang terkait dengan pengguna administratif.
   - `source_instance_URL` – URL instans server penerbitan.
   - `database` – Basis data yang akan terhubung dengan server langganan.
   - `publication_name` – Nama server penerbitan.
   - `replication_slot_name` – Nama slot replikasi.

```
CREATE SUBSCRIPTION subscription_name
CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
database' PUBLICATION publication_name
```

```
WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
 slot_name = 'replication_slot_name');
```

2. Setelah membuat langganan, jalankan kueri terhadap tampilan pg_replication_origin untuk mengambil nilai roname, yang merupakan pengidentifikasi asal replikasi. Setiap instans memiliki satu `roname`:

   ```
   SELECT * FROM pg_replication_origin;
   ```

   Sebagai contoh:

   ```
   postgres=>
   SELECT * FROM pg_replication_origin;

   roident | roname
   ---------+----------
   1 | pg_24586
   ```

3. Berikan LSN yang Anda catat dari kueri sebelumnya dari simpul penerbitan dan `roname` yang dihasilkan dari simpul langganan [INSTANCE] dalam perintah. Perintah ini menggunakan fungsi `pg_replication_origin_advance` untuk menentukan titik awal dalam urutan log untuk replikasi.

   ```
   SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
   ```

   `roname` adalah pengidentifikasi yang dihasilkan oleh tampilan pg_replication_origin.

   `log_sequence_number` adalah nilai yang dihasilkan oleh kueri sebelumnya terhadap fungsi `aurora_volume_logical_start_lsn`.

4. Kemudian, gunakan klausa `ALTER SUBSCRIPTION... ENABLE` untuk mengaktifkan replikasi logis.

   ```
   ALTER SUBSCRIPTION subscription_name ENABLE;
   ```

5. Pada tahap ini, Anda dapat mengonfirmasi bahwa replikasi berfungsi. Tambahkan nilai ke instans penerbitan, lalu konfirmasikan bahwa nilainya direplikasi ke simpul langganan.

   Kemudian, gunakan perintah berikut untuk memantau lag replikasi pada simpul penerbitan:

```
SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';
```

Sebagai contoh:

```
postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                   | slot_name             | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

Anda dapat memantau lag replikasi menggunakan nilai `diff_size` dan `diff_bytes`. Saat nilai ini mencapai 0, replika telah menyamai instans DB sumber.

## Melakukan tugas pasca-upgrade

Saat upgrade selesai, status instans ditampilkan sebagai Tersedia di kolom Status dalam dasbor konsol. Pada instans baru, kami menyarankan agar Anda melakukan hal berikut:

- Arahkan ulang aplikasi Anda untuk mengarah ke simpul penulis.
- Tambahkan simpul pembaca untuk mengelola caseload dan memberikan ketersediaan tinggi jika terjadi masalah dengan simpul penulis.
- Klaster DB Aurora PostgreSQL terkadang memerlukan pembaruan sistem operasi. Pembaruan ini mungkin menyertakan pustaka glibc versi yang lebih baru. Selama pembaruan tersebut, sebaiknya Anda mengikuti pedoman seperti yang dijelaskan dalam .
- Perbarui izin pengguna pada instans baru untuk memastikan akses.

Setelah menguji aplikasi dan data pada instans baru, kami sarankan agar Anda membuat cadangan akhir instans awal Anda sebelum menghapusnya. Untuk informasi selengkapnya tentang penggunaan replikasi logis pada host Aurora, lihat Menyiapkan replikasi logis untuk cluster DB Aurora Postgre SQL Anda.

## Memecahkan masalah penyimpanan di Aurora PostgreSQL

Jika jumlah memori kerja yang diperlukan untuk operasi pengurutan atau pembuatan indeks melampaui jumlah yang dialokasikan oleh parameter `work_mem`, Aurora PostgreSQL akan menulis kelebihan data ke file disk sementara. Ketika menulis data, Aurora PostgreSQL menggunakan ruang penyimpanan yang sama seperti yang digunakan untuk menyimpan log kesalahan dan pesan, yaitu penyimpanan lokal. Setiap instans di klaster DB Aurora PostgreSQL Anda memiliki jumlah penyimpanan lokal yang tersedia. Jumlah penyimpanan didasarkan pada kelas instans DB-nya. Untuk meningkatkan jumlah penyimpanan lokal, Anda perlu mengubah instans agar menggunakan kelas instans DB yang lebih besar. Untuk spesifikasi kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Anda dapat memantau ruang penyimpanan lokal klaster PostgreSQL DB Aurora Anda dengan menonton metrik Amazon. CloudWatch `FreeLocalStorage` Metrik ini melaporkan jumlah penyimpanan yang tersedia untuk setiap instans DB di klaster DB Aurora untuk tabel dan log sementara. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Amazon CloudWatch.

Operasi pengurutan, pengindeksan, dan pengelompokan dimulai dalam memori kerja, tetapi sering harus dialihkan ke penyimpanan lokal. Jika klaster DB Aurora PostgreSQL Anda kehabisan penyimpanan lokal karena jenis operasi ini, Anda dapat menyelesaikan masalahnya dengan mengambil salah satu tindakan berikut.

- Tingkatkan jumlah memori kerja. Tindakan ini mengurangi kebutuhan untuk menggunakan penyimpanan lokal. Secara default, PostgreSQL mengalokasikan 4 MB untuk setiap operasi pengurutan, pengelompokan, dan pengindeksan. Untuk memeriksa nilai memori kerja saat ini untuk instans penulis klaster DB Aurora PostgreSQL Anda, hubungkan ke instans menggunakan `psql` dan jalankan perintah berikut.

```
postgres=> SHOW work_mem;
work_mem
----------
 4MB
(1 row)
```

Anda dapat meningkatkan memori kerja di tingkat sesi sebelum operasi pengurutan, pengelompokan, dan lainnya, sebagai berikut.

```
SET work_mem TO '1 GB';
```

Untuk informasi selengkapnya tentang memori kerja, lihat Resource Consumption dalam dokumentasi PostgreSQL.

- Ubah periode retensi log sehingga log disimpan untuk jangka waktu yang lebih pendek. Untuk mempelajari caranya, lihat File log basis data Aurora PostgreSQL.

Untuk klaster DB Aurora PostgreSQL yang lebih besar dari 40 TB, jangan gunakan kelas instans db.t2, db.t3, atau db.t4g. Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Lihat informasi yang lebih lengkap di Jenis kelas instans DB.

# Replikasi dengan Amazon Aurora PostgreSQL

Berikut ini, Anda dapat menemukan informasi tentang replikasi dengan Amazon Aurora PostgreSQL, termasuk cara memantau dan menggunakan replikasi logis.

Topik

- Menggunakan Aurora Replica
- Meningkatkan ketersediaan baca Aurora Replica
- Memantau replikasi Aurora PostgreSQL
- Ikhtisar replikasi SQL logis Postgre dengan Aurora
- Menyiapkan replikasi logis untuk cluster DB Aurora Postgre SQL Anda
- Menggunakan replikasi logis
- Memantau cache write-through dan slot logis untuk replikasi logis Aurora Postgre SQL
- Contoh: Menggunakan replikasi logis dengan cluster Aurora Postgre DB SQL
- Contoh: Replikasi logis menggunakan SQL Aurora Postgre dan AWS Database Migration Service

# Menggunakan Aurora Replica

Aurora Replica adalah titik akhir independen dalam klaster Aurora DB, paling baik digunakan untuk menskalakan operasi baca dan meningkatkan ketersediaan. Cluster Aurora DB dapat mencakup hingga 15 Replika Aurora yang terletak di seluruh Zona Ketersediaan Wilayah cluster Aurora DB. AWS

Volume klaster DB terdiri atas beberapa salinan data untuk klaster DB. Namun, data dalam volume klaster direpresentasikan sebagai satu volume logis untuk instans DB penulis primer dan Aurora Replica di klaster DB. Untuk informasi selengkapnya tentang Aurora Replica, lihat Replika Aurora.

Aurora Replica sangat cocok untuk penskalaan baca karena ditujukan sepenuhnya untuk membaca operasi pada volume klaster Anda. Instans DB penulis mengelola operasi tulis. Volume klaster dibagikan di semua instans di klaster DB Aurora PostgreSQL Anda. Dengan demikian, tidak diperlukan kerja tambahan untuk mereplikasi salinan data untuk setiap Aurora Replica.

Dengan Aurora PostgreSQL, saat Aurora Replica dihapus, titik akhir instansnya akan segera dibuang, dan Aurora Replica akan dibuang dari titik akhir pembaca. Jika ada pernyataan yang dijalankan dari Aurora Replica yang sudah dihapus, ada tiga menit masa tenggang. Pernyataan yang ada dapat diselesaikan sepenuhnya selama masa tenggang ini. Setelah masa tenggang berakhir, Aurora Replica akan dimatikan dan dihapus.

Cluster Aurora PostgreSQL DB mendukung Aurora Replicas di berbagai Wilayah, menggunakan database global Aurora. AWS Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

> **Note**
>
> Dengan fitur ketersediaan baca, jika Anda ingin me-reboot Replika Aurora di cluster DB, Anda harus melakukannya secara manual. Untuk klaster DB yang dibuat sebelum fitur ini melakukan boot ulang, instans DB penulis secara otomatis melakukan boot ulang Aurora Replica. Boot ulang otomatis menetapkan kembali titik entri untuk memastikan konsistensi pembacaan/penulisan di seluruh klaster DB.

# Meningkatkan ketersediaan baca Aurora Replica

Aurora PostgreSQL meningkatkan ketersediaan baca di klaster DB dengan terus melayani permintaan baca ketika instans DB penulis dimulai ulang atau ketika Aurora Replica tidak dapat mengikuti lalu lintas tulis.

Fitur ketersediaan baca tersedia secara default pada versi Aurora PostgreSQL berikut:

- 16.1 dan semua versi yang lebih tinggi
- Versi 15.2 dan versi 15 yang lebih tinggi
- Versi 14.7 dan versi 14 yang lebih tinggi
- Versi 13.10 dan versi 13 yang lebih tinggi
- Versi 12.14 dan versi 12 yang lebih tinggi

Fitur ketersediaan baca didukung oleh database global Aurora dalam versi berikut:

- 16.1 dan semua versi yang lebih tinggi
- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi
- 13.12 dan 13 versi yang lebih tinggi
- 12.16 dan versi 12 yang lebih tinggi

Guna menggunakan fitur ketersediaan baca untuk klaster DB yang dibuat pada salah satu versi ini sebelum peluncuran ini, mulai ulang instans penulis klaster DB.

Saat Anda mengubah parameter statis klaster DB Aurora PostgreSQL, Anda harus memulai ulang instans penulis sehingga perubahan parameter berlaku. Misalnya, Anda harus memulai ulang instans penulis ketika Anda menetapkan nilai `shared_buffers`. Dengan fitur ketersediaan baca Aurora Replicas, cluster DB mempertahankan ketersediaan yang lebih baik, mengurangi dampaknya saat instance penulis memulai ulang. Instans pembaca tidak memulai ulang dan terus menanggapi permintaan baca. Untuk menerapkan perubahan parameter statis, mulai ulang setiap individu instans pembaca.

Aurora Replica klaster DB Aurora PostgreSQL dapat pulih dari kesalahan replikasi seperti mulai ulang penulis, failover, replikasi lambat, dan masalah jaringan dengan cepat memulihkan ke status basis data dalam memori setelah terhubung kembali dengan penulis. Pendekatan ini memungkinkan

instans Aurora Replica untuk mencapai konsistensi dengan pembaruan penyimpanan terbaru saat basis data klien masih tersedia.

Transaksi yang sedang berlangsung yang bertentangan dengan pemulihan replikasi mungkin menerima kesalahan, tetapi klien dapat mencoba kembali transaksi ini, setelah pembaca mengejar ketinggalan dengan penulis.

## Memantau Aurora Replica

Anda dapat memantau Aurora Replica saat pulih dari pemutusan penulis. Gunakan metrik di bawah ini untuk memeriksa informasi terbaru tentang instans pembaca dan untuk melacak transaksi hanya-baca dalam proses.

- `aurora_replica_status`Fungsi ini diperbarui untuk mengembalikan sebagian besar up-to-date informasi untuk instance pembaca ketika masih terhubung. Stempel waktu pembaruan terakhir di `aurora_replica_status` selalu kosong untuk baris yang sesuai dengan instans DB tempat kueri dijalankan. Ini menunjukkan bahwa instans pembaca memiliki data terbaru.
- Ketika replika Aurora terputus dari instans penulis dan menghubungkan kembali, peristiwa basis data berikut dipancarkan:

  ```
  Read replica has been disconnected from the writer instance and
  reconnected.
  ```
- Jika kueri hanya-baca dibatalkan karena konflik pemulihan, Anda mungkin melihat satu atau beberapa pesan galat berikut di log kesalahan database:

  ```
  Canceling statement due to conflict with recovery.

  User query may not have access to page data to replica disconnect.

  User query might have tried to access a file that no longer exists.

  When the replica reconnects, you will be able to repeat your command.
  ```

## Batasan

Batasan berikut berlaku untuk Aurora Replicas dengan fitur ketersediaan baca:

- Aurora Replicas dari cluster DB sekunder dapat dimulai ulang jika data tidak dapat dialirkan dari instance penulis selama pemulihan replikasi.

- Aurora Replica tidak mendukung pemulihan replikasi online jika sudah dalam proses dan akan dimulai ulang.
- Aurora Replica akan dimulai ulang ketika instans DB Anda mendekati wraparound ID transaksi. Untuk informasi selengkapnya tentang penyelesaian ID transaksi, lihat Mencegah Kegagalan Penyelesaian ID Transaksi.
- Aurora Replica dapat dimulai kembali ketika proses replikasi diblokir dalam keadaan tertentu.

## Memantau replikasi Aurora PostgreSQL

Penskalaan baca dan ketersediaan yang tinggi tergantung dari waktu lag minimal. Anda dapat memantau seberapa jauh Aurora Replica tertinggal dari instance DB penulis dari cluster DB Aurora PostgreSQL Anda dengan memantau metrik Amazon. CloudWatch `ReplicaLag` Karena Aurora Replicas membaca dari volume klaster yang sama dengan instans DB penulis, metrik `ReplicaLag` memiliki makna berbeda untuk klaster DB Aurora PostgreSQL. Metrik `ReplicaLag` untuk Replika Aurora menunjukkan lag untuk cache halaman Replika Aurora dibandingkan dengan yang ada di instans DB penulis.

Untuk informasi selengkapnya tentang pemantauan instans dan CloudWatch metrik RDS, lihat. Memantau metrik di klaster Amazon Aurora

## Ikhtisar replikasi SQL logis Postgre dengan Aurora

Dengan menggunakan fitur replikasi logis Postgre SQL dengan cluster SQL DB Aurora Postgre Anda, Anda dapat mereplikasi dan menyinkronkan tabel individual daripada seluruh instance database. Replikasi logis menggunakan model penerbit dan berlangganan untuk mereplikasi perubahan dari sumber ke satu atau lebih penerima. Ia bekerja dengan menggunakan catatan perubahan dari Postgre SQL write-ahead log (). WAL Sumber, atau penerbit, mengirimkan WAL data untuk tabel yang ditentukan ke satu atau lebih penerima (pelanggan), sehingga mereplikasi perubahan dan menjaga tabel pelanggan disinkronkan dengan tabel penerbit. Kumpulan perubahan dari penerbit diidentifikasi menggunakan publikasi. Pelanggan mendapatkan perubahan dengan membuat langganan yang mendefinisikan koneksi ke basis data penerbit dan publikasinya. Slot replikasi adalah mekanisme yang digunakan dalam skema ini untuk melacak progres langganan.

Untuk cluster Aurora Postgre SQL DB, catatan WAL disimpan di penyimpanan Aurora. Cluster Aurora Postgre SQL DB yang bertindak sebagai penerbit dalam skenario replikasi logis membaca data WAL dari penyimpanan Aurora, menerjemahkannya, dan mengirimkannya ke pelanggan sehingga perubahan dapat diterapkan ke tabel pada instance itu. Penerbit menggunakan decoder logis untuk

memecahkan kode data untuk digunakan oleh pelanggan. Secara default, cluster Aurora Postgre SQL DB menggunakan plugin SQL `pgoutput` Postgre asli saat mengirim data. Decoder logis lainnya tersedia. Sebagai contoh, Aurora Postgre SQL juga mendukung `wal2json` plugin yang mengkonversi data ke. WAL JSON

Pada Aurora Postgre SQL versi 14.5, 13.8, 12.12, dan 11.17, Aurora SQL Postgre menambah proses replikasi logis Postgre dengan cache write-through untuk meningkatkan kinerja. SQL Log WAL transaksi di-cache secara lokal, dalam buffer, untuk mengurangi jumlah disk I/O, yaitu membaca dari penyimpanan Aurora selama decoding logis. Cache write-through digunakan secara default setiap kali Anda menggunakan replikasi logis untuk cluster Aurora Postgre DB Anda. SQL Aurora menyediakan beberapa fungsi yang dapat Anda gunakan untuk mengelola cache. Untuk informasi selengkapnya, lihat Memantau cache penulisan replikasi logis Aurora Postgre SQL.

Replikasi logis didukung oleh semua versi Aurora SQL Postgre yang tersedia saat ini. Untuk informasi lebih lanjut, Amazon Aurora Postgre SQL memperbarui di Catatan Rilis untuk Aurora Postgre. SQL

Replikasi logis didukung oleh Babelfish untuk Aurora SQL Postgre dari versi berikut:

- 15.7 dan versi yang lebih tinggi
- 16.3 dan versi yang lebih tinggi

> **Note**
>
> Selain fitur replikasi SQL logis Postgre asli yang diperkenalkan di Postgre SQL 10, Aurora SQL Postgre juga mendukung ekstensi. `pglogical` Untuk informasi selengkapnya, lihat Menggunakan pglogical untuk menyinkronkan data di seluruh instans.

Untuk informasi lebih lanjut tentang replikasi SQL logis Postgre, lihat Replikasi logis dan konsep decoding logis dalam dokumentasi Postgre. SQL

## Menyiapkan replikasi logis untuk cluster DB Aurora Postgre SQL Anda

Menyiapkan replikasi logis membutuhkan `rds_superuser` hak istimewa. Cluster Aurora Postgre SQL DB Anda harus dikonfigurasi untuk menggunakan grup parameter cluster DB kustom sehingga Anda dapat mengatur parameter yang diperlukan sebagaimana dirinci dalam prosedur berikut. Untuk informasi selengkapnya, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

Untuk mengatur replikasi SQL logis Postgre untuk cluster DB Aurora Postgre SQL

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih cluster Aurora SQL Postgre DB Anda.
3. Buka tab Konfigurasi. Di antara detail Instans, temukan tautan Grup parameter dengan Grup parameter klaster DB untuk Jenis.
4. Pilih tautan untuk membuka parameter khusus yang terkait dengan cluster Aurora SQL Postgre DB Anda.
5. Di bidang pencarian Parameter, ketik `rds` untuk menemukan parameter `rds.logical_replication`. Nilai default untuk parameter ini adalah `0`, artinya fitur dinonaktifkan secara default.
6. Pilih Edit parameter untuk mengakses nilai properti, lalu pilih 1 dari pemilih untuk mengaktifkan fitur tersebut. Tergantung penggunaan yang Anda harapkan, Anda mungkin juga perlu mengubah pengaturan parameter berikut. Namun, dalam banyak kasus, nilai default sudah cukup.

   - `max_replication_slots` – Tetapkan parameter ini ke nilai yang setidaknya sama dengan jumlah total publikasi dan langganan replikasi logis yang Anda rencanakan. Jika Anda menggunakan AWS DMS, parameter ini harus sama setidaknya dengan tugas pengambilan data perubahan yang Anda rencanakan dari cluster, ditambah publikasi replikasi logis dan langganan.
   - `max_wal_sendersdan max_logical_replication_workers` — Tetapkan parameter ini ke nilai yang setidaknya sama dengan jumlah slot replikasi logis yang ingin Anda aktif, atau jumlah aktif AWS DMS tugas untuk mengubah pengambilan data. Membiarkan slot replikasi logis tidak aktif mencegah vakum menghilangkan tuple usang dari tabel, jadi kami sarankan Anda memantau slot replikasi dan menghapus slot yang tidak aktif sesuai kebutuhan.
   - `max_worker_processes` – Tetapkan parameter ini ke nilai yang setidaknya sama dengan total nilai `max_logical_replication_workers`, `autovacuum_max_workers`, dan `max_parallel_workers`. Pada kelas instans DB kecil, proses pekerja latar belakang dapat memengaruhi beban kerja aplikasi, jadi pantau performa basis data jika Anda menetapkan `max_worker_processes` lebih tinggi dari nilai default. (Nilai default adalah hasil dari`GREATEST(${DBInstanceVCPU*2},8}`, yang berarti bahwa, secara default, ini adalah 8 atau dua kali CPU setara dengan kelas instans DB, mana yang lebih besar).

> **Note**
>
> Anda dapat mengubah nilai parameter dalam grup parameter DB buatan pelanggan; Anda tidak dapat mengubah nilai parameter dalam grup parameter DB default.

7. Pilih Simpan perubahan.
8. Reboot instance penulis cluster Aurora Postgre SQL DB Anda sehingga perubahan Anda berlaku. Di RDS konsol Amazon, pilih instans DB utama cluster dan pilih Reboot dari menu Tindakan.
9. Ketika instans tersedia, Anda dapat memverifikasi bahwa replikasi logis diaktifkan sebagai berikut.

   a. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

   ```
   psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=postgres --password --dbname=labdb
   ```

   b. Verifikasi bahwa replikasi logis telah diaktifkan dengan menggunakan perintah berikut.

   ```
   labdb=> SHOW rds.logical_replication;
    rds.logical_replication
   -------------------------
    on
   (1 row)
   ```

   c. Verifikasi bahwa `wal_level` diatur ke `logical`.

   ```
   labdb=> SHOW wal_level;
     wal_level
   -----------
    logical
   (1 row)
   ```

Untuk contoh menggunakan replikasi logis untuk menjaga tabel database disinkronkan dengan perubahan dari sumber Aurora SQL Postgre DB cluster, lihat. Contoh: Menggunakan replikasi logis dengan cluster Aurora Postgre DB SQL

## Menggunakan replikasi logis

Setelah Anda menyelesaikan tugas replikasi, hentikan proses replikasi, letakkan slot replikasi, dan nonaktifkan replikasi logis. Sebelum meletakkan slot, pastikan bahwa slot tidak diperlukan lagi. Slot replikasi aktif tidak dapat diletakkan.

Untuk menonaktifkan replikasi logis

1. Letakkan semua slot replikasi.

   Untuk menjatuhkan semua slot replikasi, sambungkan ke penerbit dan jalankan perintah berikutSQL.

   ```
   SELECT pg_drop_replication_slot(slot_name)
     FROM pg_replication_slots
    WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
   ```

   Slot replikasi tidak dapat aktif saat Anda menjalankan perintah ini.
2. Ubah grup parameter klaster DB kustom yang terkait dengan penerbit seperti yang dijelaskan dalamMenyiapkan replikasi logis untuk cluster DB Aurora Postgre SQL Anda, tetapi atur parameter `rds.logical_replication` ke 0.

   Untuk informasi selengkapnya tentang grup parameter kustom, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.
3. Mulai ulang klaster DB Aurora SQL Postgre penerbit agar perubahan parameter diterapkan`rds.logical_replication`.

## Memantau cache write-through dan slot logis untuk replikasi logis Aurora Postgre SQL

Pantau cache penulisan replikasi logis dan kelola slot logis untuk meningkatkan kinerja cluster Aurora Postgre DB Anda. SQL Berikut, temukan informasi lebih lanjut tentang cache tulis dan slot logis.

Topik

- Memantau cache penulisan replikasi logis Aurora Postgre SQL
- Mengelola slot logis untuk Aurora Postgre SQL

## Memantau cache penulisan replikasi logis Aurora Postgre SQL

Secara default, Aurora Postgre SQL versi 14.5, 13.8, 12.12, dan 11.17 dan yang lebih tinggi menggunakan cache write-through untuk meningkatkan kinerja replikasi logis. Tanpa cache write-through, Aurora Postgre SQL menggunakan lapisan penyimpanan Aurora dalam implementasi proses replikasi logis Postgre asli. SQL Ia melakukannya dengan menulis WAL data ke penyimpanan dan kemudian membaca data kembali dari penyimpanan untuk memecahkan kode dan mengirim (mereplikasi) ke targetnya (pelanggan). Hal ini dapat mengakibatkan kemacetan selama replikasi logis untuk cluster Aurora Postgre DB. SQL

Cache write-through meminimalkan ketergantungan pada lapisan penyimpanan Aurora. Alih-alih secara konsisten menulis ke dan membaca dari lapisan ini, Aurora Postgre SQL menggunakan buffer untuk menyimpan WAL aliran logis untuk digunakan selama proses replikasi, mengurangi kebutuhan untuk mengakses disk. Buffer ini adalah SQL cache Postgre asli yang digunakan dalam replikasi logis dan diidentifikasi dalam parameter cluster Aurora Postgre DB sebagai. SQL `rds.logical_wal_cache`

Saat Anda menggunakan replikasi logis dengan cluster Aurora SQL Postgre DB Anda (untuk versi yang mendukung cache write-through), Anda dapat memantau rasio hit cache untuk melihat seberapa baik kerjanya untuk kasus penggunaan Anda. Untuk melakukannya, sambungkan ke instance tulis cluster Aurora Postgre SQL DB Anda menggunakan `psql` dan kemudian gunakan fungsi Auroraaurora_stat_logical_wal_cache, seperti yang ditunjukkan pada contoh berikut.

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

Fungsi mengembalikan output seperti berikut ini.

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

Nilai-nilai `last_reset_timestamp` telah dipersingkat untuk keterbacaan. Untuk informasi selengkapnya tentang fungsi ini, lihat aurora_stat_logical_wal_cache.

Aurora Postgre SQL menyediakan dua fungsi berikut untuk memantau cache write-through.

- Fungsi `aurora_stat_logical_wal_cache` – Untuk dokumentasi referensi, lihat aurora_stat_logical_wal_cache.
- Fungsi `aurora_stat_reset_wal_cache` – Untuk dokumentasi referensi, lihat aurora_stat_reset_wal_cache.

Jika Anda menemukan bahwa ukuran WAL cache yang disesuaikan secara otomatis tidak cukup untuk beban kerja Anda, Anda dapat mengubah nilai secara `rds.logical_wal_cache` manual. Pertimbangkan hal berikut:

- Ketika `rds.logical_replication` parameter dinonaktifkan, `rds.logical_wal_cache` diatur ke nol (0).
- Ketika `rds.logical_replication` parameter diaktifkan, `rds.logical_wal_cache` memiliki nilai default 16 MB.
- `rds.logical_wal_cache`Parameternya statis dan memerlukan reboot instance database agar perubahan diterapkan. Parameter ini didefinisikan dalam hal blok 8 Kb. Perhatikan bahwa setiap nilai positif kurang dari 32 Kb diperlakukan sebagai 32 Kb. Untuk informasi selengkapnya, `wal_buffers` lihat Menulis Ahead Log di dokumentasi PostgreSQL.

## Mengelola slot logis untuk Aurora Postgre SQL

Aktivitas streaming ditangkap dalam tampilan `pg_replication_origin_status`. Untuk melihat konten tampilan ini, Anda dapat menggunakan fungsi `pg_show_replication_origin_status()`, seperti yang ditunjukkan berikut:

```
SELECT * FROM pg_show_replication_origin_status();
```

Anda bisa mendapatkan daftar slot logis Anda dengan menggunakan SQL kueri berikut.

```
SELECT * FROM pg_replication_slots;
```

Untuk meletakkan slot logis, gunakan `pg_drop_replication_slot` dengan nama slot, seperti yang ditunjukkan dalam perintah berikut.

```
SELECT pg_drop_replication_slot('test_slot');
```

# Contoh: Menggunakan replikasi logis dengan cluster Aurora Postgre DB SQL

Prosedur berikut menunjukkan kepada Anda bagaimana memulai replikasi logis antara dua cluster Aurora SQL Postgre DB. Baik penerbit dan pelanggan harus dikonfigurasi untuk replikasi logis seperti yang dijelaskan dalam Menyiapkan replikasi logis untuk cluster DB Aurora Postgre SQL Anda.

Cluster Aurora Postgre SQL DB yang merupakan penerbit yang ditunjuk juga harus mengizinkan akses ke slot replikasi. Untuk melakukannya, ubah grup keamanan yang terkait dengan cloud publik virtual klaster Aurora Postgre SQL DB (VPC) berdasarkan layanan Amazon. VPC Izinkan akses masuk dengan menambahkan grup keamanan yang terkait dengan pelanggan VPC ke grup keamanan penerbit. Untuk informasi selengkapnya, lihat Mengontrol lalu lintas ke sumber daya menggunakan grup keamanan di Panduan VPC Pengguna Amazon.

Dengan langkah-langkah awal ini selesai, Anda dapat menggunakan SQL perintah Postgre `CREATE PUBLICATION` pada penerbit dan `CREATE SUBSCRIPTION` pada pelanggan, sebagaimana dirinci dalam prosedur berikut.

Untuk memulai proses replikasi logis antara dua cluster Aurora Postgre SQL DB

Langkah-langkah ini mengasumsikan bahwa cluster Aurora Postgre SQL DB Anda memiliki instance penulis dengan database untuk membuat tabel contoh.

1. Pada klaster Aurora Postgre DB SQL penerbit

   a. Buat tabel menggunakan SQL pernyataan berikut.

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. Masukkan data ke dalam database penerbit dengan menggunakan SQL pernyataan berikut.

   ```
   INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
   ```

   c. Verifikasi bahwa data ada dalam tabel dengan menggunakan SQL pernyataan berikut.

   ```
   SELECT count(*) FROM LogicalReplicationTest;
   ```

   d. Buat publikasi untuk tabel ini dengan menggunakan pernyataan `CREATE PUBLICATION`, sebagai berikut.

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

2. Pada pelanggan Aurora SQL Postgre DB cluster

   a. Buat tabel `LogicalReplicationTest` yang sama pada pelanggan yang Anda buat di penerbit, sebagai berikut.

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. Verifikasi bahwa tabel ini kosong.

   ```
   SELECT count(*) FROM LogicalReplicationTest;
   ```

   c. Buat langganan untuk mendapatkan perubahan dari penerbit. Anda perlu menggunakan detail berikut tentang klaster DB Aurora SQL Postgre penerbit.

   - host - Instans DB penulis klaster Aurora SQL Postgre DB penerbit.
   - port – Port tempat instans DB penulis mendengarkan. Default untuk Postgre SQL adalah 5432.
   - dbname – Nama basis data.

   ```
   CREATE SUBSCRIPTION testsub CONNECTION
      'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
    password=password'
     PUBLICATION testpub;
   ```

   > **Note**
   >
   > Tentukan kata sandi selain prompt yang ditampilkan di sini sebagai praktik keamanan terbaik.

   Setelah langganan dibuat, slot replikasi logis dibuat di penerbit.

   d. Untuk memverifikasi contoh ini bahwa data awal direplikasi pada pelanggan, gunakan SQL pernyataan berikut pada database pelanggan.

```
SELECT count(*) FROM LogicalReplicationTest;
```

Setiap perubahan lebih lanjut pada penerbit direplikasi ke pelanggan.

Replikasi logis memengaruhi performa. Kami menyarankan Anda menonaktifkan replikasi logis setelah tugas replikasi Anda selesai.

## Contoh: Replikasi logis menggunakan SQL Aurora Postgre dan AWS Database Migration Service

Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk mereplikasi database atau bagian dari database. Gunakan AWS DMS untuk memigrasikan data Anda dari database Aurora SQL Postgre ke database open source atau komersial lainnya. Untuk informasi lebih lanjut tentang AWS DMS, lihat AWS Database Migration Service Panduan Pengguna.

Contoh berikut menunjukkan cara mengatur replikasi logis dari database Aurora SQL Postgre sebagai penerbit dan kemudian menggunakan AWS DMS untuk migrasi. Contoh ini menggunakan penerbit dan pelanggan sama yang dibuat di Contoh: Menggunakan replikasi logis dengan cluster Aurora Postgre DB SQL.

Untuk mengatur replikasi logis dengan AWS DMS, Anda memerlukan detail tentang penerbit dan pelanggan Anda dari Amazon. RDS Secara khusus, Anda perlu detail tentang instans DB penulis penerbit dan instans DB pelanggan.

Dapatkan informasi berikut untuk instans DB penulis dari penerbit:

- Pengidentifikasi cloud pribadi virtual (VPC)
- Grup subnet
- Zona Ketersediaan (AZ)
- Kelompok VPC keamanan
- ID instans DB

Dapatkan informasi berikut untuk instans DB pelanggan:

- ID instans DB

- Mesin sumber

Untuk menggunakan AWS DMS untuk replikasi logis dengan Aurora Postgre SQL

1. Siapkan database penerbit untuk bekerja dengan AWS DMS.

   Untuk melakukan ini, Postgre SQL 10.x dan database yang lebih baru mengharuskan Anda mendaftar AWS DMS fungsi pembungkus ke database penerbit. Untuk detail tentang langkah ini dan yang lebih baru, lihat petunjuk di Menggunakan Postgre SQL versi 10.x dan yang lebih baru sebagai sumber untuk AWS DMSdi AWS Database Migration Service Panduan Pengguna.
2. Masuk ke AWS Management Console dan buka AWS DMS konsol dihttps://console.aws.amazon.com/dms/v2. Di kanan atas, pilih yang sama AWS Wilayah di mana penerbit dan pelanggan berada.
3. Buat sebuah AWS DMS contoh replikasi.

   Pilih nilai yang sama seperti nilai instans DB penulis dari penerbit Anda. Hal ini termasuk pengaturan berikut:

   - Untuk VPC, pilih yang sama VPC seperti untuk instance DB penulis.
   - Untuk Grup Subnet Replikasi, pilih grup subnet dengan nilai yang sama dengan instans DB penulis. Buat yang baru jika perlu.
   - Untuk Zona ketersediaan, pilih zona yang sama seperti untuk instans DB penulis.
   - Untuk Grup VPC Keamanan, pilih grup yang sama dengan instans DB penulis.
4. Buat sebuah AWS DMS titik akhir untuk sumbernya.

   Tentukan penerbit sebagai titik akhir sumber dengan menggunakan pengaturan berikut:

   - Untuk Jenis titik akhir, pilih Titik akhir sumber.
   - Pilih Pilih Instans RDS DB.
   - RDSMisalnya, pilih pengenal DB dari instans DB penulis penerbit.
   - Untuk Mesin sumber, pilih postgres.
5. Buat sebuah AWS DMS Endpoint untuk target.

   Tentukan pelanggan sebagai titik akhir target dengan menggunakan pengaturan berikut:

   - Untuk Jenis titik akhir, pilih Titik akhir target.
   - Pilih Pilih Instans RDS DB.

- RDSMisalnya, pilih pengidentifikasi DB dari instans DB pelanggan.
- Pilih nilai untuk Mesin sumber. Misalnya, jika pelanggan adalah SQL database RDS Postgre, pilih postgres. Jika pelanggan adalah database Aurora SQL Postgre, pilih aurora-postgresql.

6. Buat sebuah AWS DMS tugas migrasi database.

   Gunakan tugas migrasi basis data untuk menentukan tabel basis data yang akan dimigrasi, memetakan data menggunakan skema target, dan membuat tabel baru pada basis data target. Setidaknya, gunakan pengaturan berikut untuk Konfigurasi tugas:

   - Untuk Instans replikasi, pilih instans replikasi yang Anda buat pada langkah sebelumnya.
   - Untuk Titik akhir basis data sumber, pilih sumber penerbit yang Anda buat pada langkah sebelumnya.
   - Untuk Titik akhir basis data target, pilih target pelanggan yang Anda buat pada langkah sebelumnya.

   Detail tugas lainnya tergantung pada proyek migrasi Anda. Untuk informasi selengkapnya tentang menentukan semua detail DMS tugas, lihat Bekerja dengan AWS DMStugas di AWS Database Migration Service Panduan Pengguna.

Setelah AWS DMS membuat tugas, itu mulai memigrasikan data dari penerbit ke pelanggan.

# Penerusan tulis lokal di Aurora PostgreSQL

Penerusan tulis lokal (dalam klaster) memungkinkan aplikasi Anda mengeluarkan transaksi baca/tulis langsung pada Replika Aurora. Perintah tulis kemudian diteruskan ke instance DB penulis untuk dikomit. Anda dapat menggunakan penerusan tulis lokal untuk aplikasi Anda yang memiliki penulisan sesekali dan memerlukan read-after-write konsistensi, yang merupakan kemampuan untuk membaca tulisan terbaru dalam suatu transaksi.

Tanpa menulis penerusan, aplikasi Anda harus sepenuhnya membagi semua lalu lintas baca dan tulis, mempertahankan dua set koneksi database untuk mengirim lalu lintas ke titik akhir yang sesuai. Replika baca menerima pembaruan secara asinkron dari instance penulis. Selain itu, karena kelambatan replikasi dapat berbeda di antara replika baca, mencapai konsistensi baca global di semua replika sulit. Anda harus bertransaksi setiap pembacaan yang membutuhkan read-after-write konsistensi pada instance database penulis. Atau, Anda perlu mengembangkan logika aplikasi

kustom yang kompleks untuk memanfaatkan beberapa replika baca untuk skalabilitas sambil memastikan konsistensi.

Dengan penerusan tulis, Anda menghindari kebutuhan untuk membagi transaksi tersebut atau mengirimkannya secara eksklusif ke instance penulis. Anda juga tidak perlu mengembangkan logika aplikasi yang kompleks untuk mencapai read-after-write konsistensi konsistensi.

Penerusan tulis lokal tersedia di setiap Wilayah di mana Aurora PostgreSQL tersedia. Ini didukung dalam versi Aurora PostgreSQL berikut:

- 16.4 dan versi 16 yang lebih tinggi
- 15.8 dan versi 15 yang lebih tinggi
- 14.13 dan versi 14 yang lebih tinggi

Penerusan tulis lokal digunakan untuk meneruskan penulisan dari replika In-region. Untuk meneruskan penulisan dari replika global, lihatMenggunakan penerusan menulis dalam basis data global Amazon Aurora.

Topik

- Keterbatasan dan pertimbangan penerusan tulis lokal di Aurora PostgreSQL
- Mengkonfigurasi Aurora SQL Postgre untuk penerusan tulis Lokal
- Bekerja dengan penerusan tulis lokal untuk Aurora PostgreSQL
- Memantau penerusan penulisan lokal di Aurora Postgre SQL

## Keterbatasan dan pertimbangan penerusan tulis lokal di Aurora PostgreSQL

Batasan berikut saat ini berlaku untuk penerusan tulis lokal di Aurora PostgreSQL:

- Penerusan tulis lokal tidak didukung dengan RDS Proxy.
- Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil basi saat Anda menggunakannya di Aurora PostgreSQL dengan penerusan tulis. Selain itu, fungsi yang ditentukan pengguna dan prosedur yang ditentukan pengguna tidak didukung. Dengan demikian, `EnableLocalWriteForwarding` pengaturan dimatikan secara default untuk cluster DB. Sebelum mengaktifkan pengaturan tersebut, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.
- Jenis pernyataan SQL berikut ini tidak didukung dengan penerusan tulis:

> **Note**
>
> Pernyataan ini dapat digunakan secara implisit oleh Anda dalam aplikasi Anda atau disimpulkan oleh protokol PostgreSQL. Misalnya, penanganan pengecualian PL/SQL dapat mengakibatkan penggunaan SAVEPOINT, yang bukan merupakan pernyataan yang didukung.

- `ANALYZE`
- `CLUSTER`
- `COPY`
- Kursor - Kursor tidak didukung, jadi pastikan untuk menutupnya sebelum menggunakan penerusan tulis lokal.
- Pernyataan bahasa definisi data (DDL)
- `GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL`
- `LISTEN / NOTIFY`
- `LOCK`
- `SAVEPOINT`
- `SELECT INTO`
- `SET CONSTRAINTS`
- Pembaruan urutan:`nextval()`, `setval()`
- `TRUNCATE`
- Perintah komit dua fase:`PREPARE TRANSACTION`,, `COMMIT PREPARED` `ROLLBACK PREPARED`
- Fungsi yang ditentukan pengguna dan prosedur yang ditentukan pengguna.
- `VACUUM`

Anda dapat mempertimbangkan untuk menggunakan pernyataan SQL berikut dengan menulis penerusan:

- Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke instans DB tulis dan berjalan di sana.

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`.
- `EXPLAIN`pernyataan dengan pernyataan dalam daftar ini.
- Pernyataan `PREPARE` dan `EXECUTE`.
- `SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }`pernyataan.

## Mengkonfigurasi Aurora SQL Postgre untuk penerusan tulis Lokal

Dengan menggunakan bagian berikut, Anda dapat mengaktifkan penerusan tulis lokal untuk klaster DB Amazon Aurora SQL Postgre Anda, mengonfigurasi tingkat konsistensi, dan mengelola transaksi dengan penerusan tulis.

### Mengaktifkan penerusan tulis lokal

Secara default, penerusan tulis lokal tidak diaktifkan untuk klaster DB Aurora Postgre. SQL Anda mengaktifkan penerusan tulis lokal pada tingkat klaster, bukan pada tingkat instans.

Konsol

Menggunakan AWS Management Console, pilih kotak centang Aktifkan penerusan tulis lokal di bawah Baca replika tulis penerusan saat Anda membuat atau memodifikasi cluster DB.

AWS CLI

Untuk mengaktifkan penerusan tulis lokal dengan AWS CLI, gunakan opsi. `--enable-local-write-forwarding` Opsi ini berfungsi saat Anda membuat klaster DB baru menggunakan perintah `create-db-cluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster DB yang ada dengan menggunakan perintah `modify-db-cluster`. Anda dapat menonaktifkan penerusan tulis lokal dengan menggunakan `--no-enable-local-write-forwarding` opsi dengan perintah yang sama CLI ini.

Contoh berikut membuat cluster Aurora Postgre SQL DB dengan penerusan tulis lokal diaktifkan.

```
aws rds create-db-cluster \
--db-cluster-identifier write-forwarding-test-cluster \
--enable-local-write-forwarding \
--engine aurora-postgresql \
--engine-version 16.4 \
```

```
                    --master-username myuser \
                    --master-user-password mypassword \
                    --backup-retention 1
```

Anda kemudian membuat instans DB penulis dan pembaca sehingga Anda dapat menggunakan penerusan tulis. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

RDS API

Untuk mengaktifkan penerusan tulis lokal menggunakan Amazon RDSAPI, setel `EnableLocalWriteForwarding` parameter ke. `true` Parameter ini berfungsi saat Anda membuat klaster DB baru menggunakan operasi `CreateDBCluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster DB yang ada dengan menggunakan operasi `ModifyDBCluster`. Anda dapat menonaktifkan penerusan tulis lokal dengan mengatur `EnableLocalWriteForwarding` parameter ke. `false`

Mengaktifkan penerusan penulisan lokal untuk sesi database

Parameter `apg_write_forward.consistency_mode` adalah parameter DB dan parameter klaster DB yang memungkinkan penerusan tulis. Anda dapat menentukan`SESSION`,`EVENTUAL`,`GLOBAL`, atau `OFF` untuk tingkat konsistensi baca. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Konsistensi dan isolasi untuk penerusan tulis lokal di Aurora Postgre SQL.

Aturan berikut berlaku untuk parameter ini:

- Nilai default-nya adalah `SESSION`.
- Penerusan tulis lokal hanya tersedia jika Anda menyetel `apg_write_forward.consistency_mode` ke`EVENTUAL`,`SESSION`, atau. `GLOBAL` Parameter ini hanya relevan dalam instance pembaca cluster DB yang mengaktifkan penerusan tulis lokal.
- Menyetel nilai untuk `OFF` menonaktifkan penerusan tulis lokal dalam sesi.

## Konsistensi dan isolasi untuk penerusan tulis lokal di Aurora Postgre SQL

Anda dapat mengontrol tingkat konsistensi baca pada replika baca. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi penulisan yang diteruskan dari sesi Anda terlihat di replika baca sebelum kueri berikutnya. Anda juga dapat menggunakan

pengaturan ini untuk memastikan bahwa kueri pada replika baca selalu melihat pembaruan terbaru dari instans DB penulis. Hal ini bahkan berlaku untuk yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, Anda memilih nilai yang sesuai untuk parameter tingkat sesi `apg_write_forward.consistency_mode`. `apg_write_forward.consistency_mode`Parameter hanya berpengaruh pada replika baca yang mengaktifkan penerusan tulis lokal.

> **Note**
>
> Untuk parameter `apg_write_forward.consistency_mode`, Anda dapat menentukan nilai `SESSION`, `EVENTUAL`, `GLOBAL`, atau `OFF`. Secara default, nilainya diatur ke `SESSION`. Mengatur nilai untuk `OFF` menonaktifkan penerusan tulis.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu menunggu perubahan disebarkan untuk membaca replika. Anda dapat memilih keseimbangan antara latensi yang lebih rendah dan memastikan bahwa perubahan yang dibuat di lokasi lain sepenuhnya tersedia sebelum kueri dijalankan.

Dengan setiap pengaturan mode konsistensi yang tersedia, efeknya adalah sebagai berikut:

- `SESSION`— Sesi pada replika baca yang menggunakan penerusan tulis lokal melihat hasil dari semua perubahan yang dibuat dalam sesi itu. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi penulisan yang diteruskan untuk direplikasi ke instance DB pembaca saat ini. Itu tidak menunggu hasil yang diperbarui dari operasi penulisan yang dilakukan di sesi lain dalam cluster DB saat ini.
- `EVENTUAL`— Sesi pada replika baca yang menggunakan penerusan tulis lokal mungkin melihat data yang sedikit basi karena kelambatan replikasi. Hasil operasi penulisan dalam sesi yang sama tidak terlihat sampai operasi penulisan dilakukan pada instance DB penulis dan direplikasi ke replika baca. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi.
- `GLOBAL`— Sesi pada replika baca melihat perubahan yang dibuat oleh sesi itu. Ini juga melihat semua perubahan yang dilakukan dari instance DB penulis dan replika baca lainnya. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlanjut ketika replika baca up-to-date dengan semua data yang dikomit dari instance DB penulis, pada saat kueri dimulai.

> **Note**
>
> Mode konsistensi global memengaruhi latensi kueri yang dijalankan dalam satu sesi. Ini akan melakukan menunggu bahkan ketika sesi belum mengirim kueri tulis apa pun.

- `OFF`— Penerusan tulis lokal dinonaktifkan.

Dalam sesi yang menggunakan penerusan tulis, Anda dapat menggunakan tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`. Namun, tingkat isolasi `SERIALIZABLE` tidak didukung.

Untuk informasi selengkapnya tentang semua parameter yang terlibat dalam penerusan tulis, lihat Pengaturan parameter default untuk penerusan tulis.

### Mode akses transaksi dengan penerusan tulis

Jika mode akses transaksi disetel ke hanya baca, penerusan tulis lokal tidak digunakan. Anda dapat mengatur mode akses untuk membaca tulis hanya saat Anda terhubung ke cluster DB dan sesi yang mengaktifkan penerusan tulis lokal.

Untuk informasi selengkapnya tentang mode akses transaksi, lihat SETTRANSACTION.

## Bekerja dengan penerusan tulis lokal untuk Aurora PostgreSQL

Dengan menggunakan bagian berikut, Anda dapat memeriksa apakah kluster database mengaktifkan penerusan tulis lokal, melihat pertimbangan kompatibilitas, dan melihat parameter yang dapat dikonfigurasi dan pengaturan otentikasi. Informasi ini melengkapi Anda dengan detail untuk memanfaatkan fitur penerusan tulis lokal di Aurora PostgreSQL secara efektif.

> **Note**
>
> Ketika instans penulis dalam sebuah klaster yang menggunakan penerusan tulis lokal dimulai ulang, transaksi dan kueri yang aktif dan diteruskan pada instans pembaca yang menggunakan penerusan tulis lokal akan ditutup secara otomatis. Setelah instans penulis tersedia lagi, Anda dapat mencoba kembali transaksi ini.

## Memeriksa apakah cluster DB mengaktifkan penerusan tulis lokal

Untuk menentukan bahwa Anda dapat menggunakan penerusan penulisan lokal di cluster DB, konfirmasikan bahwa cluster memiliki atribut yang `LocalWriteForwardingStatus` disetel ke. `enabled`

Di AWS Management Console, pada tab Konfigurasi halaman detail untuk klaster, Anda melihat status Diaktifkan untuk penerusan tulis replika baca lokal.

Untuk melihat status setelan penerusan tulis lokal untuk semua cluster Anda, jalankan perintah berikut. AWS CLI

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
{
"LocalWriteForwardingStatus": "enabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-1"
},
{
"LocalWriteForwardingStatus": "disabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-2"
},
{
"LocalWriteForwardingStatus": "requested",
"DBClusterIdentifier": "test-global-cluster-2"
},
{
"LocalWriteForwardingStatus": "null",
"DBClusterIdentifier": "aurora-postgresql-v2-cluster"
}
]
```

Sebuah klaster DB dapat memiliki nilai berikut untuk `LocalWriteForwardingStatus`:

- `disabled`— Penerusan tulis lokal dinonaktifkan.
- `disabling`— Penerusan tulis lokal sedang dalam proses dinonaktifkan.

- `enabled`— Penerusan tulis lokal diaktifkan.
- `enabling`— Penerusan tulis lokal sedang dalam proses diaktifkan.
- `null`— Penerusan tulis lokal tidak tersedia untuk cluster DB ini.
- `requested`— Penerusan tulis lokal telah diminta, tetapi belum aktif.

## Pengaturan parameter default untuk penerusan tulis

Grup parameter cluster Aurora menyertakan pengaturan untuk fitur penerusan tulis lokal. Karena ini adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk variabel ini. Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

| Parameter | Cakupan | Tipe | Nilai default | Nilai valid |
|---|---|---|---|---|
| `apg_write_forward.connect_timeout` | Sesi | detik | 30 | 0–2147483647 |
| `apg_write_forward.consistency_mode` | Sesi | enum | Sesi | `SESSION`, `EVENTUAL`, `GLOBAL`, dan `OFF` |
| `apg_write_forward.idle_in_transaction_session_timeout` | Sesi | milidetik | 86400000 | 0–2147483647 |
| `apg_write_forward.idle_session_timeout` | Sesi | milidetik | 300000 | 0–2147483647 |
| `apg_write_forward.max_forwarding_connections_percent` | Global | int | 25 | 1–100 |

Parameter `apg_write_forward.max_forwarding_connections_percent` adalah batas atas pada slot koneksi basis data yang dapat digunakan untuk menangani kueri yang diteruskan dari pembaca. Hal ini dinyatakan sebagai persentase dari `max_connections` pengaturan untuk instance DB penulis. Misalnya, jika `max_connections` adalah `800` dan

`apg_write_forward.max_forwarding_connections_percent` adalah `10`, maka penulis mengizinkan maksimum 80 sesi yang diteruskan secara serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan `max_connections`. Pengaturan ini hanya berlaku pada instance DB penulis ketika cluster mengaktifkan penerusan tulis lokal.

Gunakan pengaturan berikut untuk mengontrol permintaan penerusan tulis lokal:

- `apg_write_forward.consistency_mode`— Parameter tingkat sesi yang mengontrol tingkat konsistensi baca pada replika baca. Nilai yang valid adalah `SESSION`, `EVENTUAL`, `GLOBAL`, atau `OFF`. Secara default, nilainya diatur ke `SESSION`. Menyetel nilai untuk `OFF` menonaktifkan penerusan tulis lokal dalam sesi. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Konsistensi dan isolasi untuk penerusan tulis lokal di Aurora Postgre SQL. Parameter ini hanya relevan dalam instance pembaca yang mengaktifkan penerusan penulisan lokal.
- `apg_write_forward.connect_timeout`— Jumlah maksimum detik replika baca menunggu saat membuat koneksi ke instans DB penulis sebelum menyerah. Nilai `0` berarti menunggu tanpa batas.
- `apg_write_forward.idle_in_transaction_session_timeout`— Jumlah milidetik instans DB penulis menunggu aktivitas pada koneksi yang diteruskan dari replika baca yang memiliki transaksi terbuka sebelum menutupnya. Jika sesi tetap idle dalam transaksi setelah periode ini, Aurora akan mengakhiri sesi. Nilai `0` menonaktifkan batas waktu.
- `apg_write_forward.idle_session_timeout`— Jumlah milidetik instance DB penulis menunggu aktivitas pada koneksi yang diteruskan dari replika baca sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan mengakhiri sesi. Nilai `0` menonaktifkan batas waktu.

## rdswriteforwarduser

`rdswriteforwarduser`Ini adalah pengguna yang akan kita gunakan untuk membuat koneksi antara replika baca dan instance DB penulis.

> **Note**
>
> `rdswriteforwarduser`mewarisi hak istimewa CONNECT ke database pelanggan melalui peran PUBLIK. Jika hak istimewa untuk peran PUBLIK dicabut, Anda harus MEMBERIKAN hak istimewa CONNECT untuk database yang Anda butuhkan untuk meneruskan penulisan.

# Memantau penerusan penulisan lokal di Aurora Postgre SQL

Dengan menggunakan bagian berikut, Anda dapat memantau penerusan penulisan lokal di SQL klaster Aurora Postgre, termasuk CloudWatch metrik yang relevan dan acara tunggu untuk melacak kinerja dan mengidentifikasi potensi masalah.

## CloudWatch Metrik Amazon dan variabel status Aurora SQL Postgre untuk penerusan tulis

CloudWatch Metrik Amazon berikut berlaku untuk instans DB penulis saat Anda menggunakan penerusan tulis pada satu atau beberapa replika baca.

| CloudWatch Metrik | Unit dan deskripsi |
| --- | --- |
| `AuroraLocal ForwardingWriterDML Throughput` | Hitungan (per detik). Jumlah DML pernyataan diteruskan diproses setiap detik oleh instance DB penulis ini. |
| `AuroraLocalForwardingWriter OpenSessions` | Hitungan. Jumlah sesi terbuka pada instans DB penulis ini yang memproses kueri yang diteruskan. |
| `AuroraLocal ForwardingWriterTo talSessions` | Hitungan. Total jumlah sesi yang diteruskan pada instans DB penulis ini. |

CloudWatch Metrik berikut berlaku untuk setiap replika baca. Metrik ini diukur pada setiap instans DB pembaca di cluster DB dengan penerusan tulis lokal diaktifkan.

| CloudWatch Metrik | Unit dan deskripsi |
| --- | --- |
| `AuroraForwardingReplicaCommitThro ughput` | Hitungan (per detik). Jumlah commit dalam sesi yang diteruskan oleh replika ini setiap detik. |
| `AuroraForwardingReplicaDMLL atency` | Milidetik. Rata-rata waktu respons dalam milidetik penerusan DMLs di replica. |

| CloudWatch Metrik | Unit dan deskripsi |
| --- | --- |
| `AuroraForwardingReplicaDMLThroughput` | Hitungan (per detik). Jumlah DML pernyataan diteruskan diproses pada replika ini setiap detik. |
| `AuroraForwardingReplicaErrorSessionsLimit` | Hitungan. Jumlah sesi yang ditolak oleh instans DB penulis karena batas untuk koneksi maks atau koneksi maju tulis maks tercapai. |
| `AuroraForwardingReplicaOpenSessions` | Hitungan. Jumlah sesi yang menggunakan penerusan tulis lokal pada instance replika. |
| `AuroraForwardingReplicaReadWaitLatency` | Milidetik. Waktu tunggu rata-rata dalam milidetik yang ditunggu replika agar konsisten dengan instans LSN DB penulis. Batas tunggu instans DB pembaca ditentukan oleh pengaturan `apg_write_forward.consistency_mode` . Untuk informasi tentang pengaturan ini, lihat the section called “Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL”. |

## Tunggu acara untuk penerusan tulis lokal di Aurora Postgre SQL

Amazon Aurora menghasilkan peristiwa tunggu berikut saat Anda menggunakan penerusan tulis dengan Aurora Postgre. SQL

Topik

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

`IPC:AuroraWriteForwardConnect`Peristiwa terjadi ketika proses backend pada replika baca sedang menunggu koneksi ke instans DB penulis dibuka.

Kemungkinan menyebabkan peningkatan waktu tunggu

Peristiwa ini meningkat seiring dengan meningkatnya jumlah upaya koneksi dari replika baca ke node penulis.

Tindakan

Kurangi jumlah koneksi simultan dari replika baca ke node penulis.

IPC:AuroraWriteForwardConsistencyPoint

`IPC:AuroraWriteForwardConsistencyPoint`Acara ini menjelaskan berapa lama kueri dari node pada replika baca akan menunggu hasil operasi penulisan yang diteruskan direplikasi ke Wilayah saat ini. Peristiwa ini hanya dihasilkan jika parameter tingkat sesi `apg_write_forward.consistency_mode` diatur ke salah satu dari berikut ini:

- `SESSION`— kueri pada replika baca menunggu hasil dari semua perubahan yang dibuat dalam sesi itu.
- `GLOBAL`— kueri pada replika baca menunggu hasil perubahan yang dibuat oleh sesi itu, ditambah semua perubahan yang dilakukan dari instance DB penulis dan replika baca.

Untuk informasi selengkapnya tentang pengaturan parameter `apg_write_forward.consistency_mode`, lihat the section called “Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL”.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum untuk waktu tunggu yang lebih lama termasuk yang berikut:

- Peningkatan lag replika, yang diukur dengan CloudWatch `ReplicaLag` metrik Amazon. Untuk informasi selengkapnya tentang metrik ini, lihat Memantau replikasi Aurora PostgreSQL.
- Peningkatan beban pada instance DB penulis atau replika baca.

Tindakan

Ubah mode konsistensi Anda, tergantung kebutuhan aplikasi Anda.

IPC:AuroraWriteForwardExecute

`IPC:AuroraWriteForwardExecute`Peristiwa terjadi ketika proses backend pada replika baca sedang menunggu kueri yang diteruskan untuk diselesaikan dan mendapatkan hasil dari node penulis cluster DB.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Mengambil sejumlah besar baris dari node penulis.
- Peningkatan latensi jaringan antara node penulis dan replika baca meningkatkan waktu yang dibutuhkan replika baca untuk menerima data dari node penulis.
- Peningkatan beban pada replika baca dapat menunda transmisi permintaan kueri dari replika baca ke node penulis.
- Peningkatan beban pada node penulis dapat menunda transmisi data dari node penulis ke replika baca.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Optimalkan kueri untuk mengambil hanya data yang diperlukan.
- Optimalkan operasi bahasa manipulasi data (DML) untuk hanya memodifikasi data yang diperlukan.
- Jika replika baca atau node penulis dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instance dengan CPU kapasitas lebih besar atau lebih banyak bandwidth jaringan.

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

`IPC:AuroraWriteForwardGetGlobalConsistencyPoint`Peristiwa terjadi ketika proses backend pada replika baca yang menggunakan mode GLOBAL konsistensi menunggu untuk mendapatkan titik konsistensi global dari node penulis sebelum menjalankan kueri.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara replika baca dan node penulis meningkatkan waktu yang dibutuhkan replika baca untuk menerima data dari node penulis.
- Peningkatan beban pada replika baca dapat menunda transmisi permintaan kueri dari replika baca ke node penulis.
- Peningkatan beban pada node penulis dapat menunda transmisi data dari node penulis ke replika baca.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Ubah mode konsistensi Anda, tergantung kebutuhan aplikasi Anda.
- Jika replika baca atau node penulis dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instance dengan CPU kapasitas lebih besar atau lebih banyak bandwidth jaringan.

IPC:AuroraWriteForwardXactAbort

`IPC:AuroraWriteForwardXactAbort`Peristiwa terjadi ketika proses backend pada replika baca sedang menunggu hasil kueri pembersihan jarak jauh. Kueri pembersihan dikeluarkan untuk mengembalikan proses ke keadaan yang sesuai setelah transaksi dengan penerusan tulis dibatalkan. Amazon Aurora melakukannya karena ada kesalahan yang ditemukan atau karena pengguna mengeluarkan perintah `ABORT` eksplisit atau membatalkan kueri yang sedang berjalan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara replika baca dan node penulis meningkatkan waktu yang dibutuhkan replika baca untuk menerima data dari node penulis.
- Peningkatan beban pada replika baca dapat menunda transmisi permintaan kueri pembersihan dari replika baca ke node penulis.
- Peningkatan beban pada node penulis dapat menunda transmisi data dari node penulis ke replika baca.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Selidiki penyebab transaksi yang dibatalkan.
- Jika replika baca atau instans DB penulis dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi jenis instans dengan CPU kapasitas lebih besar atau lebih banyak bandwidth jaringan.

IPC:AuroraWriteForwardXactCommit

`IPC:AuroraWriteForwardXactCommit`Peristiwa ini terjadi ketika proses backend pada replika baca sedang menunggu hasil dari perintah transaksi komit yang diteruskan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara replika baca dan node penulis meningkatkan waktu yang dibutuhkan replika baca untuk menerima data dari node penulis.
- Peningkatan beban pada replika baca dapat menunda transmisi permintaan kueri dari replika baca ke node penulis.
- Peningkatan beban pada node penulis dapat menunda transmisi data dari node penulis ke replika baca.

Tindakan

Jika replika baca atau node penulis dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instance dengan CPU kapasitas lebih besar atau lebih banyak bandwidth jaringan.

IPC:AuroraWriteForwardXactStart

`IPC:AuroraWriteForwardXactStart`Peristiwa ini terjadi ketika proses backend pada replika baca sedang menunggu hasil dari perintah transaksi awal yang diteruskan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara replika baca dan node penulis meningkatkan waktu yang dibutuhkan replika baca untuk menerima data dari node penulis.

- Peningkatan beban pada replika baca dapat menunda transmisi permintaan kueri dari replika baca ke node penulis.
- Peningkatan beban pada node penulis dapat menunda transmisi data dari node penulis ke replika baca.

Tindakan

Jika replika baca atau node penulis dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instance dengan CPU kapasitas lebih besar atau lebih banyak bandwidth jaringan.

# Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock

Anda dapat menggunakan cluster Aurora PostgreSQL DB sebagai Basis Pengetahuan untuk Amazon Bedrock. Untuk informasi selengkapnya, lihat Membuat penyimpanan vektor di Amazon Aurora. Basis Pengetahuan secara otomatis mengambil data teks tidak terstruktur yang disimpan dalam bucket Amazon S3, mengubahnya menjadi potongan teks dan vektor, dan menyimpannya dalam database PostgreSQL. Dengan aplikasi AI generatif, Anda dapat menggunakan Agen untuk Amazon Bedrock untuk menanyakan data yang disimpan di Pangkalan Pengetahuan dan menggunakan hasil kueri tersebut untuk menambah jawaban yang disediakan oleh model dasar. Alur kerja ini disebut Retrieval Augmented Generation (RAG). Untuk informasi lebih lanjut tentang RAG, lihat Retrieval Augmented Generation (RAG).

Untuk informasi rinci tentang penggunaan Aurora PostgreSQL untuk membangun aplikasi AI generatif menggunakan RAG, lihat posting blog ini.

Topik

- Prasyarat
- Mempersiapkan Aurora PostgreSQL untuk digunakan sebagai Basis Pengetahuan untuk Amazon Bedrock
- Membuat Basis Pengetahuan di konsol Bedrock
- Cepat membuat SQL Basis Pengetahuan Aurora Postgre untuk Amazon Bedrock

## Prasyarat

Biasakan diri Anda dengan prasyarat berikut untuk menggunakan klaster Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock. Pada tingkat tinggi, Anda perlu mengkonfigurasi layanan berikut untuk digunakan dengan Bedrock:

- Amazon Aurora PostgreSQL DB cluster dibuat dalam salah satu versi berikut:
  - 16.1 dan semua versi yang lebih tinggi
  - 15.4 dan versi yang lebih tinggi
  - 14.9 dan versi yang lebih tinggi
  - 13.12 dan versi yang lebih tinggi
  - 12.16 dan versi yang lebih tinggi

  > **Note**
  >
  > Anda harus mengaktifkan `pgvector` ekstensi di database target Anda dan menggunakan versi 0.5.0 atau lebih tinggi. Untuk informasi lebih lanjut, lihat pgvector v0.5.0 dengan pengindeksan HNSW.

- API Data RDS
- Seorang pengguna dikelola di AWS Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.

## Mempersiapkan Aurora PostgreSQL untuk digunakan sebagai Basis Pengetahuan untuk Amazon Bedrock

Ikuti langkah-langkah yang dijelaskan di bagian di bawah ini untuk mempersiapkan Aurora PostgreSQL untuk digunakan sebagai Basis Pengetahuan untuk Amazon Bedrock.

### Membuat dan mengkonfigurasi Aurora PostgreSQL

Untuk mengonfigurasi Amazon Bedrock dengan cluster DB PostgreSQL Aurora, Anda harus terlebih dahulu membuat klaster DB PostgreSQL Aurora dan mencatat bidang penting untuk mengonfigurasinya dengan Amazon Bedrock. Untuk informasi selengkapnya tentang membuat klaster Aurora PostgreSQL DB, lihat. Membuat dan terhubung ke klaster DB Aurora PostgreSQL

- Aktifkan API Data saat membuat cluster Aurora PostgreSQL DB. Untuk informasi selengkapnya tentang versi yang didukung, lihatMenggunakan Amazon RDS Data API.
- Pastikan untuk mencatat Nama Sumber Daya Amazon (ARN) dari cluster Aurora PostgreSQL DB Anda. Anda akan memerlukannya untuk mengonfigurasi cluster DB untuk digunakan dengan Amazon Bedrock. Untuk informasi selengkapnya, lihat Nama Sumber Daya Amazon (ARNs).

## Menghubungkan ke database dan menginstal pgvector

Anda dapat terhubung ke Aurora PostgreSQL menggunakan salah satu utilitas koneksi. Untuk informasi lebih rinci tentang utilitas ini, lihatMenghubungkan ke klaster DB Amazon Aurora PostgreSQL.. Atau, Anda dapat menggunakan editor kueri konsol RDS untuk menjalankan kueri. Anda memerlukan cluster Aurora DB dengan API Data RDS diaktifkan untuk menggunakan editor kueri.

1. Masuk ke database dengan pengguna master Anda dan atur pgvector. Gunakan perintah berikut jika ekstensi tidak diinstal:

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   Gunakan `pgvector` 0.5.0 dan versi yang lebih tinggi yang mendukung pengindeksan HNSW. Untuk informasi lebih lanjut, lihat pgvector v0.5.0 dengan pengindeksan HNSW.
2. Gunakan perintah berikut untuk memeriksa versi yang `pg_vector` diinstal:

   ```
   SELECT extversion FROM pg_extension WHERE extname='vector';
   ```

## Menyiapkan objek database dan hak istimewa

1. Buat skema tertentu yang dapat digunakan Bedrock untuk menanyakan data. Gunakan perintah berikut untuk membuat skema:

   ```
   CREATE SCHEMA bedrock_integration;
   ```
2. Buat peran baru yang dapat digunakan Bedrock untuk menanyakan database. Gunakan perintah berikut untuk membuat peran baru:

   ```
   CREATE ROLE bedrock_user WITH PASSWORD 'password' LOGIN;
   ```

> **Note**
>
> Catat kata sandi ini karena Anda akan memerlukannya nanti untuk membuat kata sandi Secrets Manager.

Jika Anda menggunakan `psql` klien, maka gunakan perintah berikut untuk membuat peran baru:

```
CREATE ROLE bedrock_user LOGIN;
\PASSWORD password;
```

3. Berikan `bedrock_user` izin untuk mengelola `bedrock_integration` skema. Ini akan memberikan kemampuan untuk membuat tabel atau indeks dalam skema.

```
GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
```

4. Login sebagai `bedrock_user` dan membuat tabel di`bedrock_integration schema`.

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(n), chunks text, metadata json, custom_metadata jsonb);
```

Perintah ini akan membuat `bedrock_kb` tabel dalam `bedrock_integration` skema dengan embeddings Titan.

Ganti n dalam tipe `vector(n)` data dengan dimensi yang sesuai untuk model embedding yang Anda gunakan. Gunakan rekomendasi di bawah ini untuk membantu memilih dimensi Anda:

- Untuk model Titan v2, gunakan`vector(1024)`, atau`vector(512)`, atau`vector (256)`. Untuk mempelajari lebih lanjut, lihat Amazon Titan Embeddings Text.
- Untuk model Titan v1.2, gunakan. `vector(1536)` Untuk mempelajari lebih lanjut, lihat Amazon Titan Multimodal Embeddings G1.
- Untuk model Cohere Embed, gunakan. `vector(1024)` Untuk mempelajari lebih lanjut, lihat Cohere Embed models.
- Untuk Cohere Embed Multilingual v3, gunakan. `vector(1024)`

Empat kolom pertama adalah wajib. Untuk penanganan metadata, Bedrock menulis data dari file metadata Anda ke kolom. `custom_metadata` Sebaiknya buat kolom ini jika Anda berencana

menggunakan metadata dan pemfilteran. Jika Anda tidak membuat `custom_metadata` kolom, tambahkan kolom individual untuk setiap atribut metadata dalam tabel Anda sebelum Anda mulai menelan. Untuk informasi selengkapnya, lihat Mengonfigurasi dan menyesuaikan kueri serta pembuatan respons.

5. Ikuti langkah-langkah berikut untuk membuat indeks yang diperlukan yang digunakan Bedrock untuk menanyakan data Anda:

   - Buat indeks dengan operator kosinus yang dapat digunakan batuan dasar untuk menanyakan data.

     ```
     CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
      vector_cosine_ops);
     ```

   - Kami menyarankan Anda untuk menetapkan nilai 256 `ef_construction` untuk `pgvector` 0.6.0 dan versi yang lebih tinggi yang menggunakan bangunan indeks paralel.

     ```
     CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
      vector_cosine_ops) WITH (ef_construction=256);
     ```

   - Buat indeks yang dapat digunakan Bedrock untuk menanyakan data teks.

     ```
     CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (to_tsvector('simple',
      chunks));
     ```

   - Jika Anda membuat kolom untuk metadata kustom, buat indeks yang dapat digunakan Bedrock untuk menanyakan metadata.

     ```
     CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (custom_metadata);
     ```

## Buat rahasia di Secrets Manager

Secrets Manager memungkinkan Anda menyimpan kredensil Aurora Anda sehingga mereka dapat dikirim dengan aman ke aplikasi. Jika Anda tidak memilih opsi manajer AWS rahasia saat membuat cluster Aurora PostgreSQL DB, Anda dapat membuat rahasia sekarang. Untuk informasi selengkapnya tentang membuat rahasia AWS Secrets Manager database, lihat AWS Secrets Manager rahasia database.

# Membuat Basis Pengetahuan di konsol Bedrock

Saat menyiapkan Aurora PostgreSQL untuk digunakan sebagai penyimpanan vektor untuk Pangkalan Pengetahuan, Anda harus mengumpulkan detail berikut yang perlu Anda berikan ke konsol Amazon Bedrock.

- Amazon Aurora DB cluster ARN — ARN cluster DB Anda.
- Rahasia ARN — ARN kunci Secrets Manager untuk AWS cluster DB Anda.
- Nama Database — Nama database Anda. Misalnya, Anda dapat menggunakan database default*`postgres`*.
- Nama tabel - Kami menyarankan Anda untuk memberikan nama skema yang memenuhi syarat saat membuat tabel menggunakan perintah yang mirip dengan berikut ini:

  ```
  CREATE TABLE bedrock_integration.bedrock_kb;
  ```

  Perintah ini akan membuat `bedrock_kb` tabel dalam `bedrock_integration` skema.
- Saat membuat tabel, pastikan untuk mengkonfigurasinya dengan kolom dan tipe data yang ditentukan. Anda dapat menggunakan nama kolom pilihan Anda alih-alih yang tercantum dalam tabel. Ingatlah untuk mencatat nama-nama yang Anda pilih untuk referensi selama pengaturan Pangkalan Pengetahuan.

| Nama kolom | Jenis data | Deskripsi |
|---|---|---|
| id | Kunci utama UUID | Berisi pengidentifikasi unik untuk setiap catatan. |
| potongan | Teks | Berisi potongan teks mentah dari sumber data Anda. |
| menyematkan | vektor | Berisi embeddings vektor dari sumber data. |
| Metadata | JSON | Berisi metadata yang diperlukan untuk melakukan atribusi sumber dan untuk mengaktifkan konsumsi dan kueri data. |

| Nama kolom | Jenis data | Deskripsi |
| --- | --- | --- |
| custom_metadata | JSONB | (Opsional) Mendefinisikan kolom target tempat Amazon Bedrock menulis detail metadata dari sumber data Anda. |

Dengan detail ini, Anda sekarang dapat membuat Basis Pengetahuan di konsol Bedrock. Untuk informasi lebih rinci tentang menyiapkan indeks vektor dan membuat informasi Pangkalan Pengetahuan, lihat Membuat penyimpanan vektor di Amazon Aurora dan Membuat penyimpanan vektor di Amazon Aurora.

Setelah menambahkan Aurora sebagai Basis Pengetahuan Anda, Anda sekarang dapat menelan sumber data Anda untuk pencarian dan kueri. Untuk informasi selengkapnya, lihat Menyerap sumber data Anda ke dalam Pangkalan Pengetahuan.

## Cepat membuat SQL Basis Pengetahuan Aurora Postgre untuk Amazon Bedrock

Alur kerja pengambilan augmented generation (RAG) Amazon Bedrock bergantung pada data vektor yang disimpan dalam database Aurora Postgre untuk mendukung pengambilan konten. SQL Sebelumnya, menyiapkan Aurora Postgre SQL sebagai penyimpanan data vektor untuk Pangkalan Pengetahuan Batuan Dasar adalah proses multi-langkah, membutuhkan banyak tindakan manual di berbagai antarmuka pengguna. Hal ini membuat sulit bagi ilmuwan data dan pengembang untuk memanfaatkan Aurora untuk proyek Bedrock mereka.

Untuk meningkatkan pengalaman pengguna, AWS telah membuat opsi pembuatan cepat CloudFormation berbasis baru yang menyederhanakan proses penyiapan. Dengan Aurora quick create, Anda sekarang dapat menyediakan cluster Aurora SQL Postgre DB yang telah dikonfigurasi sebelumnya sebagai penyimpanan vektor untuk Pangkalan Pengetahuan Amazon Bedrock Anda dengan satu klik.

Topik

- Wilayah yang didukung dan versi Aurora Postgre SQL
- Memahami proses pembuatan cepat
- Manfaat menggunakan Aurora quick create

- Keterbatasan proses pembuatan cepat Aurora

## Wilayah yang didukung dan versi Aurora Postgre SQL

Opsi pembuatan cepat Aurora tersedia di semua AWS wilayah yang mendukung Pangkalan Pengetahuan Amazon Bedrock. Secara default, ini menciptakan cluster Aurora Postgre SQL DB dengan versi 15.7. Untuk informasi selengkapnya tentang Wilayah yang didukung, lihat Model dan wilayah yang didukung untuk Pangkalan Pengetahuan Amazon Bedrock.

## Memahami proses pembuatan cepat

Proses pembuatan cepat secara otomatis menyediakan sumber daya berikut untuk menyiapkan database Amazon Aurora Postgre sebagai penyimpanan data vektor untuk SQL Pangkalan Pengetahuan Batuan Dasar Anda:

Cluster Aurora Postgre SQL DB di akun Anda, dikonfigurasi dengan pengaturan default.

- ACUs(Unit Kapasitas Aurora) diatur dari 0 hingga 16. Ini memungkinkan penyimpanan vektor Anda turun menjadi nol saat tidak digunakan, menghemat biaya komputasi. ACUsDapat disesuaikan nanti di RDS konsol Amazon.
- (Hierarchical Navigable Small World) HNSW indeks menggunakan jarak Euclidean sebagai ukuran kesamaan untuk embeddings vektor Bedrock yang disimpan di Aurora.
- Instans DB adalah instance v2 tanpa server.
- Cluster dikaitkan dengan default VPC dan subnet, dan memiliki RDS Data API diaktifkan.
- Kredensi admin cluster dikelola oleh AWS Secrets Manager.

Selain pengaturan default, pengaturan berikut diatur untuk Anda. Saat Anda menjalani proses, Anda akan melihat layar yang menjelaskan alur kerja.

- Menyemai cluster Aurora dengan objek database yang diperlukan:
  - Buat ekstensi pgvector, skema, peran, dan tabel yang diperlukan untuk Basis Pengetahuan Batuan Dasar.
  - Daftarkan pengguna database hak istimewa terbatas untuk Bedrock untuk berinteraksi dengan cluster.
- Spanduk kemajuan akan ditampilkan di seluruh proses penyediaan sumber daya, memungkinkan Anda melacak status sub-peristiwa berikut:

  - Pembuatan cluster Aurora
  - Pembibitan cluster Aurora
  - Penciptaan Basis Pengetahuan

  Spanduk tetap terlihat sampai basis pengetahuan sepenuhnya dibuat, bahkan jika Anda menavigasi jauh dari halaman dan kembali.
- Anda dapat `View details` mengklik spanduk kemajuan untuk melihat status setiap langkah. Untuk informasi selengkapnya tentang peristiwa selama pembuatan basis pengetahuan, pilih CloudFormation tautan di layar detail tampilan. Setelah proses selesai, Basis Pengetahuan Batuan Dasar baru Anda akan siap digunakan.
- Tumpukan IDs untuk semua sumber daya pembuatan cepat dapat ditemukan di tag Basis Pengetahuan Batuan Dasar, jika Anda perlu mereferensikannya.

Basis Pengetahuan Batuan Dasar, dengan konfigurasi ke cluster Aurora yang baru disediakan saat penyimpanan vektor dibuat.

## Manfaat menggunakan Aurora quick create

- Proses pembuatan cepat CloudFormation berbasis secara signifikan mengurangi waktu dan kompleksitas yang diperlukan untuk menggunakan Aurora sebagai penyimpanan vektor.
- Aurora menawarkan kinerja yang sangat baik, skalabilitas vektor, dan manfaat biaya dengan kemampuan untuk menskalakan hingga nol biaya komputasi saat tidak digunakan.
- Proses pembuatan cepat merampingkan end-to-end pengalaman, memungkinkan Anda untuk dengan mudah membuat dan mengonfigurasi Pangkalan Pengetahuan Batuan Dasar Anda menggunakan Aurora.
- Pelanggan dapat membangun berdasarkan CloudFormation template untuk menyesuaikan penyediaan dengan konfigurasi mereka sendiri.

## Keterbatasan proses pembuatan cepat Aurora

- Dengan opsi Aurora quick create, cluster DB disediakan dengan konfigurasi default. Namun, pengaturan default ini mungkin tidak memenuhi persyaratan khusus Anda atau kasus penggunaan yang dimaksudkan. Pembuatan cepat tidak menawarkan opsi untuk memodifikasi konfigurasi selama proses penyediaan. Konfigurasi diatur secara otomatis untuk merampingkan pengalaman

penerapan. Jika Anda perlu menyesuaikan konfigurasi cluster Aurora DB, Anda dapat melakukannya setelah penerapan awal dengan membuat cepat di konsol Amazon. RDS

- Sementara alur pembuatan cepat menyederhanakan proses penyiapan, waktu untuk membuat cluster Aurora DB masih sekitar 10 menit, sama dengan penerapan manual. Ini karena waktu yang diperlukan untuk menyediakan infrastruktur Aurora.
- Opsi pembuatan cepat dirancang untuk eksperimen dan pengaturan cepat. Sumber daya yang dibuat melalui pembuatan cepat mungkin tidak cocok untuk penggunaan produksi, dan Anda tidak akan dapat langsung memigrasikannya ke lingkungan produksi di lingkungan AndaVPC.

# Mengintegrasikan Amazon Aurora SQL Postgre dengan layanan lain AWS

Amazon Aurora terintegrasi dengan AWS layanan lain sehingga Anda dapat memperluas cluster Aurora Postgre SQL DB Anda untuk menggunakan kemampuan tambahan di Cloud. AWS Cluster Aurora Postgre SQL DB Anda dapat menggunakan AWS layanan untuk melakukan hal berikut:

- Kumpulkan, lihat, dan nilai performa instans DB Aurora Postgre dengan cepat dengan Amazon Performance SQL Insights. RDS Performance Insights memperluas fitur RDS pemantauan Amazon yang ada untuk menggambarkan kinerja database Anda dan membantu Anda menganalisis masalah apa pun yang mempengaruhinya. Dengan dasbor Performance Insights, Anda dapat memvisualisasikan pemuatan database dan memfilter beban berdasarkan menunggu, SQL pernyataan, host, atau pengguna. Untuk informasi selengkapnya tentang Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Konfigurasikan cluster SQL DB Aurora Postgre Anda untuk mempublikasikan data log ke Amazon Logs. CloudWatch CloudWatch Log menyediakan penyimpanan yang sangat tahan lama untuk catatan log Anda. Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Untuk informasi selengkapnya, lihat Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch .
- Impor data dari bucket Amazon S3 ke cluster DB Aurora Postgre, atau ekspor data dari cluster SQL DB Aurora Postgre ke bucket Amazon S3. SQL Untuk informasi selengkapnya, silakan lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL dan Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.
- Tambahkan prediksi berbasis pembelajaran mesin ke aplikasi database menggunakan bahasa. SQL Pembelajaran mesin Aurora menggunakan integrasi yang sangat optimal antara database Aurora dan layanan pembelajaran AWS mesin (ML) AI dan SageMaker Amazon Comprehend.

Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora Postgre SQL.

- Memanggil AWS Lambda fungsi dari cluster Aurora SQL Postgre DB. Untuk melakukan ini, gunakan SQL ekstensi `aws_lambda` Postgre yang disediakan dengan Aurora Postgre. SQL Untuk informasi selengkapnya, lihat .
- Integrasikan kueri dari Amazon Redshift dan Aurora Postgre. SQL Untuk informasi selengkapnya, lihat Memulai menggunakan kueri federasi ke Postgre di Panduan Pengembang SQL Database Amazon Redshift.

## Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL

Anda dapat mengimpor data yang telah disimpan menggunakan Amazon Simple Storage Service ke dalam tabel pada instans klaster DB Aurora PostgreSQL. Untuk melakukannya, instal ekstensi Aurora PostgreSQL `aws_s3` terlebih dahulu. Ekstensi ini menyediakan fungsi yang Anda gunakan untuk mengimpor data dari bucket Amazon S3. Bucket adalah kontainer Amazon S3 untuk objek dan file. Data dapat berada dalam file nilai yang dipisahkan koma (CSV), file teks, atau file terkompresi (gzip). Berikut ini, Anda dapat mempelajari cara menginstal ekstensi dan cara mengimpor data dari Amazon S3 ke dalam tabel.

Basis data Anda harus menjalankan PostgreSQL versi 10.7 atau yang lebih tinggi untuk mengimpor dari Amazon S3 ke Aurora PostgreSQL.

Jika tidak memiliki data yang tersimpan di Amazon S3, Anda harus terlebih dahulu membuat bucket dan menyimpan data. Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna Amazon Simple Storage Service.

- Membuat bucket
- Menambahkan objek ke bucket

Impor lintas akun dari Amazon S3 didukung. Untuk informasi selengkapnya, lihat Memberikan izin lintas akun di Panduan Pengguna Amazon Simple Storage Service User Guide.

Anda dapat menggunakan kunci yang dikelola pelanggan untuk enkripsi saat mengimpor data dari S3. Untuk informasi selengkapnya, lihat kunci KMS yang disimpan di AWS KMS dalam Panduan Pengguna Amazon Simple Storage Service.

> **Note**
>
> Pengimporan data dari Amazon S3 tidak didukung untuk Aurora Serverless v1. Fitur ini didukung untuk Aurora Serverless v2.

Topik


- Menginstal ekstensi aws_s3
- Ikhtisar impor data dari data Amazon S3
- Menyiapkan akses ke bucket Amazon S3
- Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL
- Referensi fungsi


## Menginstal ekstensi aws_s3

Sebelum Anda dapat menggunakan Amazon S3 dengan cluster Aurora Postgre SQL DB Anda, , Anda perlu menginstal ekstensi. `aws_s3` Ekstensi ini menyediakan fungsi untuk mengimpor data dari Amazon S3. Ini juga menyediakan fungsi untuk mengekspor data dari instance cluster Aurora SQL Postgre DB dan untuk RDS instance Postgre DB . Untuk informasi selengkapnya, lihat Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3. Ekstensi `aws_s3` bergantung pada beberapa fungsi pembantu dalam ekstensi `aws_commons`, yang diinstal secara otomatis bila diperlukan.

Untuk menginstal ekstensi **`aws_s3`**

1. Gunakan psql (ataupgAdmin) untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance Postgre DB istimewa. `rds_superuser` Jika Anda menyimpan nama default selama proses penyiapan, Anda terhubung sebagai `postgres`.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Untuk menginstal ekstensi, jalankan perintah berikut.

   ```
   postgres=> CREATE EXTENSION aws_s3 CASCADE;
   NOTICE: installing required extension "aws_commons"
   CREATE EXTENSION
   ```

3. Untuk memverifikasi bahwa ekstensi sudah diinstal, Anda dapat menggunakan metacommand psql `\dx`.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                Description
-------------+---------+------------+-------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

Fungsi untuk mengimpor data dari Amazon S3 dan mengekspor data ke Amazon S3 kini dapat digunakan.

## Ikhtisar impor data dari data Amazon S3

Untuk mengimpor data S3 ke Aurora PostgreSQL

Pertama, kumpulkan detail yang perlu Anda suplai ke fungsi tersebut. Ini termasuk nama tabel pada instance klaster Aurora PostgreSQL DB Anda, RDS untuk instans DB PostgreSQL Anda, dan tempat penyimpanan data Amazon S3. Wilayah AWS Untuk informasi selengkapnya, buka Melihat objek di Panduan Pengguna Amazon Simple Storage Service.

> **Note**
>
> Impor data multi bagian dari Amazon S3 saat ini tidak didukung.

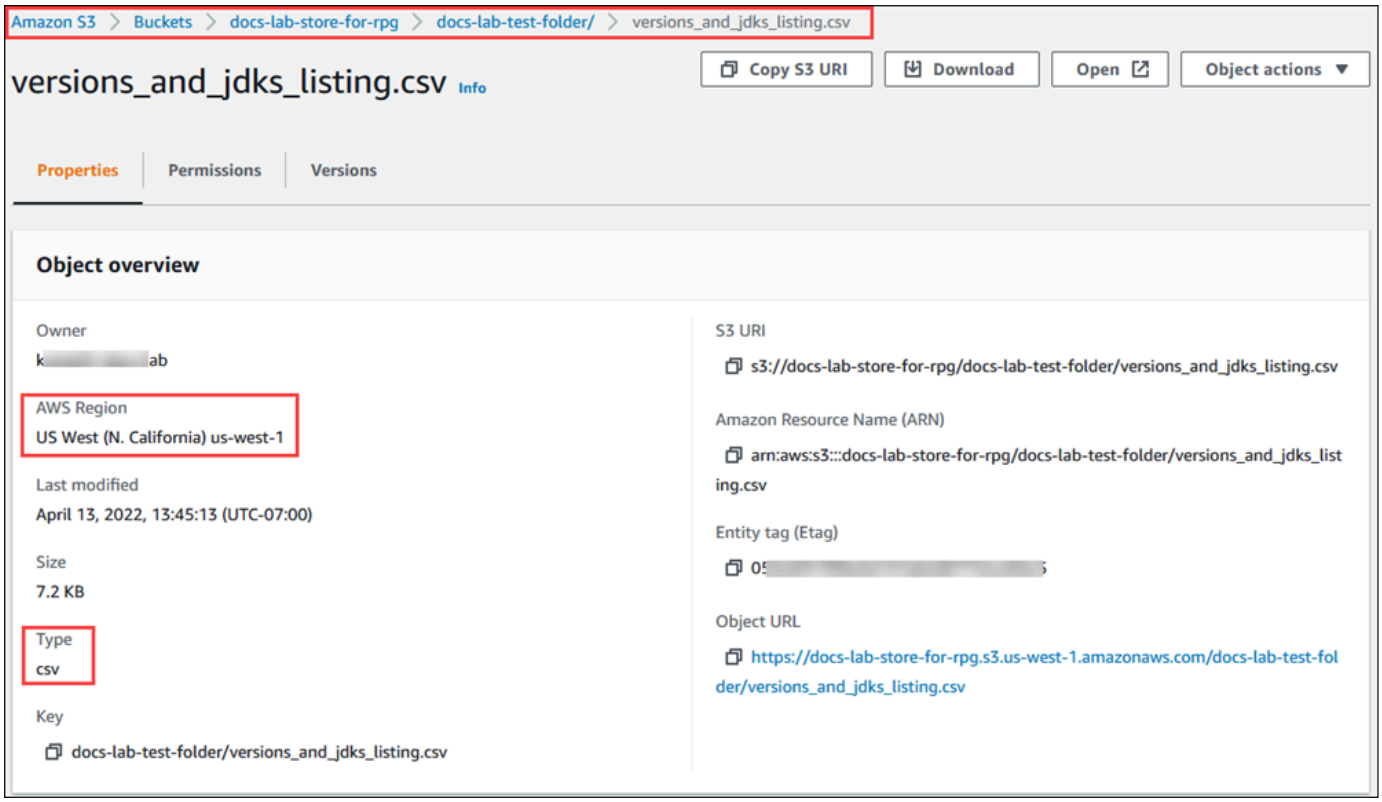
1. Dapatkan nama tabel di mana fungsi `aws_s3.table_import_from_s3` adalah untuk mengimpor data. Sebagai contoh, perintah berikut membuat tabel `t1` yang dapat digunakan di langkah selanjutnya.

```
postgres=> CREATE TABLE t1
    (col1 varchar(80),
    col2 varchar(80),
    col3 varchar(80));
```

2. Dapatkan detail tentang bucket Amazon S3 dan data yang akan diimpor. Untuk melakukan ini, buka konsol Amazon S3 di https://console.aws.amazon.com/s3/, dan pilih Bucket. Temukan

bucket yang berisi data Anda dalam daftar. Pilih bucket, buka halaman ikhtisar Object, lalu pilih Properti.

Catat nama bucket, path Wilayah AWS, dan jenis file. Anda memerlukan Amazon Resource Name (ARN) nanti, untuk menyiapkan akses ke Amazon S3 melalui peran IAM. Untuk informasi selengkapnya, lihat Menyiapkan akses ke bucket Amazon S3. Bagian berikut menunjukkan satu contoh.




3. Anda dapat memverifikasi jalur ke data di bucket Amazon S3 dengan menggunakan perintah. AWS CLI `aws s3 cp` Jika informasinya benar, perintah ini akan mengunduh salinan file Amazon S3.

```
aws s3 cp s3://amzn-s3-demo-bucket/sample_file_path ./
```

4. Siapkan izin di klaster DB Aurora PostgreSQL untuk mengizinkan akses ke file di bucket Amazon S3. Untuk melakukannya, Anda menggunakan peran AWS Identity and Access Management (IAM) atau kredensil keamanan. Untuk informasi selengkapnya, lihat Menyiapkan akses ke bucket Amazon S3.
5. Berikan jalur dan detail objek Amazon S3 lainnya yang dikumpulkan (lihat langkah 2) ke fungsi `create_s3_uri` untuk membuat konsep objek URI Amazon S3. Untuk mempelajari

selengkapnya tentang fungsi ini, lihat aws_commons.create_s3_uri. Berikut ini adalah contoh pembuatan konsep objek ini selama sesi psql.

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

Pada langkah berikutnya, Anda meneruskan objek ini (`aws_commons._s3_uri_1`) ke fungsi `aws_s3.table_import_from_s3` untuk mengimpor data ke tabel.

6. Invokasi fungsi `aws_s3.table_import_from_s3` untuk mengimpor data dari Amazon S3 ke dalam tabel Anda. Untuk informasi referensi, lihat aws_s3.table_import_from_s3. Sebagai contoh, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL.

## Menyiapkan akses ke bucket Amazon S3

Untuk mengimpor data dari file Amazon S3, beri izin klaster DB Aurora PostgreSQL untuk mengakses bucket Amazon S3 yang berisi file tersebut. Berikan akses ke bucket Amazon S3 dengan satu dari dua cara, seperti yang dijelaskan dalam topik berikut.

Topik

- Menggunakan peran IAM untuk mengakses bucket Amazon S3
- Menggunakan kredensial rahasia untuk mengakses bucket Amazon S3
- Memecahkan masalah akses ke Amazon S3

### Menggunakan peran IAM untuk mengakses bucket Amazon S3

Sebelum Anda memuat data dari file Amazon S3, beri izin klaster DB Aurora PostgreSQL untuk mengakses bucket Amazon S3 tempat file berada. Dengan cara ini, Anda tidak perlu mengelola informasi kredensial tambahan atau menyediakannya di panggilan fungsi aws_s3.table_import_from_s3.

Untuk melakukannya, buat kebijakan IAM yang memberikan akses ke bucket Amazon S3. Buat peran IAM lampirkan kebijakan ke peran tersebut. Kemudian, tetapkan peran IAM ke klaster DB Anda.

> **Note**
>
> Anda tidak dapat mengaitkan peran IAM dengan klaster DB Aurora Serverless v1, sehingga langkah berikut tidak berlaku.

Untuk memberikan klaster DB Aurora PostgreSQL akses ke Amazon S3 melalui peran IAM

1. Buat kebijakan IAM.

   Kebijakan ini memberi bucket dan objek izin yang memungkinkan klaster DB PostgreSQL Anda mengakses Amazon S3.

   Sertakan tindakan yang diperlukan berikut dalam kebijakan untuk mengizinkan transfer file dari bucket Amazon S3 ke Aurora PostgreSQL:

   - `s3:GetObject`
   - `s3:ListBucket`

   Sertakan sumber daya berikut dalam kebijakan untuk mengidentifikasi bucket dan objek Amazon S3 dalam bucket. Ini menunjukkan format Amazon Resource Name (ARN) untuk mengakses Amazon S3.

   - arn:aws:s3::: *`amzn-s3-demo-bucket`*
   - arn:aws:s3::: /* *`amzn-s3-demo-bucket`*

   Untuk informasi selengkapnya tentang cara membuat kebijakan IAM untuk Aurora PostgreSQL, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda di Panduan Pengguna IAM.

   AWS CLI Perintah berikut membuat kebijakan IAM bernama `rds-s3-import-policy` dengan opsi ini. Perintah ini akan memberikan akses ke bucket bernama *`amzn-s3-demo-bucket`*.

> **Note**
>
> Catat Amazon Resource Name (ARN) dari kebijakan yang ditampilkan oleh perintah ini. Anda memerlukan ARN di langkah berikutnya saat Anda melampirkan kebijakan ke peran IAM.

Example

Untuk Linux, macOS, atau Unix:

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

Untuk Windows:

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
```

```
        "Action": [
          "s3:GetObject",
          "s3:ListBucket"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    ]
  }'
```

2. Buat peran IAM.

   Anda melakukan ini sehingga Aurora PostgreSQL dapat mengambil peran IAM ini untuk mengakses bucket Amazon S3 Anda. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke pengguna IAM dalam Panduan Pengguna IAM.

   Sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan berbasis sumber daya untuk membatasi izin layanan ke sumber daya tertentu. Ini adalah cara paling efektif untuk melindungi dari masalah deputi yang membingungkan.

   Jika Anda menggunakan kunci konteks kondisi global dan nilai `aws:SourceArn` berisi ID akun, nilai `aws:SourceAccount` dan akun dalam nilai `aws:SourceArn` harus menggunakan ID akun yang sama saat digunakan dalam pernyataan kebijakan yang sama.

   - Gunakan `aws:SourceArn` jika Anda menginginkan akses lintas layanan untuk satu sumber daya.
   - Gunakan `aws:SourceAccount` jika Anda ingin mengizinkan sumber daya apa pun di akun tersebut dikaitkan dengan penggunaan lintas layanan.

   Dalam kebijakan, pastikan untuk menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN penuh sumber daya. Contoh berikut menunjukkan bagaimana melakukannya dengan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-import-role`.

   Example

   Untuk Linux, macOS, atau Unix:

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
   }'
```

Untuk Windows:

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
```

```
      }
    ]
  }'
```

3. Lampirkan kebijakan IAM yang Anda buat ke peran IAM yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat pada langkah sebelumnya ke peran bernama `rds-s3-import-role` Ganti *`your-policy-arn`* dengan ARN kebijakan yang Anda catat di langkah sebelumnya.

   Example

   Untuk Linux, macOS, atau Unix:

   ```
   aws iam attach-role-policy \
      --policy-arn your-policy-arn \
      --role-name rds-s3-import-role
   ```

   Untuk Windows:

   ```
   aws iam attach-role-policy ^
      --policy-arn your-policy-arn ^
      --role-name rds-s3-import-role
   ```

4. Tambahkan peran IAM ke klaster DB.

   Anda melakukannya dengan menggunakan AWS Management Console atau AWS CLI, seperti yang dijelaskan berikut.

Konsol

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih nama klaster DB PostgreSQL untuk menampilkan detailnya.
3. Di tab Konektivitas & keamanan, di bagian Kelola peran IAM, pilih peran yang akan ditambahkan pada bagian Tambahkan peran IAM ke klaster  ini.
4. Di bagian Fitur, pilih s3Import.

5. Pilih Tambahkan peran.

AWS CLI

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan CLI

- Gunakan perintah berikut untuk menambahkan peran ke klaster DB PostgreSQL bernama `my-db-cluster`. Ganti *`your-role-arn`* dengan ARN peran yang Anda catat pada langkah sebelumnya. Gunakan `s3Import` untuk nilai opsi `--feature-name`.

  Example

  Untuk Linux, macOS, atau Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Import \
   --role-arn your-role-arn   \
   --region your-region
```

  Untuk Windows:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Import ^
   --role-arn your-role-arn ^
   --region your-region
```

API RDS

Untuk menambahkan peran IAM untuk cluster PostgreSQL DB menggunakan Amazon RDS API, panggil operasi. AddRoleToDBCluster

Menggunakan kredensial rahasia untuk mengakses bucket Amazon S3

Jika ingin, Anda dapat menggunakan kredensial rahasia untuk memberikan akses ke bucket Amazon S3, alih-alih memberikan akses ke peran IAM. Anda melakukannya dengan menentukan parameter `credentials` dalam panggilan fungsi aws_s3.table_import_from_s3.

`credentials`Parameternya adalah struktur tipe`aws_commons._aws_credentials_1`, yang berisi AWS kredensyal. Gunakan fungsi aws_commons.create_aws_credentials untuk mengatur kunci akses dan kunci rahasia dalam struktur `aws_commons._aws_credentials_1`, seperti yang ditunjukkan berikut ini.

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

Setelah membuat struktur `aws_commons._aws_credentials_1` , gunakan fungsi aws_s3.table_import_from_s3 yang dengan parameter `credentials` untuk mengimpor data, seperti yang ditunjukkan berikut.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

Atau Anda dapat menyertakan panggilan fungsi aws_commons.create_aws_credentials sebaris dalam panggilan fungsi `aws_s3.table_import_from_s3`.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Memecahkan masalah akses ke Amazon S3

Jika Anda mengalami masalah koneksi saat mencoba mengimpor data dari Amazon S3, lihat rekomendasi berikut:

- Memecahkan masalah identitas dan akses Amazon Aurora
- Memecahkan Masalah Amazon S3 di Panduan Pengguna Amazon Simple Storage Service
- Memecahkan Masalah Amazon S3 dan IAM di Panduan Pengguna IAM

## Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL

Anda mengimpor data dari bucket Amazon S3 dengan menggunakan fungsi `table_import_from_s3` aws_s3. Untuk informasi referensi, lihat aws_s3.table_import_from_s3.

> **Note**
>
> Contoh berikut menggunakan metode peran IAM untuk mengizinkan akses ke bucket Amazon S3. Dengan demikian, panggilan fungsi `aws_s3.table_import_from_s3` tidak termasuk parameter kredensial.

Berikut ini menunjukkan contoh khas.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

Parameternya sebagai berikut:

- `t1` – Nama untuk tabel dalam klaster DB PostgreSQL tempat tujuan penyalinan data.
- `''` – Daftar opsional kolom dalam tabel basis data. Anda dapat menggunakan parameter ini untuk menunjukkan kolom data S3 mana yang masuk ke kolom tabel mana. Jika tidak ada kolom yang ditentukan, semua kolom akan disalin ke tabel. Untuk contoh cara menggunakan daftar kolom, lihat Mengimpor file Amazon S3 yang menggunakan pemisah kustom.
- `(format csv)` – Argumen PostgreSQL COPY. Proses penyalinan menggunakan argumen dan format perintah PostgreSQL COPY untuk mengimpor data. Pilihan untuk format mencakup nilai yang dipisahkan koma (CSV) seperti yang ditunjukkan dalam contoh ini, teks, dan biner. Defaultnya adalah teks.
- `s3_uri` – Struktur yang berisi informasi yang mengidentifikasi file Amazon S3. Untuk contoh penggunaan fungsi aws_commons.create_s3_uri untuk membuat struktur `s3_uri`, lihat Ikhtisar impor data dari data Amazon S3.

Untuk informasi selengkapnya tentang fungsi ini, lihat aws_s3.table_import_from_s3.

Fungsi `aws_s3.table_import_from_s3` menampilkan teks. Untuk menentukan jenis file lain yang akan diimpor dari bucket Amazon S3, lihat salah satu contoh berikut.

> **Note**
>
> Mengimpor file 0 byte akan menyebabkan kesalahan.

Topik

- Mengimpor file Amazon S3 yang menggunakan pemisah kustom
- Mengimpor file terkompresi (gzip) Amazon S3
- Mengimpor file Amazon S3 yang dienkode

### Mengimpor file Amazon S3 yang menggunakan pemisah kustom

Contoh berikut menunjukkan cara mengimpor file yang menggunakan pemisah kustom. Ini juga menunjukkan cara mengontrol lokasi penempatan data dalam tabel basis data menggunakan parameter `column_list` fungsi aws_s3.table_import_from_s3.

Untuk contoh ini, asumsikan bahwa informasi berikut ini diatur ke dalam kolom yang dipisahkan tanda pipa di file Amazon S3.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

Untuk mengimpor file yang menggunakan pemisah kustom

1. Buat tabel dalam basis data untuk data yang diimpor.

   ```
   postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
   ```

2. Gunakan bentuk berikut fungsi aws_s3.table_import_from_s3 berikut untuk mengimpor data dari file Amazon S3.

   Anda dapat memasukkan panggilan fungsi aws_commons.create_s3_uri sebaris dalam panggilan fungsi `aws_s3.table_import_from_s3` untuk menentukan file.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   'test',
   'a,b,d,e',
   'DELIMITER ''|''',
   aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'pipeDelimitedSampleFile', 'us-
east-2')
);
```

Data tersebut ini berada dalam tabel di kolom berikut.

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Mengimpor file terkompresi (gzip) Amazon S3

Contoh berikut menunjukkan cara mengimpor file dari Amazon S3 yang dikompresi menggunakan gzip. File yang Anda impor harus memiliki metadata Amazon S3 berikut:

- Kunci: `Content-Encoding`
- Nilai: `gzip`

Jika Anda mengunggah file menggunakan AWS Management Console, metadata biasanya diterapkan oleh sistem. Untuk informasi tentang mengunggah file ke Amazon S3 menggunakan AWS Management Console, API, AWS CLI atau API, lihat Mengunggah objek di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Untuk informasi selengkapnya tentang metadata Amazon S3 dan detail tentang metadata yang disediakan sistem, lihat Mengedit metadata objek di konsol Amazon S3 di Panduan Pengguna Amazon Simple Storage Service.

Impor file gzip ke dalam klaster DB Aurora PostgreSQL seperti yang ditunjukkan berikut ini.

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
```

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'amzn-s3-demo-bucket', 'test-data.gz', 'us-east-2'
);
```

Mengimpor file Amazon S3 yang dienkode

Contoh berikut menunjukkan cara mengimpor file dari Amazon S3 yang memiliki pengodean Windows-1252.

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'SampleFile', 'us-east-2')
);
```

## Referensi fungsi

Fungsi

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

Ekstensi `aws_s3` memberikan fungsi `aws_s3.table_import_from_s3`. Nilai yang ditampilkan berupa teks.

Sintaksis

Parameter yang diperlukan adalah `table_name`, `column_list`, dan `options`. Parameter ini mengidentifikasi tabel basis data dan menentukan cara data disalin ke dalam tabel.

Anda juga dapat menggunakan parameter berikut:

- Parameter `s3_info` menentukan file Amazon S3 yang akan diimpor.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
```

```
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- Parameter `credentials` menentukan kredensial untuk mengakses Amazon S3. Saat Anda menggunakan parameter ini, Anda tidak menggunakan IAM peran.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
   credentials aws_commons._aws_credentials_1
)
```

Parameter

table_name

Sebuah string teks yang diperlukan berisi nama tabel SQL database Postgre untuk mengimpor data ke dalam.

column_list

String teks yang diperlukan yang berisi daftar opsional kolom tabel SQL database Postgre untuk menyalin data. Jika string kosong, semua kolom tabel akan digunakan. Sebagai contoh, lihat Mengimpor file Amazon S3 yang menggunakan pemisah kustom.

options

Sebuah string teks yang diperlukan berisi argumen untuk perintah Postgre SQL`COPY`. Argumen ini menentukan bagaimana data akan disalin ke dalam tabel PostgreSQL. Untuk lebih jelasnya, lihat dokumentasi Postgre SQL COPY.

s3_info

Jenis komposit `aws_commons._s3_uri_1` yang berisi informasi tentang objek S3 berikut:

- `bucket` – Nama bucket Amazon S3 yang berisi file.
- `file_path` – Nama file Amazon S3 yang mencakup jalur file.
- `region`— AWS Wilayah tempat file tersebut berada. Untuk daftar AWS Nama wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

credentials

Jenis komposit `aws_commons._aws_credentials_1` yang berisi kredensial berikut yang akan digunakan untuk operasi impor:

- Kunci akses
- Kunci rahasia
- Token sesi

Untuk informasi tentang cara membuat struktur komposit `aws_commons._aws_credentials_1`, lihat aws_commons.create_aws_credentials.

Sintaksis alternatif

Untuk memudahkan pengujian, Anda dapat menggunakan serangkaian parameter yang diperluas, bukan parameter `s3_info` dan `credentials`. Berikut ini adalah variasi sintaks tambahan untuk fungsi `aws_s3.table_import_from_s3`:

- Alih-alih menggunakan parameter `s3_info` untuk mengidentifikasi file Amazon S3, gunakan kombinasi parameter `bucket`, `file_path`, dan `region`. Dengan bentuk fungsi ini, akses ke Amazon S3 disediakan oleh IAM peran pada instans SQL Postgre DB.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- Alih-alih menggunakan parameter `credentials` untuk menentukan akses Amazon S3, gunakan kombinasi parameter `access_key`, `session_key`, dan `session_token`.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text,
```

```
   access_key text,
   secret_key text,
   session_token text
)
```

Parameter alternatif

bucket

String teks yang berisi nama bucket Amazon S3 yang berisi file.

file_path

String teks yang berisi nama file Amazon S3 beserta jalur file.

region

Sebuah string teks yang mengidentifikasi Wilayah AWS lokasi file. Untuk daftar Wilayah AWS nama dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

access_key

String teks yang berisi kunci akses untuk digunakan dalam operasi impor. Defaultnya adalahNULL.

secret_key

String teks yang berisi kunci rahasia yang akan digunakan dalam operasi impor. Defaultnya adalahNULL.

session_token

(Opsional) String teks yang berisi kunci sesi yang akan digunakan dalam operasi impor. Defaultnya adalahNULL.

aws_commons.create_s3_uri

Membuat struktur `aws_commons._s3_uri_1` untuk menyimpan informasi file Amazon S3. Gunakan hasil dari fungsi `aws_commons.create_s3_uri` di parameter `s3_info` dari fungsi aws_s3.table_import_from_s3.

Sintaksis

```
aws_commons.create_s3_uri(
```

```
   bucket text,
   file_path text,
   region text
)
```

Parameter

bucket

String teks yang diperlukan yang berisi nama bucket Amazon S3 untuk file tersebut.

file_path

String teks yang diperlukan yang berisi nama file Amazon S3 beserta jalurnya.

region

Sebuah string teks yang diperlukan yang berisi Wilayah AWS bahwa file tersebut ada di. Untuk daftar Wilayah AWS nama dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

aws_commons.create_aws_credentials

Mengatur kunci akses dan kunci rahasia dalam struktur `aws_commons._aws_credentials_1`. Gunakan hasil dari fungsi `aws_commons.create_aws_credentials` di parameter `credentials` dari fungsi aws_s3.table_import_from_s3.

Sintaksis

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
   session_token text
)
```

Parameter

access_key

String teks yang diperlukan berisi kunci akses yang digunakan untuk mengimpor file Amazon S3. Defaultnya adalahNULL.

secret_key

String teks yang diperlukan yang berisi kunci rahasia yang akan digunakan untuk mengimpor file Amazon S3. Defaultnya adalahNULL.

session_token

String teks opsional yang berisi token sesi yang akan digunakan untuk mengimpor file Amazon S3. Defaultnya adalahNULL. Jika Anda memberikan `session_token` opsional, Anda dapat menggunakan kredensial sementara.

## Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3

Anda dapat mengueri data dari klaster DB Aurora PostgreSQL dan mengekspornya langsung ke file yang disimpan dalam bucket Amazon S3. Untuk melakukannya, instal ekstensi Aurora PostgreSQL `aws_s3` terlebih dahulu. Ekstensi ini memberi Anda fungsi yang Anda gunakan untuk mengekspor hasil kueri ke Amazon S3. Berikut ini, Anda dapat mengetahui cara menginstal ekstensi dan cara mengekspor data ke Amazon S3.

Anda dapat mengekspor dari yang disediakan atau Aurora Serverless v2 contoh DB. Langkah-langkah ini tidak didukung untuk Aurora Serverless v1.

> **Note**
>
> Ekspor lintas akun ke Amazon S3 tidak didukung.

Semua versi Aurora PostgreSQL yang tersedia saat ini mendukung ekspor data ke Amazon Simple Storage Service. Untuk informasi versi terperinci, lihat Pembaruan Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Jika Anda belum menyiapkan bucket untuk ekspor, lihat topik berikut, Panduan Pengguna Amazon Simple Storage Service.

- Menyiapkan Amazon S3
- Membuat ember

Secara default, data yang diekspor dari Aurora PostgreSQL ke Amazon S3 menggunakan enkripsi sisi server dengan. Kunci yang dikelola AWS Anda dapat menggunakan kunci terkelola pelanggan

yang telah Anda buat. Jika Anda menggunakan enkripsi bucket, bucket Amazon S3 harus dienkripsi dengan AWS Key Management Service (AWS KMS) key (SSE-KMS). Saat ini, bucket yang dienkripsi dengan kunci terkelola Amazon S3 (SSE-S3) tidak didukung.

> **Note**
>
> Anda dapat menyimpan data snapshot cluster DB dan DB ke Amazon S3 menggunakan AWS Management Console AWS CLI,, atau Amazon RDS API. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Topik


- Menginstal ekstensi aws_s3
- Ikhtisar ekspor data ke Amazon S3
- Menentukan jalur file Amazon S3 tujuan ekspor
- Menyiapkan akses ke bucket Amazon S3
- Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3
- Referensi fungsi
- Memecahkan masalah akses ke Amazon S3


## Menginstal ekstensi aws_s3

Sebelum Anda dapat menggunakan Amazon Simple Storage Service dengan klaster DB Aurora PostgreSQL, Anda perlu menginstal ekstensi `aws_s3`. Ekstensi ini memberikan fungsi untuk mengekspor data dari instans penulis klaster DB Aurora PostgreSQL ke bucket Amazon S3. Ini juga menyediakan fungsi untuk mengimpor data dari Amazon S3. Untuk informasi selengkapnya, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL. Ekstensi `aws_s3` bergantung pada beberapa fungsi pembantu dalam ekstensi `aws_commons`, yang diinstal secara otomatis bila diperlukan.

Untuk menginstal ekstensi **`aws_s3`**

1. Gunakan psql (atau pgAdmin) untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL sebagai pengguna yang memiliki hak istimewa `rds_superuser`. Jika Anda menyimpan nama default selama proses penyiapan, Anda terhubung sebagai `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. Untuk menginstal ekstensi, jalankan perintah berikut.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. Untuk memverifikasi bahwa ekstensi sudah diinstal, Anda dapat menggunakan metacommand psql `\dx`.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

Fungsi untuk mengimpor data dari Amazon S3 dan mengekspor data ke Amazon S3 kini dapat digunakan.

Verifikasi bahwa versi Aurora PostgreSQL Anda mendukung ekspor ke Amazon S3

Anda dapat memverifikasi bahwa versi RDS for PostgreSQL Anda mendukung ekspor ke Amazon S3 dengan menggunakan perintah `describe-db-engine-versions`. Contoh berikut memeriksa untuk melihat apakah versi 10.14 dapat mengekspor ke Amazon S3.

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

Jika output-nya menyertakan string `"s3Export"`, berarti mesinnya mendukung ekspor Amazon S3. Jika tidak, mesin tidak mendukungnya.

## Ikhtisar ekspor data ke Amazon S3

Untuk mengekspor data yang disimpan dalam basis data Aurora PostgreSQL ke bucket Amazon S3, gunakan prosedur berikut.

Untuk mengekspor data Aurora PostgreSQL ke S3

1. Identifikasi jalur file Amazon S3 yang akan digunakan untuk mengekspor data. Untuk detail tentang proses ini, lihat Menentukan jalur file Amazon S3 tujuan ekspor.
2. Berikan izin untuk mengakses bucket Amazon S3.

   Untuk mengekspor data ke file Amazon S3, beri klaster DB Aurora PostgreSQL izin untuk mengakses bucket Amazon S3 yang akan digunakan untuk penyimpanan data yang diekspor. Berikut adalah langkah-langkahnya:

   1. Buat kebijakan IAM yang memberikan akses ke bucket Amazon S3 tempat tujuan ekspor.
   2. Buat peran IAM.
   3. Lampirkan kebijakan yang Anda buat ke peran yang Anda buat.
   4. Tambahkan peran IAM ini ke klaster DB .

   Untuk detail tentang proses ini, lihat Menyiapkan akses ke bucket Amazon S3.
3. Identifikasi kueri basis data untuk mendapatkan data. Ekspor data kueri dengan memanggil fungsi `aws_s3.query_export_to_s3`.

   Setelah menyelesaikan tugas persiapan sebelumnya, gunakan fungsi aws_s3.query_export_to_s3 untuk mengekspor hasil kueri ke Amazon S3. Untuk detail tentang proses ini, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

## Menentukan jalur file Amazon S3 tujuan ekspor

Tentukan informasi berikut untuk mengidentifikasi lokasi di Amazon S3 tempat Anda ingin mengekspor data:

- Nama bucket – Bucket adalah kontainer untuk objek atau file Amazon S3.

  Untuk informasi selengkapnya tentang menyimpan data dengan Amazon S3, lihat Membuat bucket dan Bekerja dengan objek di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.
- Jalur file – Jalur file mengidentifikasi tempat penyimpanan data yang diekspor dalam bucket Amazon S3. Jalur file terdiri atas:
  - Awalan jalur opsional yang mengidentifikasi jalur folder virtual.

  - Awalan file yang mengidentifikasi satu atau beberapa file yang akan disimpan. Ekspor yang lebih besar disimpan dalam beberapa file, masing-masing berukuran maksimum sekitar 6 GB. Nama file tambahan memiliki awalan file yang sama, tetapi dengan penambahan `_part`*XX*. *XX* mewakili 2, lalu 3, dan seterusnya.

  Misalnya, jalur file dengan folder `exports` dan awalan file `query-1-export` adalah `/exports/query-1-export`.

- AWS Wilayah (opsional) - AWS Wilayah tempat bucket Amazon S3 berada.

  > **Note**
  >
  > Saat ini, AWS Wilayah harus sama dengan wilayah DB cluster DB yang mengekspor.

  Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

Untuk menyimpan informasi file Amazon S3 tentang lokasi penyimpanan file yang diekspor, Anda dapat menggunakan fungsi aws_commons.create_s3_uri untuk membuat struktur komposit `aws_commons._s3_uri_1` sebagai berikut.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Kemudian, berikan nilai `s3_uri_1` ini sebagai parameter untuk memanggil fungsi aws_s3.query_export_to_s3. Sebagai contoh, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

## Menyiapkan akses ke bucket Amazon S3

Untuk mengekspor data ke Amazon S3, berikan izin kepada klaster DB PostgreSQL Anda untuk mengakses bucket Amazon S3 yang akan dimasuki file.

Untuk melakukannya, gunakan prosedur berikut.

Untuk memberi klaster DB PostgreSQL akses ke Amazon S3 melalui peran IAM

1. Buat kebijakan IAM.

   Kebijakan ini memberikan izin kepada bucket dan objek yang memungkinkan klaster DB PostgreSQL Anda mengakses Amazon S3.

   Sebagai bagian dari pembuatan kebijakan ini, lakukan langkah-langkah berikut:

   a. Sertakan tindakan yang diperlukan berikut dalam kebijakan untuk mengizinkan transfer file dari klaster DB PostgreSQL ke bucket Amazon S3:

      - `s3:PutObject`
      - `s3:AbortMultipartUpload`

   b. Sertakan Amazon Resource Name (ARN) yang mengidentifikasi bucket dan objek Amazon S3 dalam bucket. Format ARN untuk mengakses Amazon S3 adalah: `arn:aws:s3:::amzn-s3-demo-bucket/*`

   Untuk informasi selengkapnya tentang cara membuat kebijakan IAM untuk Aurora PostgreSQL, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda di Panduan Pengguna IAM.

   AWS CLI Perintah berikut membuat kebijakan IAM bernama `rds-s3-export-policy` dengan opsi ini. Ini memberikan akses ke bucket bernama *amzn-s3-demo-bucket*.

   > **Warning**
   >
   > Sebaiknya Anda menyiapkan basis data Anda dengan VPC privat yang memiliki kebijakan titik akhir yang dikonfigurasi untuk mengakses bucket tertentu. Untuk informasi selengkapnya, lihat Menggunakan kebijakan titik akhir untuk Amazon S3 di Panduan Pengguna Amazon VPC.
   > Kami sangat menyarankan Anda agar tidak membuat kebijakan dengan akses semua sumber daya. Akses ini dapat menjadi ancaman bagi data keamanan. Jika Anda membuat kebijakan yang memberi akses `S3:PutObject` ke semua sumber daya menggunakan `"Resource":"*"`, pengguna yang memiliki hak istimewa ekspor

dapat mengekspor data ke semua bucket di akun Anda. Selain itu, pengguna dapat mengekspor data ke bucket yang dapat ditulis secara publik Wilayah AWS Anda.

Setelah Anda membuat kebijakan, catat Amazon Resource Name (ARN) dari kebijakan tersebut. Anda memerlukan ARN ini untuk langkah berikutnya ketika Anda melampirkan kebijakan ke peran IAM.

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject*",
           "s3:ListBucket",
           "s3:GetObject*",
           "s3:DeleteObject*",
           "s3:GetBucketLocation",
           "s3:AbortMultipartUpload"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

2. Buat peran IAM.

   Lakukan langkah ini agar Aurora PostgreSQL dapat mengambil peran IAM ini atas nama Anda untuk mengakses bucket Amazon S3. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke pengguna IAM dalam Panduan Pengguna IAM.

   Sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan berbasis sumber daya untuk membatasi izin layanan ke sumber daya tertentu. Ini adalah cara paling efektif untuk melindungi dari masalah deputi yang membingungkan.

Jika Anda menggunakan kunci konteks kondisi global dan nilai `aws:SourceArn` berisi ID akun, nilai `aws:SourceAccount` dan akun dalam nilai `aws:SourceArn` harus menggunakan ID akun yang sama saat digunakan dalam pernyataan kebijakan yang sama.

- Gunakan `aws:SourceArn` jika Anda menginginkan akses lintas layanan untuk satu sumber daya.
- Gunakan `aws:SourceAccount` jika Anda ingin mengizinkan sumber daya apa pun di akun tersebut dikaitkan dengan penggunaan lintas layanan.

Dalam kebijakan, pastikan untuk menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN penuh sumber daya. Contoh berikut menunjukkan bagaimana melakukannya dengan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-export-role`.

Example

Untuk Linux, macOS, atau Unix:

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
             }
       }
     ]
   }'
```

Untuk Windows:

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
             }
       }
     ]
   }'
```

3. Lampirkan kebijakan IAM yang Anda buat pada peran IAM yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat sebelumnya ke peran bernama `rds-s3-export-role`. Ganti *`your-policy-arn`* dengan ARN kebijakan yang Anda catat di langkah sebelumnya.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

4. Tambahkan peran IAM ke klaster DB. Anda melakukannya dengan menggunakan AWS Management Console atau AWS CLI, seperti yang dijelaskan berikut.

Konsol

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih nama klaster DB PostgreSQL untuk menampilkan detailnya.
3. Di tab Konektivitas & keamanan, di bagian Kelola peran IAM, pilih peran yang akan ditambahkan pada bagian Tambahkan peran IAM ke instans ini.
4. Di bagian Fitur, pilih s3Export.
5. Pilih Tambahkan peran.

AWS CLI

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan CLI

- Gunakan perintah berikut untuk menambahkan peran ke klaster DB PostgreSQL bernama `my-db-cluster`. Ganti *`your-role-arn`* dengan ARN peran yang Anda catat pada langkah sebelumnya. Gunakan `s3Export` untuk nilai opsi `--feature-name`.

  Example

  Untuk Linux, macOS, atau Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Export \
   --role-arn your-role-arn   \
   --region your-region
```

  Untuk Windows:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Export ^
   --role-arn your-role-arn ^
   --region your-region
```

## Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3

Ekspor data PostgreSQL Anda ke Amazon S3 dengan memanggil fungsi aws_s3.query_export_to_s3.

Topik

- Prasyarat
- Memanggil aws_s3.query_export_to_s3
- Mengekspor ke file CSV yang menggunakan pembatas kustom
- Mengekspor ke file biner dengan pengodean

### Prasyarat

Sebelum menggunakan fungsi `aws_s3.query_export_to_s3`, pastikan untuk melengkapi prasyarat berikut:

- Instal ekstensi PostgreSQL yang diperlukan seperti yang dijelaskan di Ikhtisar ekspor data ke Amazon S3.
- Tentukan tempat untuk mengekspor data Anda ke Amazon S3 seperti yang dijelaskan di Menentukan jalur file Amazon S3 tujuan ekspor.
- Pastikan bahwa klaster DB memiliki akses ekspor ke Amazon S3 seperti yang dijelaskan di Menyiapkan akses ke bucket Amazon S3.

Contoh berikut menggunakan tabel basis data yang disebut `sample_table`. Contoh ini mengekspor data ke dalam bucket bernama *`amzn-s3-demo-bucket`*. Contoh tabel dan data dibuat dengan pernyataan SQL berikut di psql.

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

### Memanggil aws_s3.query_export_to_s3

Berikut ini adalah cara-cara dasar untuk memanggil fungsi aws_s3.query_export_to_s3.

Contoh ini menggunakan variabel `s3_uri_1` untuk mengidentifikasi struktur berisi informasi yang mengidentifikasi file Amazon S3. Gunakan fungsi aws_commons.create_s3_uri untuk membuat struktur.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Meskipun parameter bervariasi untuk dua panggilan fungsi `aws_s3.query_export_to_s3` berikut, hasilnya sama untuk contoh ini. Semua baris dari tabel `sample_table` diekspor ke dalam bucket yang disebut *amzn-s3-demo-bucket*.

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

Parameternya dijelaskan sebagai berikut:

- `'SELECT * FROM sample_table'` – Parameter pertama adalah string teks wajib yang berisi kueri SQL. Mesin PostgreSQL menjalankan kueri ini. Hasil kueri disalin ke bucket S3 yang diidentifikasi dalam parameter lain.
- `:'s3_uri_1'` – Parameter ini adalah struktur yang mengidentifikasi file Amazon S3. Contoh ini menggunakan variabel untuk mengidentifikasi struktur yang dibuat sebelumnya. Anda dapat membuat struktur dengan menyertakan baris panggilan fungsi `aws_commons.create_s3_uri` sebaris dalam panggilan fungsi `aws_s3.query_export_to_s3` sebagai berikut.

  ```
  SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
     aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'sample-filepath', 'us-west-2')
  );
  ```

- `options :='format text'` – Parameter `options` adalah string teks opsional yang berisi argumen `COPY` PostgreSQL. Proses penyalinan menggunakan argumen dan format perintah PostgreSQL COPY.

Jika file yang ditentukan tidak ada dalam bucket Amazon S3, file tersebut akan dibuat. Jika file sudah ada, file tersebut akan ditimpa. Sintaks untuk mengakses data yang diekspor di Amazon S3 adalah sebagai berikut.

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

Ekspor yang lebih besar disimpan dalam beberapa file, masing-masing berukuran maksimum sekitar 6 GB. Nama file tambahan memiliki awalan file yang sama, tetapi dengan penambahan _part*XX*. *XX* mewakili 2, lalu 3, dan seterusnya. Sebagai contoh, misalkan Anda menentukan jalur tempat Anda menyimpan file data sebagai berikut.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

Jika ekspor harus membuat tiga file data, bucket Amazon S3 berisi file data berikut.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part2
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part3
```

Untuk referensi selengkapnya tentang fungsi ini dan cara lain untuk memanggilnya, lihat aws_s3.query_export_to_s3. Untuk informasi selengkapnya tentang cara mengakses file di Amazon S3, buka Melihat objek dalam Panduan Pengguna Amazon Simple Storage Service.

Mengekspor ke file CSV yang menggunakan pembatas kustom

Contoh berikut menunjukkan cara memanggil fungsi aws_s3.query_export_to_s3 untuk mengekspor data ke file yang menggunakan pembatas kustom. Contoh ini menggunakan argumen perintah PostgreSQL COPY untuk menetapkan format nilai yang dipisahkan koma (CSV) dan pembatas titik dua (:).

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

Mengekspor ke file biner dengan pengodean

Contoh berikut menunjukkan cara memanggil fungsi aws_s3.query_export_to_s3 untuk mengekspor data ke file biner yang memiliki pengodean Windows-1253.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

## Referensi fungsi

### Fungsi

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

### aws_s3.query_export_to_s3

Mengekspor hasil kueri PostgreSQL ke bucket Amazon S3. Ekstensi `aws_s3` memberikan fungsi `aws_s3.query_export_to_s3`.

Dua parameter yang dibutuhkan adalah `query` dan `s3_info`. Parameter ini menentukan kueri yang akan diekspor dan mengidentifikasi bucket Amazon S3 tempat tujuan ekspor. Parameter opsional yang disebut `options` disediakan untuk menentukan berbagai parameter ekspor. Sebagai contoh penggunaan fungsi `aws_s3.query_export_to_s3`, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

#### Sintaksis

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
    kms_key text
)
```

#### Parameter input

query

String teks yang diperlukan yang berisi kueri SQL yang dijalankan mesin PostgreSQL. Hasil kueri ini disalin ke bucket S3 yang diidentifikasi dalam parameter `s3_info`.

s3_info

Jenis komposit `aws_commons._s3_uri_1` yang berisi informasi tentang objek S3 berikut:

- `bucket` – Nama bucket Amazon S3 yang akan diisi file.
- `file_path` – Nama dan jalur file Amazon S3.
- `region`— AWS Wilayah tempat ember berada. Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

Saat ini, nilai ini harus AWS Wilayah yang sama dengan DB cluster DB yang mengekspor. Defaultnya adalah AWS Wilayah DB cluster DB yang mengekspor.

Untuk membuat struktur komposit `aws_commons._s3_uri_1`, lihat fungsi aws_commons.create_s3_uri.

options

String teks opsional yang berisi argumen untuk perintah `COPY` PostgreSQL. Argumen ini menentukan cara menyalin data saat diekspor. Untuk detail selengkapnya, lihat Dokumentasi PostgreSQL COPY.

teks kms_key

String teks opsional yang berisi kunci KMS yang dikelola pelanggan dari bucket S3 untuk mengekspor data.

Parameter input alternatif

Untuk memudahkan pengujian, Anda dapat menggunakan serangkaian parameter yang diperluas, bukan parameter `s3_info`. Berikut ini adalah variasi sintaks tambahan untuk fungsi `aws_s3.query_export_to_s3`.

Alih-alih menggunakan parameter `s3_info` untuk mengidentifikasi file Amazon S3, gunakan kombinasi parameter `bucket`, `file_path`, dan `region`.

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

String teks yang diperlukan yang berisi kueri SQL yang dijalankan mesin PostgreSQL. Hasil kueri ini disalin ke bucket S3 yang diidentifikasi dalam parameter `s3_info`.

bucket

String teks yang diperlukan yang berisi nama bucket Amazon S3 yang berisi file.

file_path

String teks yang diperlukan yang berisi nama file Amazon S3 beserta jalurnya.

region

String teks opsional yang berisi AWS Wilayah tempat bucket berada. Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

Saat ini, nilai ini harus AWS Wilayah yang sama dengan DB cluster DB yang mengekspor. Defaultnya adalah AWS Wilayah DB cluster DB yang mengekspor.

options

String teks opsional yang berisi argumen untuk perintah `COPY` PostgreSQL. Argumen ini menentukan cara menyalin data saat diekspor. Untuk detail selengkapnya, lihat Dokumentasi PostgreSQL COPY.

teks kms_key

String teks opsional yang berisi kunci KMS yang dikelola pelanggan dari bucket S3 untuk mengekspor data.

Parameter output

```
aws_s3.query_export_to_s3(
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

Jumlah baris tabel yang berhasil diunggah ke Amazon S3 untuk kueri tertentu.

files_uploaded

Jumlah file yang diunggah ke Amazon S3. File dibuat dalam ukuran kira-kira 6 GB. Setiap file tambahan yang dibuat memiliki `_part`*XX* yang ditambahkan pada namanya. *XX* mewakili 2, kemudian 3, dan seterusnya sesuai kebutuhan.

bytes_uploaded

Jumlah total byte yang diunggah ke Amazon S3.

Contoh

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Membuat struktur `aws_commons._s3_uri_1` untuk menyimpan informasi file Amazon S3. Gunakan hasil dari fungsi `aws_commons.create_s3_uri` dalam parameter `s3_info` dari fungsi aws_s3.query_export_to_s3. Untuk contoh penggunaan fungsi `aws_commons.create_s3_uri`, lihat Menentukan jalur file Amazon S3 tujuan ekspor.

Sintaksis

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Parameter input

bucket

String teks yang diperlukan yang berisi nama bucket Amazon S3 untuk file tersebut.

file_path

String teks yang diperlukan yang berisi nama file Amazon S3 beserta jalurnya.

region

String teks yang diperlukan yang berisi AWS Wilayah tempat file tersebut berada. Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

## Memecahkan masalah akses ke Amazon S3

Jika Anda mengalami masalah koneksi saat mencoba mengekspor data ke Amazon S3, pertama pastikan aturan akses keluar untuk grup keamanan VPC yang terkait dengan instans DB Anda mengizinkan konektivitas jaringan. Secara khusus, grup keamanan harus memiliki aturan yang memungkinkan instans DB mengirim lalu lintas TCP ke port 443 dan ke IPv4 alamat apa pun (0.0.0.0/0). Untuk informasi selengkapnya, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan.

Lihat juga rekomendasi berikut:

- Memecahkan masalah identitas dan akses Amazon Aurora
- Memecahkan Masalah Amazon S3 di Panduan Pengguna Amazon Simple Storage Service
- Memecahkan Masalah Amazon S3 dan IAM di Panduan Pengguna IAM

AWS Lambda adalah layanan komputasi berbasis peristiwa yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Misalnya, Anda dapat menggunakan fungsi Lambda untuk memproses pemberitahuan peristiwa dari basis data, atau memuat data dari file setiap kali file baru diunggah ke Amazon S3. Untuk mempelajari lebih lanjut tentang Lambda, lihat Apa itu? AWS Lambda di Panduan AWS Lambda Pengembang.

> **Note**
>
> AWS Lambda Fungsi pemanggilan didukung di Aurora PostgreSQL 11.9 dan lebih tinggi (termasuk Aurora Serverless v2).

Berikut ini, Anda dapat menemukan ringkasan langkah-langkah yang diperlukan.

Untuk informasi selengkapnya tentang fungsi Lambda, lihat Mulai menggunakan Lambda dan Dasar-dasar AWS Lambda di Panduan Developer AWS Lambda .

Topik

- Langkah 1: Konfigurasikan Aurora PostgreSQL DB cluster Anda untuk instance PostgreSQL DB untuk koneksi keluar ke AWS Lambda
- AWS Lambda

- Langkah 3: Instal ekstensi aws_lambda untuk klaster DB Aurora PostgreSQL
- Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional)
- Langkah 5: Invokasi fungsi Lambda dari klaster DB Aurora PostgreSQL Anda
- Langkah 6: Berikan pengguna lain izin untuk menginvokasi fungsi Lambda
- 
- Pesan kesalahan fungsi Lambda
- AWS Lambdafungsi dan referensi parameter

## Langkah 1: Konfigurasikan Aurora PostgreSQL DB cluster Anda untuk instance PostgreSQL DB untuk koneksi keluar ke AWS Lambda

Fungsi Lambda selalu berjalan di dalam VPC Amazon yang dimiliki oleh layanan. AWS Lambda Lambda menerapkan akses jaringan dan aturan keamanan untuk VPC ini dan mempertahankan serta memantau VPC secara otomatis. Klaster DB Aurora PostgreSQL Anda mengirimkan lalu lintas jaringan ke VPC layanan Lambda. Cara Anda mengonfigurasi ini bergantung pada apakah instans DB primer klaster DB Aurora Anda bersifat publik atau pribadi.

- Public Aurora PostgreSQL DB cluster  utama cluster DB adalah publik jika terletak di adalah. PubliclyAccessible `true` Untuk menemukan nilai properti ini, Anda dapat menggunakan describe-db-instances AWS CLI perintah. Atau, Anda dapat menggunakan AWS Management Console untuk membuka tab Konektivitas & keamanan dan memeriksa apakah opsi Dapat diakses publik adalah Ya. Untuk memverifikasi bahwa instans ini ada di subnet publik VPC, Anda dapat menggunakan AWS Management Console atau AWS CLI.

  Untuk mengatur akses ke Lambda, Anda menggunakan AWS Management Console atau AWS CLI untuk membuat aturan keluar pada grup keamanan VPC Anda. Aturan keluar menentukan bahwa TCP dapat menggunakan port 443 untuk mengirim paket ke alamat apa pun IPv4 (0.0.0.0/0).
- Private Aurora PostgreSQL DB cluster  — Dalam hal ini, properti “” instance berada di subnet pribadi. PubliclyAccessible `false` Agar instans dapat berfungsi dengan Lambda, Anda dapat menggunakan gateway Network Address Translation (NAT). Untuk informasi selengkapnya, silakan lihat Gateway NAT. Atau, Anda dapat mengonfigurasi VPC dengan titik akhir VPC untuk Lambda. Untuk informasi selengkapnya, lihat Titik akhir VPC di Panduan Pengguna Amazon VPC. Titik akhir ini merespons panggilan yang dilakukan oleh klaster DB Aurora PostgreSQL ke fungsi Lambda Anda.

VPC Anda sekarang dapat berinteraksi dengan AWS Lambda VPC di tingkat jaringan. Selanjutnya, konfigurasikan izin menggunakan IAM.

## AWS Lambda

Menginvokasi fungsi Lambda dari klaster DB Aurora PostgreSQL memerlukan hak istimewa tertentu. Untuk mengonfigurasi hak istimewa yang diperlukan, sebaiknya Anda membuat kebijakan IAM yang memungkinkan invokasi fungsi Lambda, menetapkan kebijakan ke peran, dan kemudian menerapkan peran tersebut ke klaster DB. Pendekatan ini memberikan hak istimewa klaster DB untuk menginvokasi fungsi Lambda yang ditentukan atas nama Anda. Langkah-langkah berikut menunjukkan cara melakukannya dengan menggunakan AWS CLI.

Untuk mengonfigurasi izin IAM untuk menggunakan klaster dengan Lambda

1. (ID pernyataan (Sid) adalah deskripsi opsional untuk pernyataan kebijakan Anda dan tidak berpengaruh pada penggunaan.) Kebijakan ini memberi klaster DB Aurora izin minimum yang diperlukan untuk menginvokasi fungsi Lambda yang ditentukan.

   ```
   aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
       "Version": "2012-10-17",
       "Statement": [
           {
           "Sid": "AllowAccessToExampleFunction",
           "Effect": "Allow",
           "Action": "lambda:InvokeFunction",
           "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
           }
       ]
   }'
   ```

   Sebagai alternatif, Anda dapat menggunakan kebijakan `AWSLambdaRole` yang ditentukan sebelumnya yang memungkinkan Anda menginvokasi fungsi Lambda apa pun. Untuk informasi selengkapnya, lihat Kebijakan IAM berbasis identitas untuk Lambda

2. Gunakan AWS CLI perintah create-role untuk membuat peran IAM yang dapat diasumsikan oleh kebijakan saat runtime.

   ```
   aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
       "Version": "2012-10-17",
       "Statement": [
           {
   ```

```
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Terapkan kebijakan ke peran dengan menggunakan attach-role-policy AWS CLI perintah.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/rds/add-role-to-db-cluster.html AWS CLI Langkah terakhir ini memungkinkan pengguna basis data klaster DB menginvokasi fungsi Lambda.

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

Setelah menyelesaikan konfigurasi VPC dan IAM, Anda sekarang dapat menginstal ekstensi `aws_lambda`. (Perhatikan bahwa Anda dapat menginstal ekstensi kapan saja, tetapi sebelum menyiapkan dukungan VPC dan hak istimewa IAM yang benar, ekstensi `aws_lambda` tidak menambahkan apa pun ke kapabilitas klaster DB Aurora PostgreSQL.)

## Langkah 3: Instal ekstensi **`aws_lambda`** untuk klaster DB Aurora PostgreSQL

Ekstensi ini memberi klaster DB Aurora PostgreSQL kemampuan untuk memanggil fungsi Lambda dari PostgreSQL.

Untuk menginstal ekstensi **`aws_lambda`** di klaster DB Aurora PostgreSQL Anda

Gunakan baris perintah `psql` PostgreSQL atau alat pgAdmin untuk terhubung ke klaster DB Aurora PostgreSQL .

1. Hubungkan ke klaster DB Aurora PostgreSQL sebagai pengguna dengan hak istimewa `rds_superuser`. Pengguna `postgres` default ditampilkan dalam contoh.

```
psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
 5432
```

2. Instal ekstensi `aws_lambda`. Ekstensi `aws_commons` juga diperlukan. Ini memberikan fungsi pembantu `aws_lambda` dan berbagai ekstensi Aurora lainnya untuk PostgreSQL. Jika belum ada di klaster DB Aurora PostgreSQL , ekstensi akan diinstal dengan `aws_lambda` seperti yang ditunjukkan sebagai berikut.

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

Ekstensi `aws_lambda` diinstal di instans DB primer klaster DB Aurora PostgreSQL. Anda sekarang dapat membuat struktur kemudahan untuk menginvokasi fungsi Lambda.

## Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional)

Anda dapat menggunakan fungsi pembantu di ekstensi `aws_commons` untuk menyiapkan entitas yang dapat diinvokasi dengan lebih mudah dari PostgreSQL. Untuk melakukannya, Anda harus memiliki informasi berikut tentang fungsi Lambda:

- Nama fungsi – Nama, Amazon Resource Name (ARN), versi, atau alias fungsi Lambda. Kebijakan IAM yang dibuat Langkah 2: Konfigurasikan IAM untuk klaster dan Lambda memerlukan ARN, jadi kami sarankan Anda menggunakan ARN fungsi Anda.
- AWS Wilayah

Untuk menyimpan informasi nama fungsi Lambda, Anda menggunakan fungsi aws_commons.create_lambda_function_arn. Fungsi pembantu ini menciptakan struktur komposit `aws_commons._lambda_function_arn_1` dengan detail yang dibutuhkan oleh fungsi invokasi. Berikut ini, Anda dapat menemukan tiga pendekatan alternatif untuk menyiapkan struktur komposit ini.

```
SELECT aws_commons.create_lambda_function_arn(
```

```
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

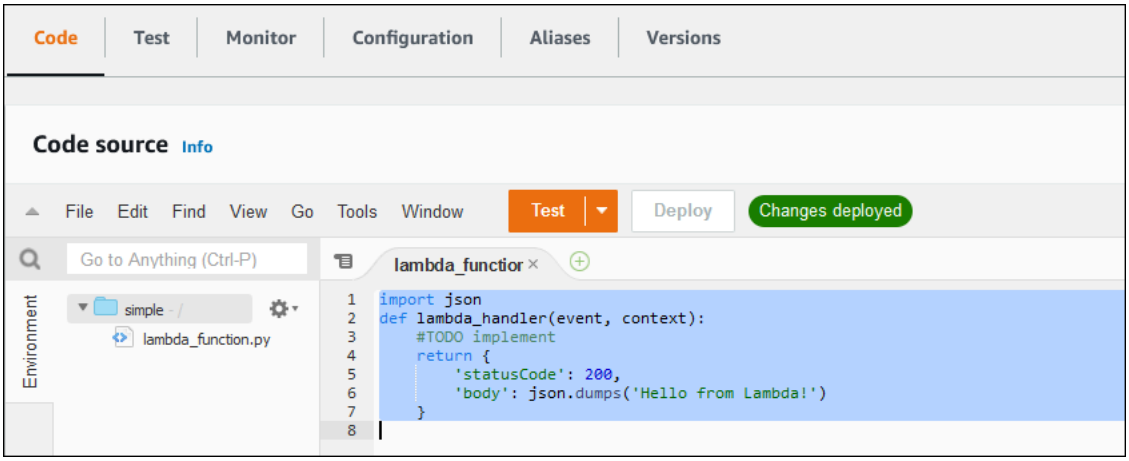
Salah satu dari nilai ini dapat digunakan dalam panggilan ke fungsi aws_lambda.invoke. Sebagai contoh, lihat Langkah 5: Invokasi fungsi Lambda dari klaster DB Aurora PostgreSQL Anda.

### Langkah 5: Invokasi fungsi Lambda dari klaster DB Aurora PostgreSQL Anda

Fungsi `aws_lambda.invoke` berperilaku sinkron atau asinkron, bergantung pada `invocation_type`. Dua alternatif untuk parameter ini adalah `RequestResponse` (default) dan `Event`, sebagai berikut.

- **`RequestResponse`** – Jenis invokasi ini sinkron. Ini adalah perilaku default saat panggilan dilakukan tanpa menentukan jenis invokasi. Payload respons mencakup hasil dari fungsi `aws_lambda.invoke`. Gunakan jenis invokasi ini jika alur kerja Anda perlu menerima hasil dari fungsi Lambda sebelum melanjutkan.
- **`Event`** – Jenis invokasi ini asinkron. Respons tidak mencakup payload yang berisi hasil. Gunakan jenis invokasi ini jika alur kerja Anda tidak memerlukan hasil dari fungsi Lambda untuk melanjutkan pemrosesan.

Sebagai pengujian sederhana terhadap pengaturan Anda, Anda dapat terhubung ke instans DB menggunakan `psql` dan menginvokasi contoh fungsi dari baris perintah. Misalkan Anda memiliki salah satu fungsi dasar yang disiapkan pada layanan Lambda, seperti fungsi Python sederhana yang diperlihatkan pada tangkapan layar berikut.




Untuk menginvokasi contoh fungsi

1. Hubungkan ke instans DB primer menggunakan `psql` atau pgAdmin.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Invokasi fungsi menggunakan ARN-nya.

```
SELECT * from
 aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
 Postgres!"}'::json );
```

   Respons-nya terlihat sebagai berikut.

```
status_code |                         payload                         |
 executed_version | log_result
-------------+---------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

Jika upaya invokasi tidak berhasil, lihat Pesan kesalahan fungsi Lambda .

## Langkah 6: Berikan pengguna lain izin untuk menginvokasi fungsi Lambda

Dalam langkah ini, hanya Anda sebagai `rds_superuser` yang dapat menginvokasi fungsi Lambda. Untuk mengizinkan pengguna lain menginvokasi fungsi apa pun yang Anda buat, Anda harus memberi mereka izin.

Untuk memberi pengguna lain izin untuk menginvokasi fungsi Lambda

1. Hubungkan ke instans DB primer menggunakan `psql` atau pgAdmin.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Jalankan perintah SQL berikut:

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

Berikut ini, Anda dapat menemukan beberapa contoh pemanggilan fungsi aws_lambda.invoke. Sebagian besar contoh menggunakan struktur komposit `aws_lambda_arn_1` yang Anda buat Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional) untuk menyederhanakan meneruskan detail fungsi. Untuk contoh panggilan asinkron, lihat Contoh: Invokasi fungsi Lambda asinkron (Event). Semua contoh lain yang tercantum menggunakan panggilan sinkron.

Untuk mempelajari lebih lanjut tentang jenis pemanggilan Lambda, lihat Memanggil fungsi Lambda di AWS Lambda Panduan Pengembang. Untuk informasi selengkapnya tentang `aws_lambda_arn_1`, lihat aws_commons.create_lambda_function_arn.

Daftar contoh

- Contoh: Synchronous (RequestResponse) pemanggilan fungsi Lambda
- Contoh: Invokasi fungsi Lambda asinkron (Event)
- Contoh: Menangkap log eksekusi Lambda dalam respons fungsi
- Contoh: Menyertakan konteks klien dalam fungsi Lambda
- Contoh: Menginvokasi fungsi Lambda versi spesifik

Contoh: Synchronous (RequestResponse) pemanggilan fungsi Lambda

Berikut ini adalah dua contoh dari invokasi fungsi Lambda sinkron. Hasil dari panggilan fungsi `aws_lambda.invoke` ini sama.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

Parameternya dijelaskan sebagai berikut:

- `:'aws_lambda_arn_1'` – Parameter ini mengidentifikasi struktur komposit yang dibuat di[Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional)](), dengan fungsi pembantu `aws_commons.create_lambda_function_arn`. Anda juga dapat membuat struktur ini sebaris dalam panggilan `aws_lambda.invoke` Anda sebagai berikut.

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- `'{"body": "Hello from PostgreSQL!"}'::json`— JSON Payload untuk diteruskan ke fungsi Lambda.
- `'RequestResponse'` – Jenis invokasi Lambda.

Contoh: Invokasi fungsi Lambda asinkron (Event)

Berikut ini adalah contoh invokasi fungsi Lambda asinkron. Jenis invokasi `Event` menjadwalkan invokasi fungsi Lambda dengan payload input yang ditentukan dan segera kembali. Gunakan jenis invokasi `Event` di alur kerja tertentu yang tidak bergantung pada hasil fungsi Lambda.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

Contoh: Menangkap log eksekusi Lambda dalam respons fungsi

Anda dapat menyertakan 4 KB terakhir log eksekusi di respons fungsi dengan menggunakan parameter `log_type` dalam panggilan fungsi `aws_lambda.invoke` Anda. Secara default, parameter ini diatur ke `None`, tetapi Anda dapat menentukan `Tail` untuk menangkap hasil log eksekusi Lambda dalam respons, seperti yang ditunjukkan berikut.

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Atur parameter `log_type` fungsi aws_lambda.invoke ke `Tail` untuk menyertakan log eksekusi dalam respons. Nilai default untuk parameter `log_type` adalah `None`.

`log_result` yang ditampilkan string yang dienkode `base64`. Anda dapat memecahkan kode konten menggunakan kombinasi fungsi `decode` dan `convert_from` PostgreSQL.

Untuk informasi selengkapnya tentang `log_type`, lihat aws_lambda.invoke.

Contoh: Menyertakan konteks klien dalam fungsi Lambda

Fungsi `aws_lambda.invoke` memiliki parameter `context` yang dapat Anda gunakan untuk meneruskan informasi yang terpisah dari payload, seperti yang diperlihatkan di bawah.

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Untuk menyertakan konteks klien, gunakan JSON objek untuk `context` parameter aws_lambda.invoke fungsi.

Untuk informasi selengkapnya tentang parameter `context`, lihat referensi aws_lambda.invoke.

Contoh: Menginvokasi fungsi Lambda versi spesifik

Anda dapat menentukan versi tertentu dari fungsi Lambda dengan menyertakan parameter `qualifier` dengan panggilan `aws_lambda.invoke`. Berikut ini, Anda dapat menemukan contoh yang melakukan ini menggunakan '*`custom_version`*' sebagai alias untuk versi.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

Anda juga dapat menyediakan pengualifikasi fungsi Lambda dengan detail nama fungsi sebagai berikut.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

Untuk informasi selengkapnya tentang `qualifier` dan parameter lainnya, lihat referensi aws_lambda.invoke.

## Pesan kesalahan fungsi Lambda

Dalam daftar berikut, Anda dapat menemukan informasi tentang pesan kesalahan, dengan kemungkinan penyebab dan solusi.

- VPCmasalah konfigurasi

  VPCmasalah konfigurasi dapat memunculkan pesan kesalahan berikut saat mencoba menghubungkan:

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  Penyebab umum untuk kesalahan ini adalah grup VPC keamanan yang tidak dikonfigurasi dengan benar. Pastikan Anda memiliki aturan keluar untuk TCP membuka port 443 grup VPC keamanan Anda sehingga Anda VPC dapat terhubung ke Lambda. VPC
- Kurangnya izin yang diperlukan untuk menginvokasi fungsi Lambda

  Jika Anda melihat salah satu pesan galat berikut, berarti pengguna (peran) yang menginvokasi fungsi ini tidak memiliki izin yang tepat.

  ```
  ERROR:  permission denied for schema aws_lambda
  ```

  ```
  ERROR:  permission denied for function invoke
  ```

  Pengguna (peran) harus diberi izin khusus untuk menginvokasi fungsi Lambda. Untuk informasi selengkapnya, lihat Langkah 6: Berikan pengguna lain izin untuk menginvokasi fungsi Lambda.
- Penanganan kesalahan yang tidak tepat dalam fungsi Lambda

Jika fungsi Lambda melempar pengecualian selama pemrosesan permintaan, `aws_lambda.invoke` gagal dengan SQL kesalahan Postgre seperti berikut ini.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

Pastikan untuk menangani kesalahan dalam fungsi Lambda Anda atau di aplikasi SQL Postgre Anda.

## AWS Lambdafungsi dan referensi parameter

Berikut ini adalah referensi untuk fungsi dan parameter yang akan digunakan untuk memanggil Lambda dengan Aurora .

Fungsi dan parameter

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- parameter aws_lambda

aws_lambda.invoke

Menjalankan fungsi Lambda untuk klaster DB Aurora PostgreSQL .

Untuk detail lebih lanjut tentang memanggil fungsi Lambda, lihat juga Invokasi di Panduan Developer AWS Lambda.

Sintaksis

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
```

```
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
```

```
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

Parameter input

function_name

Nama yang mengidentifikasi fungsi Lambda. Nilai tersebut dapat berupa nama fungsi, sebuah ARN, atau ARN parsial. Untuk daftar format yang memungkinkan, lihat Format nama fungsi Lambda dalam Panduan Developer AWS Lambda.

payload

Input untuk fungsi Lambda. Formatnya dapat berupa JSON atau JSONB. Untuk informasi selengkapnya, lihat Jenis JSON dalam dokumentasi PostgreSQL.

region

(Opsional) Wilayah Lambda untuk fungsi tersebut. Secara default, Aurora menyelesaikan Wilayah AWS dari ARN penuh di `function_name` atau menggunakan Wilayah instans DB Aurora PostgreSQL . Jika nilai Wilayah ini bertentangan dengan nilai yang disediakan dalam ARN `function_name`, pesan kesalahan akan muncul.

invocation_type

Jenis invokasi fungsi Lambda. Nilai ini peka huruf besar/kecil. Kemungkinan nilainya termasuk yang berikut ini:

- `RequestResponse` – Default. Jenis invokasi untuk fungsi Lambda bersifat sinkron dan menampilkan payload respons dalam hasilnya. Gunakan jenis invokasi `RequestResponse` ketika alur kerja Anda bergantung pada penerimaan hasil fungsi Lambda dengan segera.
- `Event` – Jenis invokasi untuk fungsi Lambda ini bersifat asinkron dan segera kembali tanpa menampilkan payload. Gunakan jenis invokasi `Event` ketika Anda tidak membutuhkan hasil dari fungsi Lambda sebelum alur kerja Anda berlanjut.
- `DryRun` – Jenis invokasi ini menguji akses tanpa menjalankan fungsi Lambda.

log_type

Jenis log Lambda untuk ditampilkan dalam parameter output `log_result`. Nilai ini peka huruf besar/kecil. Kemungkinan nilainya termasuk yang berikut ini:

- Ekor – Parameter output `log_result` yang ditampilkan akan mencakup 4 KB terakhir log eksekusi.
- Tidak Ada – Tidak ada informasi log Lambda yang ditampilkan.

context

Konteks klien dalam format JSON atau JSONB. Kolom yang akan digunakan termasuk `custom` dan `env`.

qualifier

Pengualifikasi yang mengidentifikasi versi fungsi Lambda yang akan diinvokasi. Jika nilai ini bertentangan dengan nilai yang disediakan dalam ARN `function_name`, pesan kesalahan akan muncul.

Parameter output

status_code

Kode respons status HTTP. Untuk informasi selengkapnya, lihat Elemen respons invokasi Lambda di Panduan Developer AWS Lambda.

payload

Informasi yang ditampilkan dari fungsi Lambda yang berjalan. Formatnya berupa JSON atau JSONB.

executed_version

Versi fungsi Lambda yang berjalan.

log_result

Informasi log eksekusi yang ditampilkan jika nilai `log_type` adalah `Tail` ketika fungsi Lambda diinvokasi. Hasilnya berisi 4 KB terakhir log eksekusi yang dikodekan dalam Base64.

aws_commons.create_lambda_function_arn

Membuat struktur `aws_commons._lambda_function_arn_1` untuk menyimpan informasi nama fungsi Lambda. Gunakan hasil fungsi `aws_commons.create_lambda_function_arn` dalam parameter `function_name` dari fungsi aws_lambda.invoke aws_lambda.invoke.

Sintaksis

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

Parameter input

function_name

String teks yang diperlukan berisi nama fungsi Lambda. Nilai tersebut dapat berupa nama fungsi, ARN penuh, atau ARN parsial.

region

String teks opsional yang berisi Wilayah AWS tempat fungsi Lambda berada. Untuk daftar nama Wilayah dan nilai terkait, lihat Wilayah dan Zona Ketersediaan.

parameter aws_lambda

Dalam tabel ini, Anda dapat menemukan parameter yang terkait dengan `aws_lambda` fungsi tersebut.

| Parameter | Deskripsi |
| --- | --- |
| `aws_lambda.connect_timeout_ms` | Ini adalah parameter dinamis dan menetapkan waktu tunggu maksimum saat menghubungkan ke AWS Lambda. Nilai defaultnya adalah`1000`. Nilai yang diizinkan untuk parameter ini adalah 1 - 900000. |
| `aws_lambda.request_timeout_ms` | Ini adalah parameter dinamis dan menetapkan waktu tunggu maksimum sambil menunggu respons dari AWS Lambda. Nilai defaultnya adalah`3000`. Nilai yang diizinkan untuk parameter ini adalah 1 - 900000. |
| `aws_lambda.endpoint_override` | Menentukan endpoint yang dapat digunakan untuk terhubung ke LambdaAWS. String kosong memilih titik akhir AWS Lambda default untuk wilayah tersebut. Anda harus me-restart database agar perubahan parameter statis ini berlaku. |

# Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch

Anda dapat mengonfigurasi klaster DB PostgreSQL Aurora untuk mengekspor data log ke Amazon Logs secara teratur. CloudWatch Ketika Anda melakukannya, peristiwa dari log PostgreSQL cluster PostgreSQL Aurora PostgreSQL DB Anda secara otomatis dipublikasikan ke Amazon, sebagai Amazon Logs. CloudWatch CloudWatch Di CloudWatch, Anda dapat menemukan data log yang diekspor dalam grup Log untuk cluster DB Aurora PostgreSQL Anda. Grup log berisi satu atau beberapa log stream yang berisi peristiwa dari log PostgreSQL dari setiap instans di klaster.

Menerbitkan CloudWatch log ke Log memungkinkan Anda menyimpan catatan log PostgreSQL klaster Anda dalam penyimpanan yang sangat tahan lama. Dengan data log yang tersedia di CloudWatch Log, Anda dapat mengevaluasi dan meningkatkan operasi klaster Anda. Anda juga dapat menggunakan CloudWatch untuk membuat alarm dan melihat metrik. Untuk mempelajari selengkapnya, lihat Memantau peristiwa log di Amazon CloudWatch.

> **Note**
>
> Menerbitkan log PostgreSQL Anda CloudWatch ke Log menghabiskan penyimpanan, dan Anda dikenakan biaya untuk penyimpanan itu. Pastikan untuk menghapus CloudWatch Log apa pun yang tidak lagi Anda butuhkan.

Menonaktifkan opsi log ekspor untuk klaster Aurora PostgreSQL DB yang ada tidak memengaruhi data apa pun yang sudah disimpan di Log. CloudWatch Log yang ada tetap tersedia di CloudWatch Log berdasarkan pengaturan penyimpanan log Anda. Untuk mempelajari lebih lanjut tentang CloudWatch Log, lihat Apa itu CloudWatch Log Amazon?

Aurora PostgreSQL mendukung penerbitan log ke Log untuk versi berikut. CloudWatch

- 16.1 dan semua versi yang lebih tinggi
- 15.2 dan versi 15 yang lebih tinggi
- 14.3 dan versi 14 yang lebih tinggi
- 13.3 dan 13 versi yang lebih tinggi
- 12.8 dan versi 12 yang lebih tinggi
- 11.9 dan versi 11 yang lebih tinggi

Untuk informasi tentang mengaktifkan opsi untuk memublikasikan CloudWatch log ke Log, memantau peristiwa CloudWatch log di Log, dan menganalisis CloudWatch log menggunakan Wawasan Log, lihat topik berikut.

Topik

- Mengaktifkan opsi untuk menerbitkan log ke Amazon CloudWatch
- Memantau peristiwa log di Amazon CloudWatch
- Menganalisis log PostgreSQL menggunakan Wawasan Log CloudWatch

## Mengaktifkan opsi untuk menerbitkan log ke Amazon CloudWatch

Untuk mempublikasikan log PostgreSQL klaster Aurora PostgreSQL DB Anda ke Log, pilih opsi ekspor Log untuk klaster. CloudWatch  Anda dapat memilih pengaturan Ekspor log saat Anda membuat klaster DB Aurora PostgreSQL. Atau, Anda dapat mengubah klaster ini nanti. Saat Anda memodifikasi klaster yang ada, log PostgreSQL dari setiap instance dipublikasikan CloudWatch ke cluster sejak saat itu. Untuk Aurora PostgreSQL, PostgreSQL log () adalah satu-satunya log yang dipublikasikan ke Amazon. `postgresql.log` CloudWatch

Anda dapat menggunakan, API AWS Management Console AWS CLI, atau RDS untuk mengaktifkan fitur ekspor Log untuk klaster Aurora PostgreSQL DB Anda.

Konsol

Anda memilih opsi ekspor Log untuk mulai menerbitkan log PostgreSQL dari cluster Aurora PostgreSQL DB Anda ke Log. CloudWatch

Untuk mengaktifkan fitur Ekspor log dari konsol

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB PostgreSQL Aurora yang data lognya ingin Anda publikasikan ke Log. CloudWatch
4. Pilih Modifikasi.
5. Di bagian Ekspor log, pilih Log PostgreSQL.
6. Pilih Lanjutkan, lalu pilih Ubah klaster di halaman ringkasan.

AWS CLI

Anda dapat mengaktifkan opsi ekspor log untuk mulai menerbitkan log Aurora PostgreSQL ke Amazon Logs dengan file. CloudWatch AWS CLI Untuk melakukannya, jalankan modify-db-cluster AWS CLI perintah dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--cloudwatch-logs-export-configuration`—Pengaturan konfigurasi untuk jenis log yang akan disetel untuk diekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora PostgreSQL dengan menjalankan salah satu dari perintah AWS CLI berikut:

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Jalankan salah satu AWS CLI perintah ini dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--engine` – Mesin basis data.
- `--enable-cloudwatch-logs-exports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Opsi lain mungkin diperlukan tergantung pada AWS CLI perintah yang Anda jalankan.

Example

Perintah berikut membuat cluster Aurora PostgreSQL DB untuk mempublikasikan file log ke Log. CloudWatch

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
```

```
    --enable-cloudwatch-logs-exports postgresql
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
    --enable-cloudwatch-logs-exports postgresql
```

Example

Perintah berikut memodifikasi cluster Aurora PostgreSQL DB yang ada untuk mempublikasikan file log ke Log. CloudWatch Nilai `--cloudwatch-logs-export-configuration` adalah objek JSON. Kunci untuk objek ini adalah`EnableLogTypes`, dan nilainya adalah`postgresql`, dan`instance`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["postgresql","instance"]}'
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\",
\"instance\"]}'
```

> **Note**
>
> Saat menggunakan command prompt Windows, pastikan untuk meng-escape tanda kutip ganda (“) dalam kode JSON dengan memberikan garis miring terbalik (\) di depannya.

Example

Contoh berikut memodifikasi cluster Aurora PostgreSQL DB yang ada untuk menonaktifkan penerbitan file log ke Log. CloudWatch Nilai `--cloudwatch-logs-export-configuration`

adalah objek JSON. Kunci untuk objek ini adalah`DisableLogTypes`, dan nilainya adalah `postgresql` dan`instance`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":
["postgresql","instance"]}'
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\",
\"instance\"]}"
```

> **Note**
>
> Saat menggunakan command prompt Windows, Anda harus meng-escape tanda kutip ganda (") dalam kode JSON dengan memberikan garis miring terbalik (\) di depannya.

API RDS

Anda dapat mengaktifkan opsi ekspor log untuk mulai menerbitkan log Aurora PostgreSQL dengan API RDS. Untuk melakukannya, jalankan DBCluster operasi Modify dengan opsi berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `CloudwatchLogsExportConfiguration`— Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk ekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora PostgreSQL dengan API RDS dengan menjalankan salah satu dari operasi API RDS berikut:

- Buat DBCluster
- Kembalikan DBCluster dari3
- MemulihkanDBClusterFromSnapshot

- MemulihkanDBClusterToPointInTime

Jalankan tindakan API RDS dengan parameter berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `Engine` – Mesin basis data.
- `EnableCloudwatchLogsExports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Parameter lain mungkin diperlukan tergantung pada AWS CLI perintah yang Anda jalankan.

## Memantau peristiwa log di Amazon CloudWatch

Dengan peristiwa SQL log Aurora Postgre yang diterbitkan dan tersedia sebagai Amazon CloudWatch Logs, Anda dapat melihat dan memantau peristiwa menggunakan Amazon. CloudWatch Untuk informasi selengkapnya tentang pemantauan, lihat Melihat data log yang dikirim ke CloudWatch Log.

Saat Anda mengaktifkan ekspor Log, grup log baru secara otomatis dibuat menggunakan awalan `/aws/rds/cluster/` dengan nama Aurora Postgre Anda SQL dan jenis log, seperti pada pola berikut.

```
/aws/rds/cluster/your-cluster-name/postgresql
```

Sebagai contoh, misalkan cluster Aurora Postgre SQL DB bernama `docs-lab-apg-small` mengekspor lognya ke Amazon Logs. CloudWatch Nama grup lognya di Amazon CloudWatch ditampilkan sebagai berikut.

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

Jika ada grup log yang sudah memiliki nama yang ditentukan, Aurora akan menggunakan grup log tersebut untuk mengekspor data log untuk klaster DB Aurora. Setiap instans DB di cluster Aurora Postgre SQL DB mengunggah log Postgre ke grup log sebagai SQL aliran log yang berbeda. Anda dapat memeriksa grup log dan aliran lognya menggunakan berbagai alat grafis dan analitis yang tersedia di Amazon CloudWatch.

Misalnya, Anda dapat mencari informasi dalam peristiwa log dari klaster SQL DB Aurora Postgre Anda, dan memfilter peristiwa dengan menggunakan konsol Log, CloudWatch AWS CLI, atau

CloudWatch LogAPI. Untuk informasi selengkapnya, Mencari dan memfilter data log di Panduan Pengguna Amazon CloudWatch Logs.

Secara default, grup log baru dibuat menggunakan Tidak pernah kedaluwarsa untuk periode retensinya. Anda dapat menggunakan konsol CloudWatch Log, AWS CLI, atau CloudWatch Log API untuk mengubah periode retensi log. Untuk mempelajari selengkapnya, lihat Mengubah penyimpanan data CloudWatch log di Log di Panduan Pengguna CloudWatch Log Amazon.

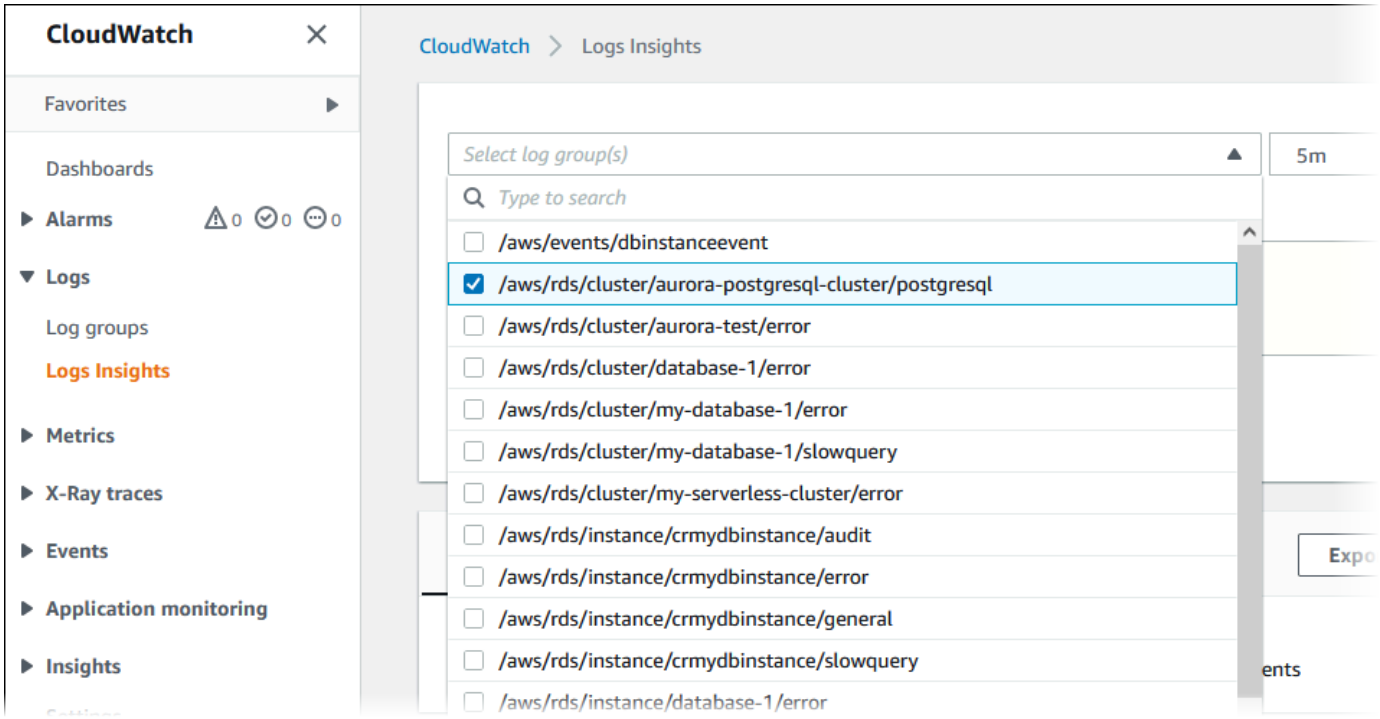
> **Tip**
>
> Anda dapat menggunakan konfigurasi otomatis, seperti AWS CloudFormation, untuk membuat grup log dengan periode penyimpanan log yang telah ditentukan sebelumnya, filter metrik, dan izin akses.

## Menganalisis log PostgreSQL menggunakan Wawasan Log CloudWatch

Dengan log PostgreSQL dari klaster DB PostgreSQL Aurora yang CloudWatch diterbitkan sebagai Log, Anda dapat CloudWatch menggunakan Wawasan Log untuk mencari dan menganalisis data log Anda secara interaktif di Log Amazon. CloudWatch CloudWatch Wawasan Log mencakup bahasa kueri, kueri sampel, dan alat lain untuk menganalisis data log sehingga Anda dapat mengidentifikasi potensi masalah dan memverifikasi perbaikan. Untuk mempelajari selengkapnya, lihat Menganalisis data CloudWatch log dengan Wawasan Log di Panduan Pengguna CloudWatch Log Amazon.

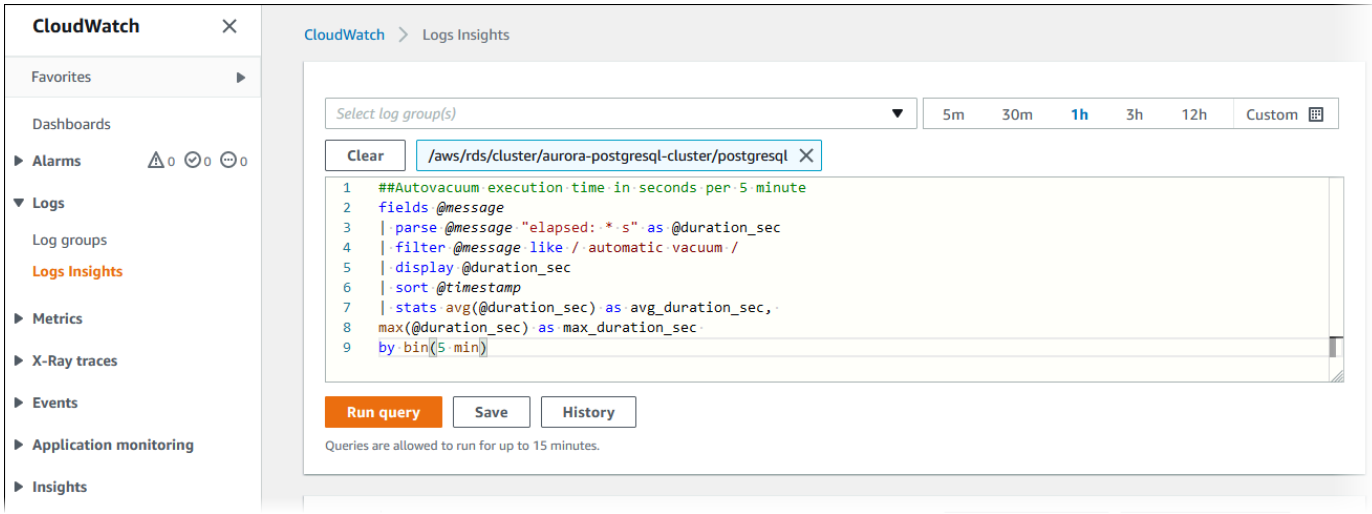
Untuk menganalisis log PostgreSQL dengan Wawasan Log CloudWatch

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Di panel navigasi, buka Log dan pilih Wawasan log.
3. Di opsi Pilih grup log, pilih grup log untuk klaster DB Aurora PostgreSQL Anda.

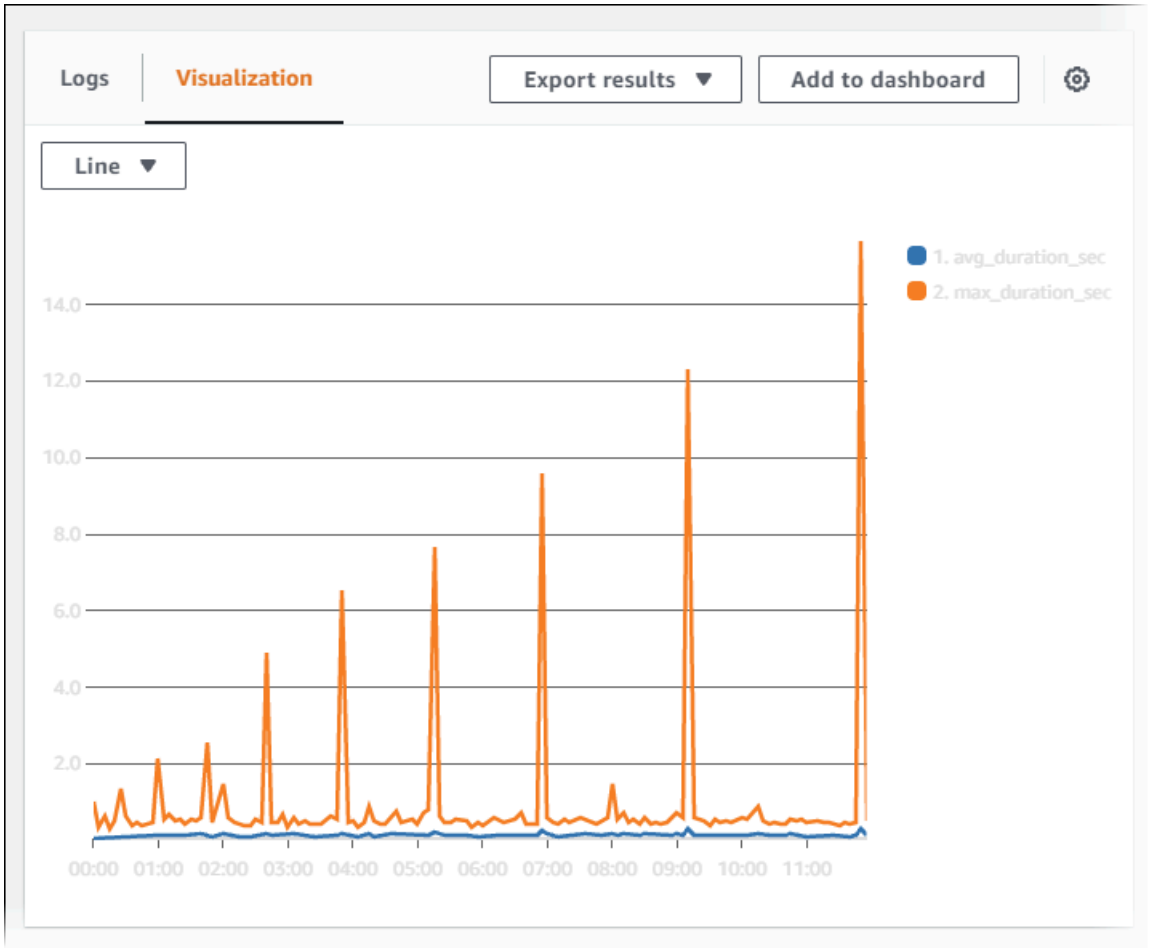



4. Di editor kueri, hapus kueri yang saat ini ditampilkan, masukkan yang berikut ini, lalu pilih Jalankan kueri.

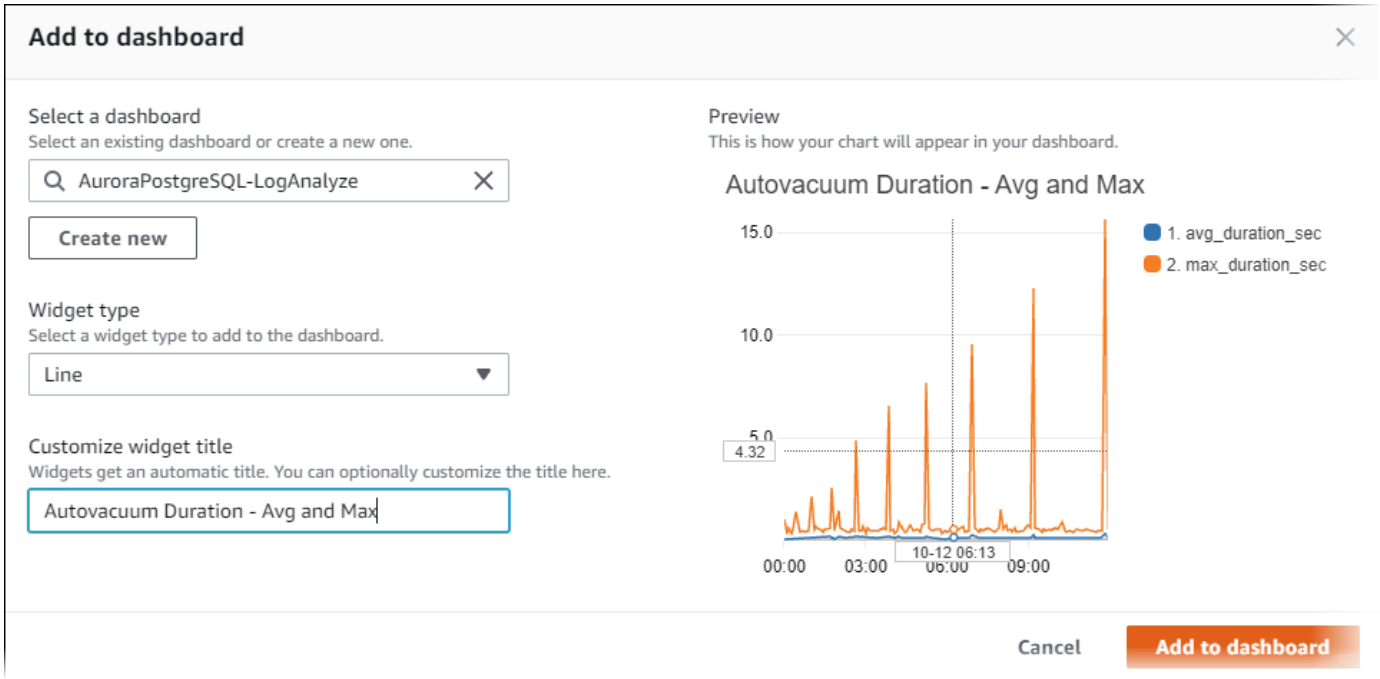
```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```




5. Pilih tab Visualisasi.



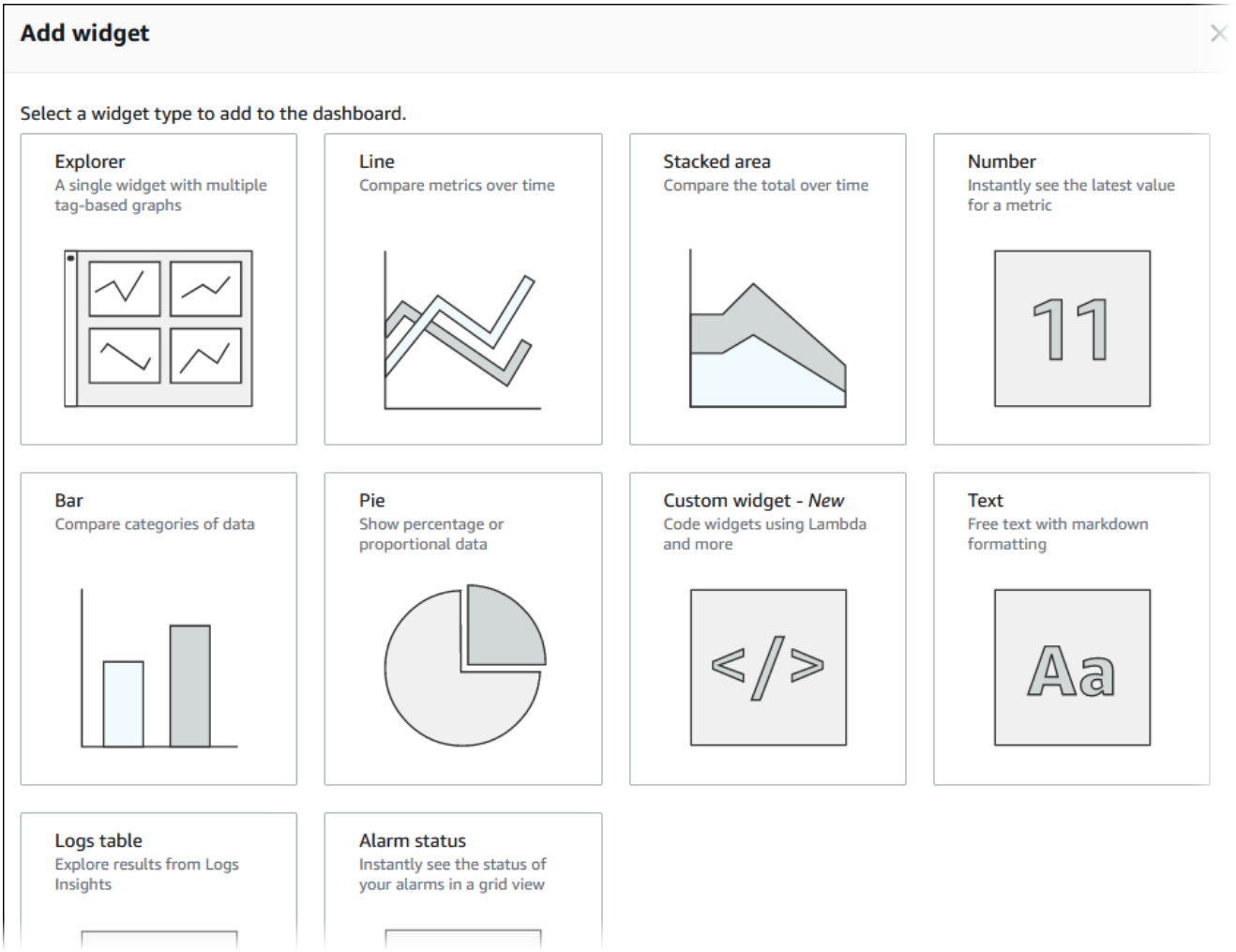

6. Pilih Tambahkan ke dasbor.
7. Di Pilih dasbor, pilih dasbor atau masukkan nama untuk membuat dasbor baru.
8. Dalam Jenis widget, pilih jenis widget untuk visualisasi Anda.



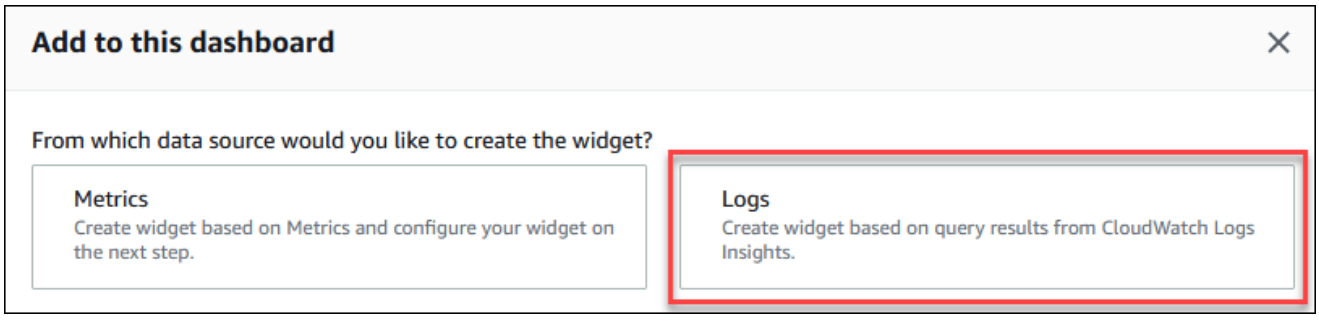

9. (Opsional) Tambahkan lebih banyak widget berdasarkan hasil kueri log Anda.

   a. Pilih Tambahkan widget.

   b. Pilih jenis widget, seperti Baris.




c. Di jendela Tambahkan ke dasbor ini, pilih Log.

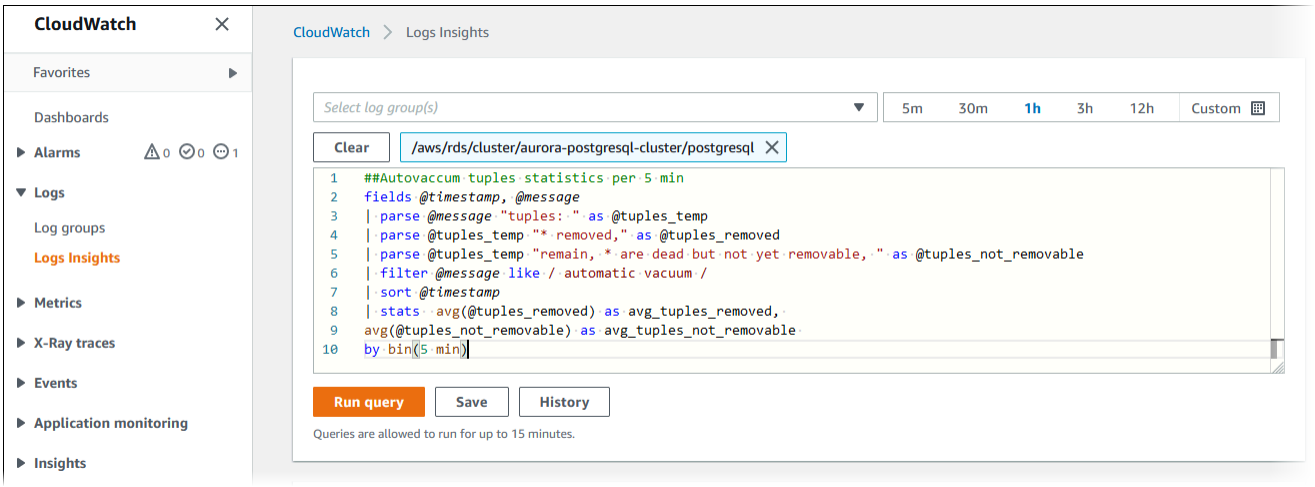



d. Di opsi Pilih grup log, pilih grup log untuk klaster DB Anda.

e. Di editor kueri, hapus kueri yang saat ini ditampilkan, masukkan yang berikut ini, lalu pilih Jalankan kueri.

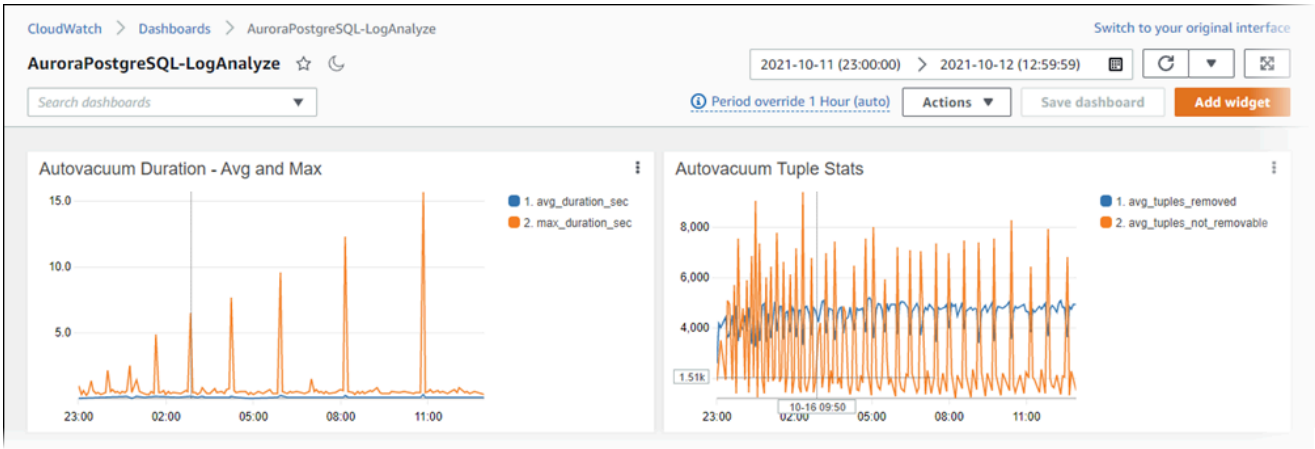
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
```

```
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```




f. Pilih Buat widget.

Dasbor Anda akan terlihat seperti gambar berikut.



# Memantau rencana eksekusi kueri dan memori puncak untuk Aurora Postgre SQL

Anda dapat memantau rencana eksekusi kueri di instans Aurora Postgre SQL DB Anda untuk mendeteksi rencana eksekusi yang berkontribusi pada pemuatan basis data saat ini dan untuk melacak statistik kinerja rencana eksekusi dari waktu ke waktu menggunakan parameter. `aurora_compute_plan_id` Setiap kali kueri dijalankan, rencana eksekusi yang digunakan oleh kueri diberi pengenal dan pengidentifikasi yang sama digunakan oleh eksekusi berikutnya dari rencana yang sama.

`aurora_compute_plan_id`Ini diubah secara `OFF` default di grup parameter DB dari Aurora Postgre SQL versi 14.10, 15.5, dan versi yang lebih tinggi. Untuk menetapkan pengidentifikasi rencana, atur `aurora_compute_plan_id` ke `ON` dalam grup parameter.

Pengidentifikasi rencana ini digunakan dalam beberapa utilitas yang melayani tujuan yang berbeda.

Anda dapat memantau penggunaan memori puncak kueri dalam instans DB Anda untuk mendeteksi kueri yang berkontribusi pada penggunaan memori database yang tinggi dari versi berikut:

- 16.3 dan semua versi yang lebih tinggi
- 15.7 dan versi yang lebih tinggi
- 14.12 dan versi yang lebih tinggi

Setiap kali kueri berjalan, memori puncak yang digunakan oleh kueri dilacak. Kueri biasanya berjalan berkali-kali; nilai penggunaan memori rata-rata, minimum dan maksimum di semua proses dapat dilihat untuk setiap kueri.

Topik

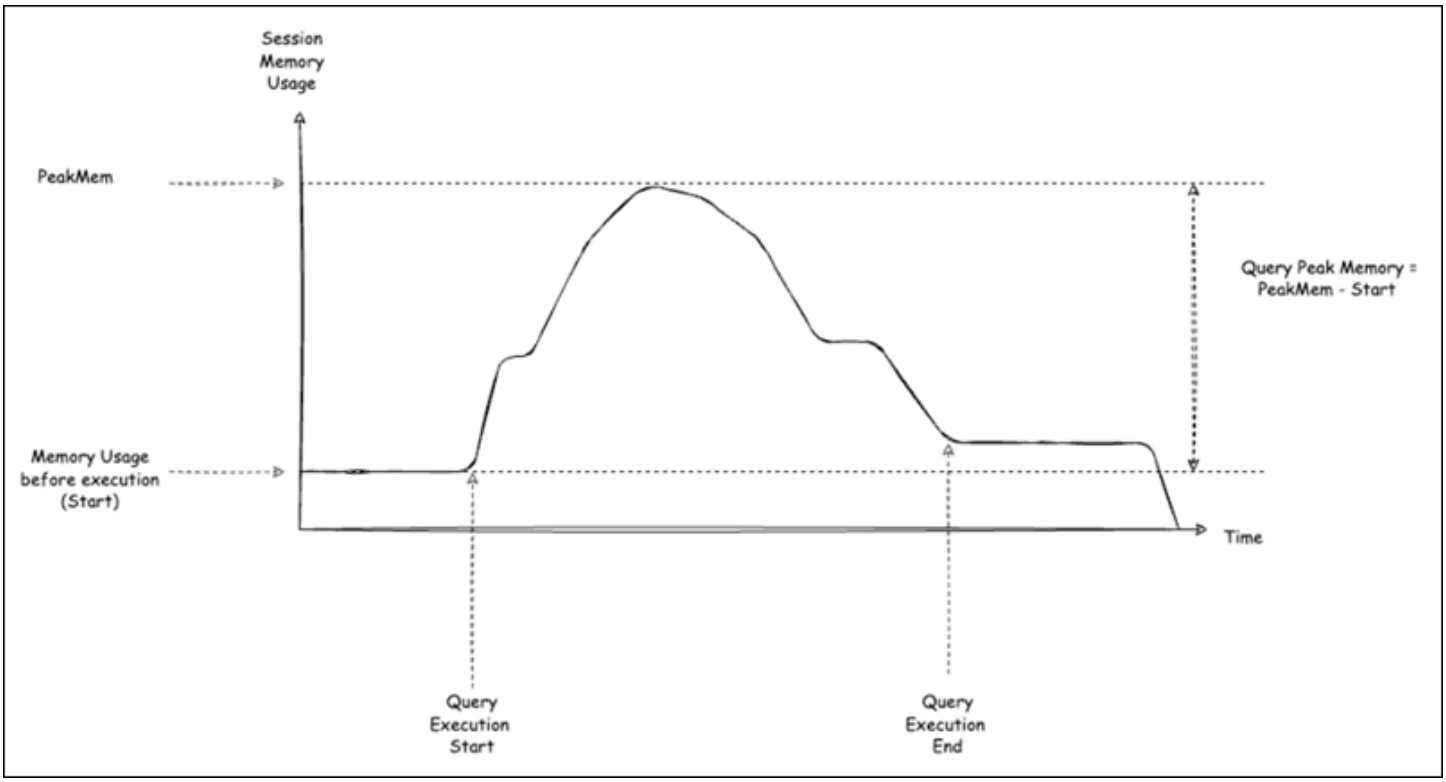

- Mengakses rencana eksekusi kueri dan memori puncak menggunakan fungsi Aurora
- Referensi parameter untuk rencana eksekusi kueri Aurora Postgre SQL


## Mengakses rencana eksekusi kueri dan memori puncak menggunakan fungsi Aurora

Dengan`aurora_compute_plan_id`, Anda dapat mengakses rencana eksekusi menggunakan fungsi-fungsi berikut:

- aurora_stat_activity
- aurora_stat_plans

Memori puncak kueri tidak termasuk memori yang dialokasikan sebelum pemrosesan kueri dimulai. Penggunaan memori puncak dilacak dan dilaporkan secara terpisah untuk fase perencanaan dan eksekusi setiap kueri.




Anda dapat mengakses statistik memori puncak kueri menggunakan fungsi-fungsi berikut:

- aurora_stat_statements
- aurora_stat_plans

Untuk informasi lebih lanjut tentang fungsi-fungsi ini, lihatReferensi fungsi Aurora PostgreSQL.

## Referensi parameter untuk rencana eksekusi kueri Aurora Postgre SQL

Anda dapat memantau rencana eksekusi kueri menggunakan parameter di bawah ini dalam grup parameter DB.

Parameter

- aurora_compute_plan_id
- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs
- aurora_stat_plans.with_analysis
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers
- aurora_stat_plans.with_wal
- aurora_stat_plans.with_trigger

> **Note**
>
> Konfigurasi untuk `aurora_stat_plans.with_*` parameter hanya berlaku untuk rencana yang baru ditangkap.

## aurora_compute_plan_id

`aurora_compute_plan_id`Ini adalah parameter konfigurasi yang mengontrol apakah pengidentifikasi rencana ditetapkan selama eksekusi kueri.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| off | 0 (nonaktif) | Setel `off` untuk mencegah pengenal rencana ditetapkan. |
| | 1 (aktif) | Setel on untuk menetapkan pengenal rencana. |

## aurora_stat_plans.minutes_until_recapture

Jumlah menit yang harus dilewati sebelum rencana direbut kembali. Defaultnya adalah 0 yang akan menonaktifkan pengambilan kembali rencana. Ketika `aurora_stat_plans.calls_until_recapture` ambang batas dilewati, rencana akan direbut kembali.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| 0 | 0-1073741823 | Atur jumlah menit yang harus dilewati sebelum rencana direbut kembali. |

### aurora_stat_plans.calls_until_recapture

Jumlah panggilan ke rencana sebelum ditangkap kembali. Defaultnya adalah 0 yang akan menonaktifkan pengambilan kembali paket setelah sejumlah panggilan. Ketika `aurora_stat_plans.minutes_until_recapture` ambang batas dilewati, rencana akan direbut kembali.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| 0 | 0-1073741823 | Tetapkan jumlah panggilan sebelum rencana ditangkap kembali. |

### aurora_stat_plans.with_costs

Menangkap EXPLAIN rencana dengan perkiraan biaya. Nilai yang diizinkan adalah on dan`off`. Default-nya adalah on.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| on | 0 (nonaktif) | Tidak menunjukkan perkiraan biaya dan baris untuk setiap node paket. |
| | 1 (aktif) | Menunjukkan perkiraan biaya dan baris untuk setiap node paket. |

### aurora_stat_plans.with_analysis

Mengontrol EXPLAIN rencana denganANALYZE. Mode ini hanya digunakan saat pertama kali rencana ditangkap. Nilai yang diizinkan adalah on dan`off`. Default-nya adalah `off`.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| off | 0 (nonaktif) | Tidak termasuk statistik waktu berjalan aktual untuk rencana tersebut. |
| | 1 (aktif) | Termasuk statistik waktu berjalan aktual untuk rencana tersebut. |

### aurora_stat_plans.with_timing

Waktu rencana akan ditangkap dalam penjelasan kapan ANALYZE digunakan. Default-nya adalah on.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| on | 0 (nonaktif) | Tidak termasuk waktu start up aktual dan waktu yang dihabiskan di setiap node rencana. |
| | 1 (aktif) | Termasuk waktu start up aktual dan waktu yang dihabiskan di setiap node rencana. |

### aurora_stat_plans.with_buffers

Statistik penggunaan buffer rencana akan ditangkap dalam penjelasan saat ANALYZE digunakan. Default-nya adalah `off`.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| off | 0 (nonaktif) | Tidak termasuk informasi tentang penggunaan buffer. |
| | 1 (aktif) | Termasuk informasi tentang penggunaan buffer. |

### aurora_stat_plans.with_wal

Statistik penggunaan plan wal akan ditangkap dalam penjelasan kapan ANALYZE digunakan. Default-nya adalah `off`.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| off | 0 (nonaktif) | Tidak termasuk informasi tentang pembuatan WAL rekaman. |
| | 1 (aktif) | Termasuk informasi tentang pembuatan WAL rekaman. |

## aurora_stat_plans.with_trigger

Statistik eksekusi pemicu rencana akan ditangkap dalam penjelasan saat `ANALYZE` digunakan. Default-nya adalah `off`.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| off | 0 (nonaktif) | Tidak termasuk statistik eksekusi pemicu. |
| | 1 (aktif) | Termasuk statistik eksekusi pemicu. |

# Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL

Manajemen rencana kueri Aurora PostgreSQL adalah fitur opsional yang dapat Anda gunakan dengan klaster DB Amazon Aurora Edisi Kompatibel PostgreSQL. Fitur ini dikemas sebagai ekstensi `apg_plan_mgmt` yang dapat Anda instal di klaster DB Aurora PostgreSQL Anda. Manajemen rencana kueri memungkinkan Anda mengelola rencana eksekusi kueri yang dihasilkan oleh pengoptimisasi untuk aplikasi SQL Anda. `apg_plan_mgmt` AWS Ekstensi dibangun di atas fungsionalitas pemrosesan kueri asli dari mesin database PostgreSQL.

Di bagian berikut ini, Anda dapat menemukan informasi tentang fitur manajemen rencana kueri Aurora PostgreSQL, cara mengaturnya, dan cara menggunakannya dengan klaster DB Aurora PostgreSQL Anda. Sebelum memulai, sebaiknya Anda meninjau catatan rilis apa pun untuk versi spesifik ekstensi `apg_plan_mgmt` yang tersedia untuk versi Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Versi ekstensi apg_plan_mgmt Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Topik


- Gambaran umum manajemen rencana kueri Aurora PostgreSQL
- Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL
- Manajemen rencana kueri Aurora PostgreSQL
- Mengambil rencana eksekusi Aurora PostgreSQL
- Menggunakan rencana terkelola Aurora PostgreSQL
- Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans
- Meningkatkan rencana kueri PostgreSQL Aurora
- Menghapus paket kueri Aurora PostgreSQL
- Mengekspor dan mengimpor rencana terkelola untuk Aurora PostgreSQL
- Referensi parameter untuk manajemen rencana kueri Aurora PostgreSQL
- Referensi fungsi untuk manajemen rencana kueri Aurora PostgreSQL
- Referensi untuk tampilan apg_plan_mgmt.dba_plans untuk Aurora PostgreSQL Compatible Edition
- Fitur lanjutan dalam Manajemen Rencana Kueri

# Gambaran umum manajemen rencana kueri Aurora PostgreSQL

Manajemen rencana kueri Aurora PostgreSQL dirancang untuk memastikan stabilitas rencana terlepas dari perubahan pada basis data yang dapat menyebabkan regresi rencana kueri. Regresi rencana kueri terjadi ketika pengoptimal memilih rencana yang kurang optimal untuk pernyataan SQL yang diberikan setelah perubahan sistem atau basis data. Perubahan pada statistik, pembatasan, pengaturan lingkungan, pengikatan parameter kueri, dan peningkatan ke mesin basis data PostgreSQL semuanya dapat menyebabkan regresi rencana.

Dengan manajemen rencana kueri Aurora PostgreSQL, Anda dapat mengendalikan bagaimana dan kapan rencana eksekusi kueri akan berubah. Manfaat manajemen rencana kueri Aurora PostgreSQL meliputi yang berikut ini.

- Meningkatkan stabilitas rencana dengan memaksa pengoptimal untuk memilih dari sejumlah kecil rencana yang sudah diketahui bagus.
- Mengoptimalkan rencana secara terpusat lalu mendistribusikan rencana terbaik secara global.
- Mengidentifikasi indeks yang tidak digunakan dan menilai dampak dari membuat atau menghapus indeks.
- Secara otomatis mendeteksi rencana biaya minimum baru yang ditemukan oleh pengoptimal.
- Mencoba fitur-fitur pengoptimal baru dengan risiko lebih rendah karena Anda dapat memilih untuk hanya menyetujui perubahan rencana yang meningkatkan performa.

Anda dapat menggunakan alat yang disediakan oleh manajemen rencana kueri secara proaktif untuk menentukan rencana terbaik untuk kueri tertentu. Atau, Anda dapat menggunakan manajemen rencana kueri untuk bereaksi terhadap perubahan keadaan dan menghindari regresi rencana. Untuk informasi selengkapnya, lihat Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL.

Topik


- Pernyataan SQL yang didukung
- Batasan manajemen rencana kueri
- Terminologi manajemen rencana kueri
- Versi manajemen rencana kueri Aurora PostgreSQL
- Mengaktifkan manajemen rencana kueri Aurora PostgreSQL
- Meningkatkan manajemen rencana kueri Aurora PostgreSQL
- Menonaktifkan manajemen rencana kueri Aurora PostgreSQL

## Pernyataan SQL yang didukung

Manajemen rencana kueri mendukung jenis pernyataan SQL berikut.

- Pernyataan SELECT, INSERT, UPDATE, atau DELETE, terlepas dari kompleksitasnya.
- Pernyataan yang disiapkan. Untuk informasi selengkapnya, lihat PREPARE dalam dokumentasi PostgreSQL.
- Pernyataan dinamis, termasuk yang dijalankan dalam mode segera. Untuk informasi selengkapnya, lihat Dynamic SQL dan EXECUTE IMMEDIATE dalam dokumentasi PostgreSQL.
- Perintah dan pernyataan SQL tersemat. Untuk informasi selengkapnya, lihat Embedded SQL Commands dalam dokumentasi PostgreSQL.
- Pernyataan di dalam fungsi bernama. Untuk informasi selengkapnya, lihat CREATE FUNCTION dalam dokumentasi PostgreSQL.
- Pernyataan yang berisi tabel temp.
- Pernyataan di dalam prosedur dan DO-block.

Anda dapat menggunakan manajemen rencana kueri dengan `EXPLAIN` dalam mode manual untuk menangkap rencana tanpa benar-benar menjalankannya. Untuk informasi selengkapnya, lihat Menganalisis rencana yang dipilih pengoptimisasi. Untuk mempelajari selengkapnya tentang mode manajemen rencana kueri (manual, otomatis), lihat Mengambil rencana eksekusi Aurora PostgreSQL.

Manajemen rencana kueri Aurora PostgreSQL mendukung semua fitur bahasa PostgreSQL, termasuk tabel yang dipartisi, pewarisan, keamanan tingkat baris, dan ekspresi tabel umum rekursif (). CTEs Untuk mempelajari selengkapnya tentang fitur bahasa PostgreSQL ini, lihat Table Partitioning, Row Security Policies, dan WITH Queries (Common Table Expressions) serta topik lainnya dalam dokumentasi PostgreSQL.

Untuk informasi tentang berbagai versi fitur manajemen rencana kueri Aurora PostgreSQL, lihat Aurora PostgreSQL versi ekstensi apg_plan_mgmt dalam Catatan Rilis untuk Aurora PostgreSQL.

## Batasan manajemen rencana kueri

Rilis manajemen rencana kueri Aurora PostgreSQL saat ini memiliki batasan berikut.

- Rencana tidak ditangkap untuk pernyataan yang mereferensikan relasi sistem – Pernyataan yang mereferensikan relasi sistem, seperti `pg_class`, tidak ditangkap. Hal ini berlaku berdasarkan

desain untuk mencegah sejumlah besar rencana yang dihasilkan sistem yang digunakan secara internal agar tidak ditangkap. Hal ini juga berlaku untuk tabel sistem dalam tampilan.

- Kelas instans DB yang lebih besar mungkin diperlukan untuk cluster Aurora PostgreSQL DB Anda — Bergantung pada beban kerja, manajemen rencana kueri mungkin memerlukan kelas instans DB yang memiliki lebih dari 2 v. CPUs Jumlah `max_worker_processes` dibatasi oleh ukuran kelas instans DB. Jumlah `max_worker_processes` yang disediakan oleh kelas instans DB 2-vCPU (db.t3.medium, misalnya) mungkin tidak cukup untuk beban kerja tertentu. Kami menyarankan Anda memilih kelas instans DB dengan lebih dari 2 v CPUs untuk klaster Aurora PostgreSQL DB Anda jika Anda menggunakan manajemen rencana kueri.

  Ketika kelas instans DB tidak dapat mendukung beban kerja, manajemen rencana kueri memunculkan pesan kesalahan seperti berikut ini.

  ```
  WARNING: could not register plan insert background process
  HINT: You may need to increase max_worker_processes.
  ```

  Dalam hal ini, Anda harus menaikkan skala klaster DB Aurora PostgreSQL Anda ke ukuran kelas instans DB dengan lebih banyak memori. Untuk informasi selengkapnya, lihat Mesin DB yang didukung untuk kelas instans DB.
- Rencana yang sudah disimpan dalam sesi tidak terpengaruh – Manajemen rencana kueri menyediakan cara untuk memengaruhi rencana kueri tanpa mengubah kode aplikasi. Namun, ketika rencana generik sudah disimpan dalam sesi yang ada dan jika Anda ingin mengubah rencana kueri, maka Anda harus terlebih dahulu mengatur `plan_cache_mode` ke `force_custom_plan` dalam grup parameter klaster DB.
- `queryid` di `apg_plan_mgmt.dba_plans` dan `pg_stat_statements` dapat menyimpang ketika:
  - Objek dihapus dan dibuat ulang setelah disimpan dalam apg_plan_mgmt.dba_plans.
  - Tabel `apg_plan_mgmt.plans` diimpor dari klaster lain.

Untuk informasi tentang berbagai versi fitur manajemen rencana kueri Aurora PostgreSQL, lihat Aurora PostgreSQL versi ekstensi apg_plan_mgmt dalam Catatan Rilis untuk Aurora PostgreSQL.

## Terminologi manajemen rencana kueri

Istilah berikut digunakan di seluruh topik ini:

pernyataan terkelola

Pernyataan SQL yang ditangkap oleh pengoptimal dalam manajemen rencana kueri. Pernyataan terkelola memiliki satu atau beberapa rencana eksekusi kueri yang disimpan dalam tampilan `apg_plan_mgmt.dba_plans`.

acuan dasar rencana

Kumpulan rencana yang disetujui untuk pernyataan terkelola tertentu. Artinya, semua rencana untuk pernyataan terkelola yang telah "Disetujui" untuk kolom `status`-nya dalam tampilan `dba_plan`.

riwayat rencana

Kumpulan semua rencana yang ditangkap untuk suatu pernyataan terkelola tertentu. Riwayat rencana berisi semua rencana yang ditangkap untuk pernyataan tersebut, terlepas dari statusnya.

regresi rencana kueri

Kasus ketika pengoptimal memilih rencana yang kurang optimal dibandingkan dengan sebelum perubahan tertentu dilakukan pada lingkungan basis data, seperti versi PostgreSQL baru atau perubahan statistik.

## Versi manajemen rencana kueri Aurora PostgreSQL

Manajemen rencana kueri didukung oleh semua rilis Aurora PostgreSQL yang tersedia saat ini. Untuk informasi selengkapnya, lihat Daftar pembaruan Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Fungsionalitas manajemen rencana kueri ditambahkan ke klaster DB Aurora PostgreSQL Anda saat Anda menginstal ekstensi `apg_plan_mgmt`. Berbagai versi Aurora PostgreSQL mendukung versi ekstensi `apg_plan_mgmt` yang berbeda-beda. Kami menyarankan agar Anda meningkatkan ekstensi manajemen rencana kueri ke rilis terbaru untuk versi Aurora PostgreSQL Anda.

> **Note**
>
> Untuk catatan rilis untuk setiap versi ekstensi `apg_plan_mgmt`, lihat Aurora PostgreSQL versi ekstensi apg_plan_mgmt dalam Catatan Rilis untuk Aurora PostgreSQL.

Anda dapat mengidentifikasi versi yang berjalan di klaster Anda dengan menghubungkan ke instans menggunakan `psql` lalu menggunakan perintah meta \dx untuk menampilkan daftar ekstensi seperti yang ditunjukkan berikut.

```
labdb=> \dx
                       List of installed extensions
     Name      | Version |    Schema     |                            Description
---------------+---------+---------------
+------------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

Output menunjukkan bahwa klaster ini menggunakan ekstensi versi 1.0. Hanya versi `apg_plan_mgmt` tertentu yang tersedia untuk versi Aurora PostgreSQL tertentu. Dalam beberapa kasus, Anda mungkin perlu meningkatkan klaster DB Aurora PostgreSQL ke rilis minor baru atau menerapkan patch sehingga Anda dapat meningkatkan ke manajemen rencana kueri versi terbaru. `apg_plan_mgmt` versi 1.0 yang ditampilkan dalam output berasal dari klaster DB Aurora PostgreSQL versi 10.17, yang tidak memiliki `apg_plan_mgmt` versi lebih baru. Dalam hal ini, klaster DB Aurora PostgreSQL harus ditingkatkan ke versi PostgreSQL yang lebih baru.

Untuk informasi selengkapnya tentang meningkatkan klaster DB Aurora PostgreSQL Anda ke PostgreSQL versi baru, lihat Pembaruan mesin database untuk Amazon Aurora Postgre SQL.

Untuk mempelajari cara meningkatkan ekstensi `apg_plan_mgmt`, lihat Meningkatkan manajemen rencana kueri Aurora PostgreSQL.

## Mengaktifkan manajemen rencana kueri Aurora PostgreSQL

Penyiapan manajemen rencana kueri untuk klaster DB Aurora PostgreSQL Anda memerlukan penginstalan ekstensi dan perubahan beberapa pengaturan parameter klaster DB. Anda memerlukan izin `rds_superuser` untuk menginstal ekstensi `apg_plan_mgmt` dan mengaktifkan fitur untuk klaster DB Aurora PostgreSQL.

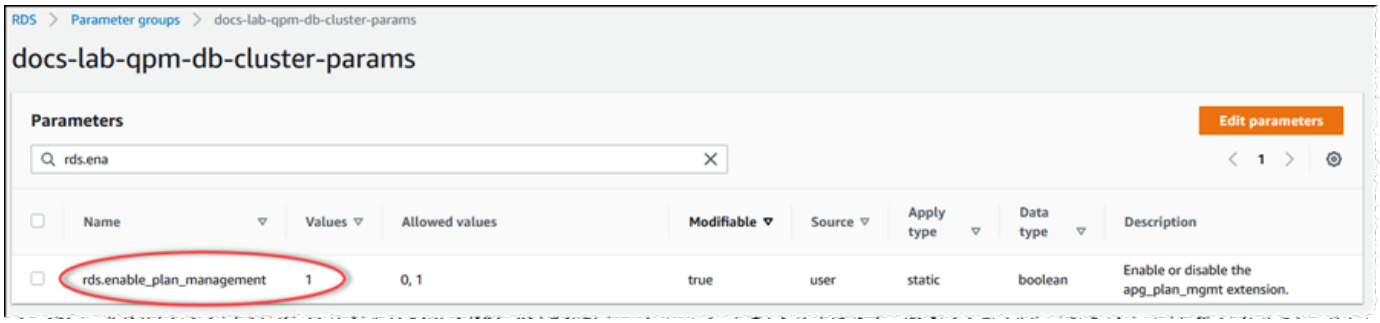
Saat ekstensi ini diinstal, sebuah peran baru, `apg_plan_mgmt`, akan dibuat. Peran ini memungkinkan pengguna basis data melihat, mengelola, dan memelihara rencana kueri. Sebagai administrator dengan hak akses `rds_superuser`, pastikan untuk memberikan peran `apg_plan_mgmt` kepada pengguna basis data sesuai kebutuhan.

Hanya pengguna dengan peran `rds_superuser` yang dapat menyelesaikan prosedur berikut. `rds_superuser` diperlukan untuk membuat ekstensi `apg_plan_mgmt` dan peran `apg_plan_mgmt`. Pengguna harus diberi peran `apg_plan_mgmt` untuk mengelola ekstensi `apg_plan_mgmt`.

Mengaktifkan manajemen rencana kueri untuk klaster DB Aurora PostgreSQL Anda

Langkah-langkah berikut mengaktifkan manajemen rencana kueri untuk semua pernyataan SQL yang dikirimkan ke klaster DB Aurora PostgreSQL. Hal ini dikenal sebagai mode otomatis. Untuk mempelajari selengkapnya tentang perbedaan di antara mode, lihat Mengambil rencana eksekusi Aurora PostgreSQL.

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Buat grup parameter klaster DB kustom untuk klaster DB Aurora PostgreSQL Anda. Anda perlu mengubah parameter tertentu untuk mengaktifkan manajemen rencana kueri dan mengatur perilakunya. Untuk informasi selengkapnya, lihat Membuat grup parameter DB di Aurora.
3. Buka grup parameter klaster DB kustom dan atur parameter `rds.enable_plan_management` ke 1, seperti yang ditunjukkan pada gambar berikut ini.




   Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.
4. Buat grup parameter DB kustom yang dapat Anda gunakan untuk mengatur parameter rencana kueri di tingkat instans. Untuk informasi selengkapnya, lihat Membuat grup parameter cluster DB di Amazon Aurora.
5. Ubah instans penulis klaster DB Aurora PostgreSQL untuk menggunakan grup parameter DB kustom. Untuk informasi selengkapnya, lihat Memodifikasi instans DB dalam klaster DB.
6. Ubah klaster DB Aurora PostgreSQL untuk menggunakan grup parameter klaster DB kustom. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.
7. Boot ulang instans DB Anda untuk mengaktifkan pengaturan grup parameter kustom.

8. Hubungkan ke titik akhir instans DB untuk klaster DB Aurora PostgreSQL Anda menggunakan `psql` atau `pgAdmin`. Contoh berikut menggunakan akun `postgres` default untuk peran `rds_superuser`.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. Buat ekstensi `apg_plan_mgmt` untuk instans DB Anda, seperti yang ditunjukkan berikut ini.

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> Instal ekstensi `apg_plan_mgmt` dalam basis data templat untuk aplikasi Anda. Basis data templat default diberi nama `template1`. Untuk mempelajari selengkapnya, lihat Template Databases dalam dokumentasi PostgreSQL.

10. Ubah parameter `apg_plan_mgmt.capture_plan_baselines` menjadi `automatic`. Pengaturan ini akan menyebabkan pengoptimal menghasilkan rencana untuk setiap pernyataan SQL yang direncanakan atau dijalankan dua kali atau lebih.

> **Note**
>
> Manajemen rencana kueri juga memiliki mode manual yang dapat Anda gunakan untuk pernyataan SQL tertentu. Untuk mempelajari selengkapnya, lihat Mengambil rencana eksekusi Aurora PostgreSQL.

11. Ubah nilai parameter `apg_plan_mgmt.use_plan_baselines` menjadi "on". Parameter ini menyebabkan pengoptimal memilih rencana untuk pernyataan dari acuan dasar rencananya. Untuk mempelajari selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL.

> **Note**
>
> Anda dapat mengubah nilai salah satu parameter dinamis ini untuk sesi tanpa perlu mem-boot ulang instans.

Ketika pengaturan manajemen rencana kueri Anda selesai, pastikan untuk memberikan peran `apg_plan_mgmt` kepada setiap pengguna basis data yang perlu melihat, mengelola, atau memelihara rencana kueri.

## Meningkatkan manajemen rencana kueri Aurora PostgreSQL

Kami menyarankan agar Anda meningkatkan ekstensi manajemen rencana kueri ke rilis terbaru untuk versi Aurora PostgreSQL Anda.

1. Hubungkan ke instans penulis klaster DB Aurora PostgreSQL Anda sebagai pengguna yang memiliki hak akses `rds_superuser`. Jika Anda menyimpan nama default saat menyiapkan instans, Anda akan terhubung sebagai `postgres`. Contoh ini menunjukkan cara menggunakan `psql`, tetapi Anda juga dapat menggunakan pgAdmin jika mau.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Jalankan kueri berikut untuk meningkatkan ekstensi.

   ```
   ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
   ```

3. Gunakan fungsi apg_plan_mgmt.validate_plans untuk memperbarui hash semua rencana. Pengoptimal akan memvalidasi semua rencana yang Disetujui, Tidak Disetujui, dan Ditolak untuk memastikan bahwa rencana tersebut masih layak untuk ekstensi versi baru.

   ```
   SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
   ```

   Untuk mempelajari selengkapnya tentang penggunaan fungsi ini, lihat Memvalidasi rencana.

4. Gunakan fungsi apg_plan_mgmt.reload untuk me-refresh rencana apa pun di memori bersama dengan rencana yang divalidasi dari tampilan dba_plans.

   ```
   SELECT apg_plan_mgmt.reload();
   ```

Untuk mempelajari selengkapnya tentang semua fungsi yang tersedia untuk manajemen rencana kueri, lihat Referensi fungsi untuk manajemen rencana kueri Aurora PostgreSQL.

### Menonaktifkan manajemen rencana kueri Aurora PostgreSQL

Anda dapat menonaktifkan manajemen rencana kueri setiap saat dengan menonaktifkan `apg_plan_mgmt.use_plan_baselines` dan `apg_plan_mgmt.capture_plan_baselines`.

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;

labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

## Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL

Manajemen rencana kueri memungkinkan Anda mengendalikan cara dan waktu perubahan rencana eksekusi kueri. Sebagai DBA, tujuan utama Anda saat menggunakan QPM termasuk mencegah regresi ketika ada perubahan pada basis data Anda, dan mengontrol apakah akan mengizinkan pengoptimisasi menggunakan rencana baru. Di bagian berikut ini, Anda dapat menemukan beberapa praktik terbaik yang direkomendasikan untuk menggunakan manajemen rencana kueri. Pendekatan manajemen rencana yang proaktif dan reaktif berbeda dari segi cara dan waktu persetujuan terhadap penggunaan rencana baru.

Daftar Isi


- Manajemen rencana proaktif untuk membantu mencegah regresi performa
  - Memastikan stabilitas rencana setelah peningkatan versi mayor
- Manajemen rencana reaktif untuk mendeteksi dan memperbaiki regresi performa


### Manajemen rencana proaktif untuk membantu mencegah regresi performa

Guna mencegah regresi performa rencana terjadi, Anda perlu mengembangkan acuan dasar rencana dengan menjalankan prosedur yang membandingkan performa rencana yang baru ditemukan dengan performa acuan dasar yang sudah ada untuk rencana yang Disetujui, lalu secara otomatis menyetujui rangkaian rencana tercepat sebagai acuan dasar baru. Dengan cara ini, acuan dasar rencana meningkat dari waktu ke waktu seiring rencana yang lebih cepat ditemukan.

1. Di lingkungan pengembangan, temukan pernyataan SQL yang memiliki dampak terbesar pada performa atau throughput sistem. Kemudian, ambil rencana untuk pernyataan tersebut seperti yang dijelaskan dalam Mengambil rencana secara manual untuk pernyataan SQL tertentu dan Mengambil rencana secara otomatis.

2. Ekspor rencana yang diambil dari lingkungan pengembangan lalu impor ke lingkungan produksi. Untuk informasi selengkapnya, lihat Mengekspor dan mengimpor rencana terkelola untuk Aurora PostgreSQL.
3. Dalam produksi, jalankan aplikasi Anda dan terapkan penggunaan rencana terkelola yang disetujui. Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL. Saat aplikasi berjalan, tambahkan juga rencana baru saat pengoptimisasi menemukan rencana tersebut. Untuk informasi selengkapnya, lihat Mengambil rencana secara otomatis.
4. Analisis rencana yang belum disetujui dan setujui rencana yang beperforma baik. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.
5. Saat aplikasi Anda terus berjalan, pengoptimisasi mulai menggunakan rencana baru sesuai kebutuhan.

### Memastikan stabilitas rencana setelah peningkatan versi mayor

Setiap versi mayor PostgreSQL menyertakan penyempurnaan dan perubahan pada pengoptimisasi kueri yang dirancang untuk menaikkan performa. Namun, rencana eksekusi kueri yang dihasilkan oleh pengoptimisasi di versi sebelumnya dapat menyebabkan regresi performa di versi yang ditingkatkan yang lebih baru. Anda dapat menggunakan manajemen rencana kueri untuk mengatasi masalah performa ini dan untuk memastikan stabilitas rencana setelah peningkatan versi mayor.

Pengoptimisasi selalu menggunakan rencana yang Disetujui dengan biaya minimum, meskipun ada lebih dari satu rencana yang Disetujui untuk pernyataan yang sama. Setelah peningkatan, pengoptimisasi mungkin menemukan rencana baru, tetapi rencana tersebut akan disimpan sebagai rencana yang Tidak Disetujui. Rencana ini dilakukan hanya jika disetujui menggunakan gaya reaktif manajemen rencana dengan parameter unapproved_plan_execution_threshold. Anda dapat memaksimalkan stabilitas rencana menggunakan gaya proaktif manajemen rencana dengan parameter evolve_plan_baselines. Parameter ini membandingkan performa rencana baru dengan rencana lama dan menyetujui atau menolak rencana yang setidaknya 10% lebih cepat dari rencana terbaik berikutnya.

Setelah peningkatan, Anda dapat menggunakan fungsi `evolve_plan_baselines` untuk membandingkan performa rencana sebelum dan sesudah peningkatan menggunakan pengikatan parameter kueri Anda. Langkah-langkah berikut mengasumsikan bahwa Anda telah menggunakan rencana terkelola yang disetujui di lingkungan produksi Anda, seperti yang dijelaskan dalam Menggunakan rencana terkelola Aurora PostgreSQL.

1. Sebelum peningkatan, jalankan aplikasi Anda bersama pengelola rencana kueri yang berjalan. Saat aplikasi berjalan, tambahkan juga rencana baru saat pengoptimisasi menemukan rencana tersebut. Untuk informasi selengkapnya, lihat Mengambil rencana secara otomatis.
2. Evaluasi performa setiap rencana. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.
3. Setelah peningkatan, analisis kembali rencana Anda yang disetujui menggunakan fungsi `evolve_plan_baselines`. Bandingkan performa sebelum dan sesudah menggunakan pengikatan parameter kueri Anda. Jika rencana baru cepat, Anda dapat menambahkannya ke rencana yang disetujui. Jika lebih cepat dari rencana lain untuk pengikatan parameter yang sama, maka Anda dapat menandai rencana yang lebih lambat sebagai Ditolak.

   Untuk informasi selengkapnya, lihat Menyetujui rencana yang lebih baik. Untuk informasi referensi tentang fungsi ini, lihat apg_plan_mgmt.evolve_plan_baselines.

Untuk informasi selengkapnya, lihat artikel Memastikan performa yang konsisten setelah peningkatan versi mayor dengan Manajemen Rencana Kueri Amazon Aurora Edisi Kompatibel PostgreSQL.

> **Note**
>
> Saat Anda melakukan peningkatan versi mayor menggunakan replikasi logis atau AWS DMS, pastikan Anda mereplikasi skema `apg_plan_mgmt` untuk memastikan rencana yang sudah ada disalin ke instans yang ditingkatkan. Untuk informasi selengkapnya tentang replikasi logis, lihat Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL.

### Manajemen rencana reaktif untuk mendeteksi dan memperbaiki regresi performa

Dengan memantau aplikasi Anda saat berjalan, Anda dapat mendeteksi rencana yang menyebabkan regresi performa. Saat Anda mendeteksi regresi, Anda dapat secara manual menolak atau memperbaiki rencana yang buruk dengan mengikuti langkah-langkah berikut:

1. Saat aplikasi Anda berjalan, terapkan penggunaan rencana terkelola dan tambahkan secara otomatis rencana yang baru ditemukan sebagai rencana yang tidak disetujui. Untuk informasi lebih lanjut, lihat Menggunakan rencana terkelola Aurora PostgreSQL dan Mengambil rencana secara otomatis.
2. Pantau regresi performa pada aplikasi Anda yang sedang berjalan.

3. Ketika Anda menemukan regresi rencana, atur status rencana menjadi `rejected`. Saat pengoptimisasi kembali menjalankan pernyataan SQL, pengoptimisasi akan secara otomatis mengabaikan rencana yang ditolak dan menggunakan rencana berbeda yang telah disetujui. Untuk informasi selengkapnya, lihat Menolak atau menonaktifkan rencana yang lebih lambat.

   Dalam beberapa kasus, Anda mungkin lebih memilih untuk memperbaiki rencana yang buruk daripada menolak, menonaktifkan, atau menghapusnya. Gunakan ekstensi `pg_hint_plan` untuk bereksperimen dalam meningkatkan rencana. Dengan `pg_hint_plan`, Anda menggunakan komentar khusus untuk meminta pengoptimisasi mengganti caranya dalam membuat rencana. Lihat informasi yang lebih lengkap di Memperbaiki rencana menggunakan pg_hint_plan.

## Manajemen rencana kueri Aurora PostgreSQL

Dengan manajemen rencana kueri diaktifkan untuk klaster DB Aurora PostgreSQL Anda, pengoptimisasi akan menghasilkan dan menyimpan rencana eksekusi kueri untuk pernyataan SQL apa pun yang diproses lebih dari sekali. Pengoptimisasi selalu mengatur status rencana yang pertama kali dibuat dari suatu pernyataan terkelola ke `Approved`, dan menyimpannya di tampilan `dba_plans`.

Rangkaian rencana yang disetujui yang disimpan untuk suatu pernyataan terkelola disebut sebagai acuan dasar rencana. Saat aplikasi berjalan, pengoptimisasi dapat membuat rencana tambahan untuk pernyataan terkelola. Pengoptimisasi mengatur rencana tambahan yang diambil ke status `Unapproved`.

Kemudian, Anda dapat memutuskan apakah rencana `Unapproved` berjalan dengan baik dan mengubahnya menjadi `Approved`, `Rejected`, atau `Preferred`. Untuk melakukannya, Anda menggunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` atau fungsi `apg_plan_mgmt.set_plan_status`.

Ketika pengoptimisasi menghasilkan rencana untuk pernyataan SQL, manajemen rencana kueri menyimpan rencana ini dalam tabel `apg_plan_mgmt.plans`. Pengguna basis data yang telah diberi peran `apg_plan_mgmt` dapat melihat detail rencana dengan mengueri tampilan `apg_plan_mgmt.dba_plans`. Misalnya, kueri berikut menampilkan daftar detail untuk rencana yang saat ini ada dalam tampilan untuk klaster DB PostgreSQL Aurora non-produksi.

- `sql_hash` – Pengidentifikasi untuk pernyataan SQL yang merupakan nilai hash untuk teks yang dinormalisasi dari pernyataan SQL.

- `plan_hash` – Pengidentifikasi unik untuk rencana yang merupakan kombinasi dari `sql_hash` dan hash rencana.
- `status` – Status rencana. Pengoptimisasi dapat menjalankan rencana yang disetujui.
- `enabled` – Menunjukkan apakah rencana siap digunakan (true) atau tidak (false).
- `plan_outline` – Representasi rencana yang digunakan untuk membuat ulang rencana eksekusi sebenarnya. Operator dalam struktur pohon akan dipetakan ke operator dalam output EXPLAIN.

Tampilan `apg_plan_mgmt.dba_plans` memiliki lebih banyak kolom yang berisi semua detail rencana, misalnya, waktu rencana terakhir digunakan. Untuk detail lengkap, lihat Referensi untuk tampilan apg_plan_mgmt.dba_plans untuk Aurora PostgreSQL Compatible Edition.

## Normalisasi dan hash SQL

Dalam tampilan `apg_plan_mgmt.dba_plans`, Anda dapat mengidentifikasi pernyataan terkelola dengan nilai hash SQL. Hash SQL dihitung pada representasi ternormalisasi dari pernyataan SQL yang menghilangkan beberapa perbedaan, seperti nilai-nilai literal.

Proses normalisasi untuk setiap pernyataan SQL mempertahankan spasi dan huruf besar/kecil, sehingga Anda masih dapat membaca dan memahami inti dari pernyataan SQL tersebut. Normalisasi akan menghapus atau mengganti item berikut.

- Komentar blok di depan
- Kata kunci EXPLAIN dan opsi EXPLAIN, serta EXPLAIN ANALYZE
- Spasi di belakang
- Semua literal

Sebagai contoh, amati pernyataan berikut.

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

Manajemen rencana kueri menormalkan pernyataan ini seperti yang ditunjukkan:

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

Normalisasi memungkinkan hash SQL yang sama digunakan untuk pernyataan SQL serupa yang mungkin hanya berbeda dalam hal nilai literal atau parameternya. Dengan kata lain, bisa ada lebih

dari satu rencana untuk hash SQL yang sama, dengan sebuah rencana berbeda yang optimal dalam kondisi yang berbeda-beda.

> **Note**
>
> Pernyataan SQL tunggal yang digunakan dengan berbagai skema akan memiliki rencana yang berbeda-beda karena terikat pada skema tertentu saat runtime. Perencana menggunakan statistik untuk pengikatan skema untuk memilih rencana yang optimal.

Untuk mempelajari selengkapnya tentang cara pengoptimisasi memilih rencana, lihat Menggunakan rencana terkelola Aurora PostgreSQL. Di bagian tersebut, Anda dapat mempelajari cara menggunakan `EXPLAIN` dan `EXPLAIN ANALYZE` untuk melihat pratinjau rencana sebelum benar-benar digunakan. Untuk detailnya, lihat Menganalisis rencana yang dipilih pengoptimisasi. Untuk gambar yang menguraikan proses untuk memilih rencana, lihat Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan.

## Mengambil rencana eksekusi Aurora PostgreSQL

Manajemen rencana kueri Aurora PostgreSQL menawarkan dua mode berbeda untuk mengambil rencana eksekusi kueri, yaitu otomatis atau manual. Anda memilih mode dengan mengatur nilai `apg_plan_mgmt.capture_plans_baselines` ke `automatic` atau ke `manual`. Anda dapat mengambil rencana eksekusi untuk pernyataan SQL tertentu menggunakan pengambilan rencana manual. Alternatifnya, Anda dapat mengambil semua rencana (atau rencana yang paling lambat) yang dijalankan dua kali atau lebih saat aplikasi Anda berjalan menggunakan pengambilan rencana otomatis.

Saat mengambil rencana, pengoptimisasi selalu mengatur status rencana yang pertama kali diambil untuk suatu pernyataan terkelola menjadi `approved`. Pengoptimisasi selalu mengatur status setiap rencana tambahan yang diambil untuk suatu pernyataan terkelola menjadi `unapproved`. Namun, kadang-kadang lebih dari satu rencana dapat disimpan dengan status `approved`. Hal ini dapat terjadi ketika beberapa rencana dibuat untuk sebuah pernyataan secara paralel dan sebelum rencana pertama untuk pernyataan tersebut ditetapkan.

Untuk mengontrol jumlah maksimum rencana yang dapat diambil dan disimpan di tampilan `dba_plans`, atur parameter `apg_plan_mgmt.max_plans` dalam grup parameter tingkat instans DB Anda. Perubahan pada parameter `apg_plan_mgmt.max_plans` mengharuskan instans DB di-boot ulang agar nilai baru dapat diterapkan. Untuk informasi selengkapnya, lihat parameter apg_plan_mgmt.max_plans.

### Mengambil rencana secara manual untuk pernyataan SQL tertentu

Jika Anda memiliki kumpulan pernyataan SQL yang diketahui untuk dikelola, masukkan pernyataan tersebut ke dalam file skrip SQL lalu ambil rencana secara manual. Hal berikut ini menunjukkan contoh psql tentang cara mengambil rencana kueri secara manual untuk kumpulan pernyataan SQL.

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Setelah mengambil rencana untuk setiap pernyataan SQL, pengoptimisasi menambahkan baris baru ke tampilan `apg_plan_mgmt.dba_plans`.

Kami menyarankan Anda menggunakan pernyataan EXPLAIN atau EXPLAIN EXECUTE di file skrip SQL. Pastikan Anda memasukkan cukup variasi dalam nilai parameter untuk mengambil semua rencana yang diinginkan.

Jika Anda mengetahui rencana yang lebih baik daripada rencana berbiaya minimum dari pengoptimisasi, Anda mungkin dapat memaksa pengoptimisasi untuk menggunakan rencana yang lebih baik. Untuk melakukannya, tentukan satu atau beberapa petunjuk pengoptimal. Untuk informasi selengkapnya, lihat Memperbaiki rencana menggunakan pg_hint_plan. Untuk membandingkan performa rencana `unapproved` dan `approved` serta menyetujui, menolak, atau menghapusnya, lihat Mengevaluasi performa rencana.

### Mengambil rencana secara otomatis

Gunakan pengambilan rencana otomatis untuk situasi seperti berikut ini:

- Anda tidak mengetahui pernyataan SQL spesifik yang ingin Anda kelola.
- Anda memiliki ratusan atau ribuan pernyataan SQL untuk dikelola.
- Aplikasi Anda menggunakan API klien. Misalnya, JDBC menggunakan pernyataan yang disiapkan tanpa nama atau pernyataan mode massal yang tidak dapat dinyatakan dalam psql.

Untuk mengambil rencana secara otomatis

1. Aktifkan pengambilan rencana otomatis dengan mengatur `apg_plan_mgmt.capture_plan_baselines` ke `automatic` dalam grup parameter tingkat instans DB. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB di .

2. Saat aplikasi berjalan, pengoptimisasi mengambil rencana untuk setiap pernyataan SQL yang berjalan setidaknya dua kali.

   Saat aplikasi berjalan dengan pengaturan parameter manajemen rencana kueri default, pengoptimisasi mengambil rencana untuk setiap pernyataan SQL yang berjalan setidaknya dua kali. Dengan mengambil semua rencana saat menggunakan pengaturan default, overhead run-time yang diperlukan akan sangat kecil dan hal ini dapat dilakukan dalam produksi.

Untuk menonaktifkan pengambilan rencana otomatis

- Tetapkan parameter `apg_plan_mgmt.capture_plan_baselines` ke `off` dari grup parameter tingkat instans DB.

Untuk mengukur performa rencana yang belum disetujui dan menyetujui, menolak, atau menghapusnya, lihat Mengevaluasi performa rencana.

# Menggunakan rencana terkelola Aurora PostgreSQL

Agar pengoptimisasi menggunakan rencana yang diambil untuk pernyataan terkelola Anda, atur parameter `apg_plan_mgmt.use_plan_baselines` ke `true`. Berikut ini adalah contoh instans lokal.

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

Sementara aplikasi berjalan, pengaturan ini menyebabkan pengoptimisasi menggunakan rencana berbiaya minimum, pilihan, atau disetujui yang valid dan diaktifkan untuk setiap pernyataan terkelola.

## Menganalisis rencana yang dipilih pengoptimisasi

Jika parameter `apg_plan_mgmt.use_plan_baselines` diatur ke `true`, Anda dapat menggunakan pernyataan SQL EXPLAIN ANALYZE untuk membuat pengoptimisasi menampilkan rencana yang akan digunakan jika pengoptimisasi akan menjalankan pernyataan tersebut. Berikut adalah contohnya.

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                      QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

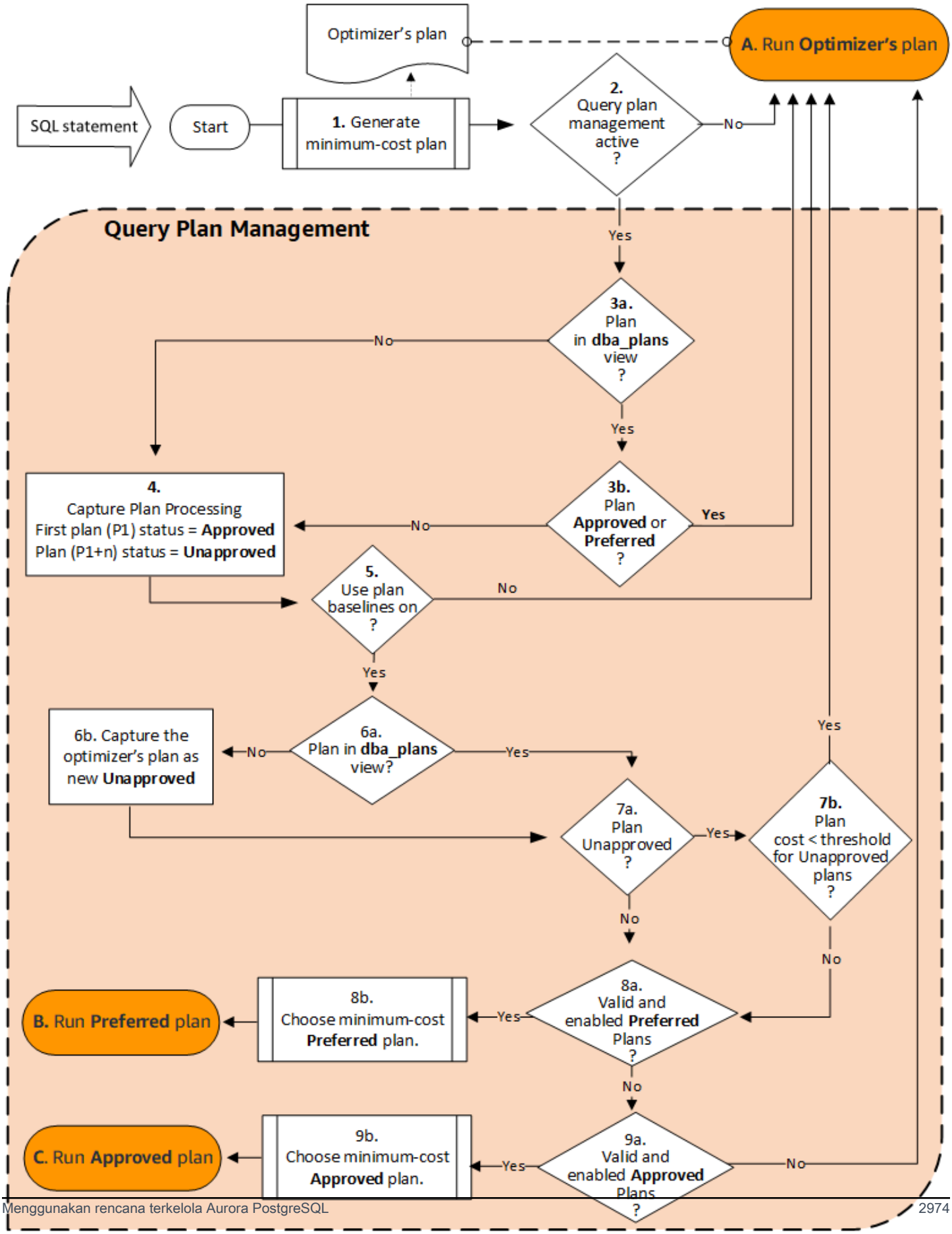
Output ini menunjukkan rencana yang Disetujui dari acuan dasar (baseline) yang akan berjalan. Namun, output ini juga menunjukkan bahwa rencana yang berbiaya lebih rendah ditemukan. Dalam hal ini, Anda perlu mengambil rencana berbiaya minimum yang baru ini dengan mengaktifkan pengambilan rencana otomatis seperti yang dijelaskan dalam Mengambil rencana secara otomatis.

Rencana baru selalu diambil oleh pengoptimisasi sebagai `Unapproved`. Gunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` untuk membandingkan rencana dan mengubahnya menjadi disetujui, ditolak, atau dinonaktifkan. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.

## Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan

Biaya rencana eksekusi adalah perkiraan yang dibuat oleh pengoptimisasi untuk membandingkan beberapa rencana yang berbeda. Saat menghitung biaya rencana, pengoptimisasi mempertimbangkan faktor-faktor seperti operasi CPU dan I/O yang diperlukan oleh rencana tersebut. Untuk mempelajari selengkapnya tentang perkiraan biaya perencana kueri PostgreSQL, lihat Query Planning dalam dokumentasi PostgreSQL.

Gambar berikut menunjukkan bagaimana rencana dipilih untuk pernyataan SQL tertentu ketika manajemen rencana kueri aktif, dan ketika tidak.


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Query Plan Management
Yes
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing First plan (P1) status = Approved Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
No
Yes
5. Use plan baselines on ?
No
Yes
6b. Capture the optimizer's plan as new Unapproved
6a. Plan in dba_plans view?
No
Yes
7a. Plan Unapproved ?
Yes
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
No
B. Run Preferred plan
8b. Choose minimum-cost Preferred plan.
Yes
8a. Valid and enabled Preferred Plans ?
No
C. Run Approved plan
9b. Choose minimum-cost Approved plan.
Yes
9a. Valid and enabled Approved Plans ?
No

Alurnya adalah sebagai berikut:

1. Pengoptimisasi menghasilkan rencana berbiaya minimum untuk pernyataan SQL.
2. Jika manajemen rencana kueri tidak aktif, rencana pengoptimisasi akan segera dijalankan (A. Jalankan rencana Pengoptimisasi). Manajemen rencana kueri tidak aktif ketika parameter `apg_plan_mgmt.capture_plan_baselines` dan `apg_plan_mgmt.use_plan_baselines` berada pada pengaturan default-nya (masing-masing “off” dan “false”).

   Jika tidak, manajemen rencana kueri aktif. Dalam hal ini, pernyataan SQL dan rencana pengoptimisasi untuk pernyataan tersebut akan dinilai lebih lanjut sebelum rencana dipilih.

   > **Tip**
   >
   > Pengguna basis data dengan peran `apg_plan_mgmt` dapat secara proaktif membandingkan rencana, mengubah status rencana, dan memaksa penggunaan rencana tertentu sesuai kebutuhan. Untuk informasi selengkapnya, lihat Meningkatkan rencana kueri PostgreSQL Aurora.

3. Pernyataan SQL mungkin sudah memiliki rencana yang disimpan oleh manajemen rencana kueri di masa lalu. Rencana disimpan di `apg_plan_mgmt.dba_plans`, bersama dengan informasi tentang pernyataan SQL yang digunakan untuk membuatnya. Informasi tentang rencana akan menyertakan statusnya. Status rencana dapat menentukan apakah rencana tersebut digunakan atau tidak, sebagai berikut.
   a. Jika rencana tidak termasuk di antara rencana yang disimpan untuk pernyataan SQL, berarti ini adalah pertama kalinya rencana spesifik ini dihasilkan oleh pengoptimisasi untuk pernyataan SQL yang diberikan. Rencana dikirim ke Pemrosesan Pengambilan Rencana (4).
   b. Jika rencana termasuk di antara rencana yang disimpan dan statusnya Disetujui atau Pilihan, rencana ini akan dijalankan (A. Jalankan rencana Pengoptimisasi).

      Jika rencana termasuk di antara rencana yang disimpan, tetapi tidak berstatus Disetujui atau Pilihan, rencana ini akan dikirim ke Pemrosesan Pengambilan Rencana (4).
4. Ketika rencana diambil pertama kalinya untuk pernyataan SQL tertentu, status rencana ini selalu diatur ke Disetujui (P1). Jika pengoptimisasi selanjutnya menghasilkan rencana yang sama untuk pernyataan SQL yang sama, status rencana tersebut diubah menjadi Tidak Disetujui (P1+n).

   Dengan rencana diambil dan statusnya diperbarui, evaluasi berlanjut pada langkah berikutnya (5).

5. Acuan dasar rencana terdiri dari riwayat pernyataan SQL dan rencananya dalam berbagai status. Manajemen rencana kueri dapat mempertimbangkan acuan dasar ini saat memilih rencana, tergantung pada apakah opsi Gunakan acuan dasar rencana diaktifkan atau tidak, sebagai berikut.
   - Gunakan acuan dasar rencana “tidak aktif” ketika parameter `apg_plan_mgmt.use_plan_baselines` diatur ke nilai default (`false`). Rencana ini tidak akan dibandingkan dengan acuan dasar (baseline) sebelum dijalankan (A. Jalankan rencana Pengoptimisasi).
   - Gunakan acuan dasar rencana “aktif” saat parameter `apg_plan_mgmt.use_plan_baselines` diatur ke `true`. Rencana tersebut dinilai lebih lanjut menggunakan acuan dasar (6).
6. Rencana tersebut dibandingkan dengan rencana lain untuk pernyataan di acuan dasar.
   a. Jika rencana pengoptimisasi termasuk di antara rencana di acuan dasar, statusnya diperiksa (7a).
   b. Jika rencana pengoptimisasi tidak termasuk di antara rencana dalam acuan dasar, rencana ini akan ditambahkan ke rencana untuk pernyataan tersebut sebagai rencana `Unapproved` baru.
7. Status rencana diperiksa hanya untuk menentukan apakah statusnya Tidak Disetujui.
   a. Jika status rencana Tidak Disetujui, perkiraan biaya rencana ini dibandingkan dengan perkiraan biaya yang ditentukan untuk ambang batas rencana eksekusi yang tidak disetujui.
      - Jika perkiraan biaya rencana di bawah ambang batas, pengoptimisasi akan menggunakannya meskipun rencana ini Tidak Disetujui (A. Jalankan rencana Pengoptimisasi). Umumnya, pengoptimisasi tidak akan menjalankan rencana yang Tidak Disetujui. Namun, ketika parameter `apg_plan_mgmt.unapproved_plan_execution_threshold` menentukan nilai ambang batas biaya, pengoptimisasi akan membandingkan biaya rencana yang Tidak Disetujui dengan ambang batas ini. Jika perkiraan biaya lebih rendah dari ambang batas, pengoptimisasi akan menjalankan rencana. Untuk informasi selengkapnya, lihat apg_plan_mgmt.unapproved_plan_execution_threshold.
      - Jika perkiraan biaya rencana tidak di bawah ambang batas, atribut lainnya dari rencana ini akan diperiksa (8a).
   b. Jika status rencana adalah apa pun selain Tidak Disetujui, atribut lainnya akan diperiksa (8a).
8. Pengoptimisasi tidak akan menggunakan rencana yang dinonaktifkan. Rencana yang dinonaktifkan adalah rencana yang atribut `enable`-nya diatur ke 'f' (false). Pengoptimisasi juga tidak akan menggunakan rencana yang memiliki status Ditolak.

Pengoptimisasi tidak dapat menggunakan rencana apa pun yang tidak valid. Rencana dapat menjadi tidak valid dari waktu ke waktu ketika objek yang diandalkannya, misalnya indeks dan partisi tabel, dihapus.

a. Jika pernyataan memiliki rencana Pilihan yang diaktifkan dan valid, pengoptimisasi akan memilih rencana berbiaya minimum di antara rencana Pilihan yang disimpan untuk pernyataan SQL ini. Pengoptimisasi kemudian menjalankan rencana Pilihan berbiaya minimum.

b. Jika pernyataan tidak memiliki rencana Pilihan yang diaktifkan dan valid, pernyataan ini akan dinilai pada langkah berikutnya (9).

9. Jika pernyataan memiliki rencana Disetujui yang diaktifkan dan valid, pengoptimisasi akan memilih rencana berbiaya minimum dari berbagai rencana Disetujui yang disimpan untuk pernyataan SQL ini. Pengoptimisasi kemudian menjalankan rencana berbiaya minimum yang Disetujui.

   Jika pernyataan tidak memiliki rencana Disetujui yang valid dan diaktifkan, pengoptimisasi akan menggunakan rencana berbiaya minimum (A. Jalankan rencana Pengoptimisasi).

## Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans

Pengguna dan administrator basis data yang telah diberi peran `apg_plan_mgmt` dapat melihat dan mengelola rencana yang disimpan dalam `apg_plan_mgmt.dba_plans`. Administrator klaster DB Aurora PostgreSQL (seseorang dengan izin `rds_superuser`) harus secara eksplisit memberikan peran ini kepada pengguna basis data yang perlu menggunakan manajemen rencana kueri.

Tampilan `apg_plan_mgmt` berisi riwayat rencana untuk semua pernyataan SQL terkelola untuk setiap basis data pada instans penulis dari klaster DB PostgreSQL Aurora. Tampilan ini memungkinkan Anda memeriksa rencana, statusnya, kapan terakhir digunakan, dan semua detail relevan lainnya.

Seperti yang dibahas dalam Normalisasi dan hash SQL, setiap rencana terkelola diidentifikasi oleh nilai hash SQL dan nilai hash rencana. Dengan pengidentifikasi ini, Anda dapat menggunakan alat seperti Wawasan Performa Amazon RDS untuk melacak performa rencana individu. Untuk informasi selengkapnya tentang Wawasan Performa, lihat Menggunakan Wawasan Performa Amazon RDS.

### Menampilkan daftar rencana yang dikelola

Untuk menampilkan daftar rencana terkelola, gunakan SELECT pada tampilan `apg_plan_mgmt.dba_plans`. Contoh berikut menampilkan beberapa kolom dalam tampilan `dba_plans` seperti `status`, yang mengidentifikasi rencana yang disetujui dan tidak disetujui.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

 sql_hash   | plan_hash |   status   | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

Agar mudah dibaca, kueri dan output yang ditampilkan hanya mencantumkan sebagian kecil kolom dari tampilan `dba_plans`. Untuk informasi selengkapnya, lihat Referensi untuk tampilan apg_plan_mgmt.dba_plans untuk Aurora PostgreSQL Compatible Edition.

## Meningkatkan rencana kueri PostgreSQL Aurora

Meningkatkan manajemen rencana kueri dengan mengevaluasi kinerja rencana dan memperbaiki rencana. Untuk informasi selengkapnya tentang meningkatkan rencana kueri Anda, lihat topik berikut.

Topik

- Mengevaluasi performa rencana
- Memperbaiki rencana menggunakan pg_hint_plan

### Mengevaluasi performa rencana

Setelah pengoptimal mengambil rencana sebagai tidak disetujui, gunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` untuk membandingkan rencana berdasarkan performanya yang sebenarnya. Bergantung pada hasil eksperimen performa Anda, Anda dapat mengubah status rencana dari tidak disetujui ke disetujui atau ditolak. Anda dapat memutuskan untuk menggunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` untuk menonaktifkan rencana secara sementara jika tidak memenuhi persyaratan Anda.

Menyetujui rencana yang lebih baik

Contoh berikut menunjukkan cara mengubah status rencana terkelola menjadi disetujui menggunakan fungsi `apg_plan_mgmt.evolve_plan_baselines`.

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
```

```
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
evolve_plan_baselines
----------------------
0
(1 row)
```

Output menampilkan laporan performa untuk pernyataan `rangequery` dengan ikatan parameter 1 dan 10.000. Rencana baru yang belum disetujui (`Baseline+1`) lebih baik daripada rencana terbaik yang telah disetujui sebelumnya (`Baseline`). Untuk mengonfirmasi bahwa rencana yang baru sekarang berstatus `Approved`, periksa tampilan `apg_plan_mgmt.dba_plans`.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash  | plan_hash |  status  | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

Rencana terkelola kini mencakup dua rencana yang disetujui yang merupakan acuan dasar rencana pernyataan. Anda juga dapat memanggil fungsi `apg_plan_mgmt.set_plan_status` untuk mengatur langsung bidang status rencana ke `'Approved'`, `'Rejected'`, `'Unapproved'`, atau `'Preferred'`.

Menolak atau menonaktifkan rencana yang lebih lambat

Untuk menolak atau menonaktifkan rencana, teruskan `'reject'` atau `'disable'` sebagai parameter tindakan ke fungsi `apg_plan_mgmt.evolve_plan_baselines`. Contoh ini

menonaktifkan setiap rencana Unapproved yang diambil yang lebih lambat setidaknya 10 persen dari rencana Approved terbaik untuk pernyataan.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

Anda juga dapat secara langsung mengatur rencana ke ditolak atau dinonaktifkan. Untuk secara langsung mengatur bidang yang diaktifkan untuk rencana ke `true` atau `false`, panggil fungsi `apg_plan_mgmt.set_plan_enabled`. Untuk secara langsung mengatur bidang status rencana ke `'Approved'`, `'Rejected'`, `'Unapproved'`, atau `'Preferred'`, panggil fungsi `apg_plan_mgmt.set_plan_status`.

Untuk menghapus rencana yang tidak valid dan yang Anda perkirakan akan tetap tidak valid, gunakan fungsi `apg_plan_mgmt.validate_plans`. Fungsi ini memungkinkan Anda menghapus atau menonaktifkan rencana yang tidak valid. Untuk informasi selengkapnya, lihat Memvalidasi rencana.

### Memperbaiki rencana menggunakan pg_hint_plan

Pengoptimal kueri dirancang dengan baik untuk menemukan rencana yang optimal untuk semua pernyataan, dan pada sebagian besar kasus, pengoptimal menemukan rencana yang baik. Namun, terkadang Anda mungkin tahu bahwa ada rencana yang jauh lebih baik dari yang dihasilkan oleh pengoptimal. Dua cara yang disarankan agar pengoptimal menghasilkan rencana yang diinginkan adalah menggunakan ekstensi `pg_hint_plan` atau mengatur variabel Grand Unified Configuration (GUC) di PostgreSQL:

- Ekstensi `pg_hint_plan` – Menentukan "petunjuk" untuk memodifikasi cara perencana bekerja menggunakan ekstensi `pg_hint_plan` PostgreSQL. Untuk menginstal dan mempelajari selengkapnya tentang cara menggunakan ekstensi `pg_hint_plan`, lihat dokumentasi pg_hint_plan.
- Variabel GUC – Menulis ulang satu atau beberapa parameter model biaya atau parameter pengoptimal lainnya, seperti `from_collapse_limit` atau `GEQO_threshold`.

Saat Anda menggunakan salah satu teknik ini untuk memaksa pengoptimal kueri agar menggunakan suatu rencana, Anda juga dapat menggunakan manajemen rencana kueri untuk mengambil dan memberlakukan penggunaan rencana baru.

Anda dapat menggunakan ekstensi `pg_hint_plan` untuk mengubah urutan join, metode join, atau jalur akses untuk pernyataan SQL. Anda menggunakan komentar SQL dengan sintaksis `pg_hint_plan` khusus untuk memodifikasi cara pengoptimal membuat rencana. Misalnya, anggaplah pernyataan SQL yang bermasalah memiliki join dua arah.

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Kemudian, anggaplah bahwa pengoptimal memilih urutan join (t1, t2), tetapi Anda tahu bahwa urutan join (t2, t1) lebih cepat. Petunjuk berikut memaksa pengoptimal untuk menggunakan urutan join yang lebih cepat, (t2, t1). Sertakan EXPLAIN sehingga pengoptimal menghasilkan rencana untuk pernyataan SQL, tetapi tanpa menjalankan pernyataan ini. (Output tidak ditampilkan.)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Langkah-langkah berikut menunjukkan cara menggunakan `pg_hint_plan`.

Untuk mengubah rencana yang dibuat oleh pengoptimal dan mengambil rencana menggunakan pg_hint_plan

1. Aktifkan mode pengambilan manual.

   ```
   SET apg_plan_mgmt.capture_plan_baselines = manual;
   ```

2. Tentukan petunjuk untuk pernyataan SQL yang ingin diamati.

   ```
   /*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
   FROM t1, t2
   WHERE t1.id = t2.id;
   ```

   Setelah proses ini berjalan, pengoptimal mengambil rencana dalam tampilan `apg_plan_mgmt.dba_plans`. Rencana yang diambil tidak mencakup sintaksis komentar

pg_hint_plan khusus karena manajemen rencana kueri menormalisasi pernyataan dengan menghapus komentar awal.

3. Lihat rencana terkelola menggunakan tampilan apg_plan_mgmt.dba_plans.

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. Atur status rencana menjadi Preferred. Tindakan ini memastikan bahwa pengoptimal memilih untuk menjalankannya dan tidak memilih dari rencana yang disetujui, saat rencana berbiaya minimum belum berstatus Approved atau Preferred.

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. Nonaktifkan pengambilan rencana manual dan berlakukan penggunaan rencana terkelola.

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

Sekarang, ketika pernyataan SQL asli berjalan, pengoptimal memilih rencana Approved atau Preferred. Jika rencana berbiaya minimum tidak berstatus Approved atau Preferred, pengoptimal akan memilih rencana Preferred.

## Menghapus paket kueri Aurora PostgreSQL

Hapus paket eksekusi yang tidak Anda gunakan atau paket yang tidak valid. Untuk informasi selengkapnya tentang menghapus paket, lihat bagian berikut.

Topik

- Menghapus rencana
- Memvalidasi rencana

### Menghapus rencana

Rencana dihapus secara otomatis jika belum digunakan selama lebih dari sebulan, khususnya, 32 hari. Ini adalah pengaturan default untuk parameter apg_plan_mgmt.plan_retention_period. Anda dapat mengubah periode retensi rencana ke periode waktu yang lebih lama, atau ke periode waktu yang lebih pendek mulai dari nilai 1. Menentukan jumlah hari sejak rencana terakhir

digunakan yang dihitung dengan mengurangi tanggal saat ini dengan tanggal `last_used`. Tanggal `last_used` adalah tanggal terbaru saat pengoptimal memilih rencana sebagai rencana berbiaya minimum atau saat rencana tersebut dijalankan. Tanggal disimpan untuk rencana dalam tampilan `apg_plan_mgmt.dba_plans`.

Kami menyarankan Anda menghapus rencana yang belum digunakan dalam waktu lama atau yang tidak berguna. Setiap rencana memiliki tanggal `last_used` yang diperbarui oleh pengoptimal setiap kali pengoptimal menjalankan rencana atau memilih rencana sebagai rencana berbiaya minimum untuk suatu pernyataan. Periksa tanggal `last_used` terakhir untuk mengidentifikasi rencana yang dapat Anda hapus dengan aman.

Kueri berikut menampilkan tabel tiga kolom dengan jumlah total rencana, rencana yang gagal dihapus, dan rencana yang berhasil dihapus. Kueri tersebut memiliki kueri bersarang yang merupakan contoh cara menggunakan fungsi `apg_plan_mgmt.delete_plan` untuk menghapus semua rencana yang belum dipilih sebagai rencana berbiaya minimum dalam 31 hari terakhir dan statusnya bukan `Rejected`.

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```
 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
           3 |                0 |                    2
```

Untuk informasi selengkapnya, lihat apg_plan_mgmt.delete_plan.

Untuk menghapus rencana yang tidak valid dan yang Anda perkirakan akan tetap tidak valid, gunakan fungsi `apg_plan_mgmt.validate_plans`. Fungsi ini memungkinkan Anda menghapus atau menonaktifkan rencana yang tidak valid. Untuk informasi selengkapnya, lihat Memvalidasi rencana.

> **Important**
>
> Jika Anda tidak menghapus rencana yang tidak berguna, Anda mungkin akan kehabisan memori bersama yang dialokasikan untuk manajemen rencana kueri. Untuk mengontrol berapa banyak memori yang tersedia untuk rencana terkelola, gunakan parameter `apg_plan_mgmt.max_plans`. Atur parameter ini di grup parameter DB kustom Anda lalu boot ulang instans DB Anda agar perubahan pada instans diterapkan. Untuk informasi selengkapnya, lihat parameter apg_plan_mgmt.max_plans.

### Memvalidasi rencana

Gunakan fungsi `apg_plan_mgmt.validate_plans` untuk menghapus atau menonaktifkan rencana yang tidak valid.

Rencana dapat menjadi tidak valid atau stale ketika objek yang diandalkan rencana tersebut dihapus, misalnya indeks atau tabel. Namun, rencana mungkin hanya menjadi tidak valid untuk sementara jika objek yang dihapus dibuat kembali. Jika rencana yang tidak valid dapat menjadi valid di lain waktu, mungkin Anda sebaiknya memilih untuk menonaktifkan rencana yang tidak valid atau tidak melakukan apa-apa daripada menghapusnya.

Untuk menemukan dan menghapus semua rencana yang tidak valid dan belum digunakan dalam kurun waktu seminggu, gunakan fungsi `apg_plan_mgmt.validate_plans` sebagai berikut.

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

Untuk mengaktifkan atau menonaktifkan rencana secara langsung, gunakan fungsi `apg_plan_mgmt.set_plan_enabled`.

## Mengekspor dan mengimpor rencana terkelola untuk Aurora PostgreSQL

Anda dapat mengekspor rencana terkelola Anda dan mengimpornya ke dalam instans DB lain.

Untuk mengekspor rencana terkelola

Pengguna dapat menyalin subset tabel `apg_plan_mgmt.plans` ke tabel lain, lalu menyimpannya menggunakan perintah pg_dump. Berikut adalah contohnya.

```
CREATE TABLE plans_copy AS SELECT *
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

Untuk mengimpor rencana terkelola

1. Salin file .tar dari rencana terkelola yang diekspor ke sistem tempat rencana tersebut akan dipulihkan.
2. Gunakan perintah `pg_restore` untuk menyalin file .tar ke dalam tabel baru.

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```

3. Gabungkan tabel `plans_copy` dengan tabel `apg_plan_mgmt.plans` seperti yang ditunjukkan dalam contoh berikut.

   > **Note**
   >
   > Dalam beberapa kasus, Anda mungkin membuang dari satu versi ekstensi `apg_plan_mgmt` dan memulihkan ke versi yang berbeda. Dalam kasus ini, kolom di tabel rencana mungkin berbeda. Jika ya, tentukan nama kolom tersebut dengan eksplisit dan bukan menggunakan SELECT *.

   ```
   INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
    ON CONFLICT ON CONSTRAINT plans_pkey
    DO UPDATE SET
    status = EXCLUDED.status,
    enabled = EXCLUDED.enabled,
    -- Save the most recent last_used date
    --
    last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
    THEN EXCLUDED.last_used ELSE plans.last_used END,
    -- Save statistics gathered by evolve_plan_baselines, if it ran:
    --
    estimated_startup_cost = EXCLUDED.estimated_startup_cost,
   ```

```
  estimated_total_cost = EXCLUDED.estimated_total_cost,
  planning_time_ms = EXCLUDED.planning_time_ms,
  execution_time_ms = EXCLUDED.execution_time_ms,
  total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
  execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. Muat ulang rencana terkelola ke dalam memori bersama dan hapus tabel rencana sementara.

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

## Referensi parameter untuk manajemen rencana kueri Aurora PostgreSQL

Anda dapat mengatur preferensi Anda untuk ekstensi `apg_plan_mgmt` dengan menggunakan parameter yang tercantum di bagian ini. Hal ini tersedia dalam parameter klaster DB kustom dan grup parameter DB yang terkait dengan klaster DB Aurora PostgreSQL Anda. Parameter ini mengontrol perilaku fitur manajemen rencana kueri dan bagaimana pengaruhnya terhadap pengoptimisasi. Untuk informasi tentang pengaturan manajemen rencana kueri, lihat Mengaktifkan manajemen rencana kueri Aurora PostgreSQL. Perubahan pada parameter berikut tidak berpengaruh jika ekstensi `apg_plan_mgmt` tidak diatur seperti yang dijelaskan dalam bagian tersebut. Untuk informasi tentang memodifikasi parameter, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora dan Grup parameter DB untuk instans Aurora DB Amazon.

Parameter

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold
- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans
- apg_plan_mgmt.plan_hash_version
- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines
- auto_explain.hashes

## apg_plan_mgmt.capture_plan_baselines

Mengambil rencana eksekusi kueri yang dihasilkan oleh pengoptimisasi untuk setiap pernyataan SQL dan menyimpannya dalam tampilan `dba_plans`. Secara default, jumlah maksimum rencana yang dapat disimpan adalah 10.000 seperti yang ditentukan oleh parameter `apg_plan_mgmt.max_plans`. Untuk informasi referensi, lihat apg_plan_mgmt.max_plans.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom. Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| off | automatic | Terapkan pengaturan ini baik di tingkat sesi atau dalam kelompok parameter untuk menangkap rencana yang digunakan dua kali atau lebih. |
| | manual | Terapkan pengaturan ini di tingkat sesi atau dalam grup parameter untuk menangkap rencana yang digunakan satu kali atau beberapa kali. |
| | off | Menonaktifkan pengambilan rencana. |

Untuk informasi selengkapnya, lihat Mengambil rencana eksekusi Aurora PostgreSQL.

## apg_plan_mgmt.plan_capture_threshold

Menentukan ambang batas sehingga jika total biaya rencana eksekusi kueri di bawah ambang batas, rencana tidak akan diambil dalam tampilan `apg_plan_mgmt.dba_plans`.

Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| 0 | 0 - 1.79769e+308 | Menetapkan ambang batas total biaya eksekusi rencana kueri `apg_plan_mgmt` untuk mengambil rencana. |

Untuk informasi selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

## apg_plan_mgmt.explain_hashes

Menentukan apakah `EXPLAIN [ANALYZE]` menunjukkan `sql_hash` dan `plan_hash` pada akhir output-nya. Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| 0 | 0 (aktif) | EXPLORE tidak menampilkan sql_hash dan plan_hash tanpa opsi hashes true. |
| | 1 (nonaktif) | EXPLORE menampilkan sql_hash dan plan_hash tanpa opsi hashes true. |

## apg_plan_mgmt.log_plan_enforcement_result

Menentukan apakah hasil harus dicatat untuk melihat apakah rencana yang dikelola QPM digunakan dengan benar. Ketika rencana generik yang disimpan digunakan, tidak akan ada catatan yang ditulis dalam file log. Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| none | none | Tidak menunjukkan hasil pemberlakuan rencana apa pun dalam file log. |
| | on_error | Hanya menampilkan hasil pemberlakuan rencana dalam file log ketika QPM gagal menggunakan rencana terkelola. |
| | all | Menampilkan semua hasil pemberlakuan rencana dalam file log termasuk keberhasilan dan kegagalan. |

## apg_plan_mgmt.max_databases

Menentukan jumlah maksimum basis data pada instans Penulis klaster DB Aurora PostgreSQL Anda yang dapat menggunakan manajemen rencana kueri. Secara default, hingga 10 basis data dapat menggunakan manajemen rencana kueri. Jika Anda memiliki lebih dari 10 basis data pada instans, Anda dapat mengubah nilai pengaturan ini. Untuk mengetahui jumlah basis data pada instans

tertentu, hubungkan ke instans menggunakan `psql`. Kemudian, gunakan metacommand psql, `\l`, untuk menampilkan daftar basis data.

Mengubah nilai parameter ini akan mengharuskan Anda mem-boot ulang instans agar pengaturan diterapkan.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| 10 | 10-2147483647 | Jumlah maksimum basis data yang dapat menggunakan manajemen rencana kueri pada instans. |

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom.

### apg_plan_mgmt.max_plans

Menetapkan jumlah maksimum pernyataan SQL yang dapat dipertahankan oleh manajer rencana kueri dalam tampilan `apg_plan_mgmt.dba_plans`. Sebaiknya atur parameter ini ke `10000` atau lebih tinggi untuk semua versi Aurora PostgreSQL.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom. Mengubah nilai parameter ini akan mengharuskan Anda mem-boot ulang instans agar pengaturan diterapkan.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| 10000 | 10-2147483647 | Jumlah maksimum rencana yang dapat disimpan dalam tampilan `apg_plan_mgmt.dba_plans` .<br><br>Default untuk Aurora PostgreSQL versi 10 dan yang lebih lama adalah 1000. |

Untuk informasi selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

## apg_plan_mgmt.plan_hash_version

Menentukan kasus penggunaan yang dapat dicakup oleh perhitungan plan_hash sesuai rancangannya. Versi `apg_plan_mgmt.plan_hash_version` yang lebih tinggi mencakup semua fungsionalitas versi yang lebih rendah. Misalnya, versi 3 mencakup kasus penggunaan yang didukung oleh versi 2.

Perubahan pada nilai parameter ini harus diikuti dengan panggilan ke `apg_plan_mgmt.validate_plans('update_plan_hash')`. Parameter ini memperbarui nilai plan_hash di setiap basis data dengan apg_plan_mgmt diinstal dan entri dalam tabel rencana. Untuk informasi selengkapnya, lihat Memvalidasi rencana

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| 1 | 1 | Perhitungan plan_hash default. |
| | 2 | Perhitungan plan_hash yang dimodifikasi untuk dukungan multi-skema. |
| | 3 | Perhitungan plan_hash yang dimodifikasi untuk dukungan multi-skema dan dukungan tabel yang dipartisi. |
| | 4 | Perhitungan plan_hash yang dimodifikasi untuk operator paralel dan untuk mendukung simpul terwujud. |

## apg_plan_mgmt.plan_retention_period

Menentukan jumlah hari untuk mempertahankan rencana dalam tampilan `apg_plan_mgmt.dba_plans`, dan setelah itu, rencana tersebut akan dihapus secara otomatis. Secara default, rencana dihapus ketika 32 hari telah berlalu sejak rencana tersebut terakhir digunakan (kolom `last_used` dalam tampilan `apg_plan_mgmt.dba_plans`). Anda dapat mengubah pengaturan ini ke angka apa pun, 1 dan lebih tinggi.

Mengubah nilai parameter ini akan mengharuskan Anda mem-boot ulang instans agar pengaturan diterapkan.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| 32 | 1-2147483647 | Jumlah hari maksimum sejak rencana terakhir digunakan sebelum dihapus. |

Untuk informasi selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

### apg_plan_mgmt.unapproved_plan_execution_threshold

Menentukan ambang batas biaya yang jika tidak tercapai, rencana yang Tidak Disetujui dapat digunakan oleh pengoptimisasi. Secara default ambang batas ini adalah 0, sehingga pengoptimisasi tidak akan menjalankan rencana yang Tidak Disetujui. Jika parameter ini diatur ke ambang batas biaya yang cukup rendah seperti 100, tidak akan ada overhead pemberlakuan rencana pada rencana sederhana (trivial). Anda juga dapat mengatur parameter ini ke nilai yang sangat besar seperti 10000000 menggunakan gaya manajemen rencana reaktif. Hal ini memungkinkan pengoptimisasi menggunakan semua rencana yang dipilih tanpa overhead pemberlakuan rencana. Namun, ketika rencana buruk ditemukan, Anda dapat menandainya secara manual sebagai “ditolak” sehingga tidak akan digunakan di lain waktu.

Nilai parameter ini merepresentasikan perkiraan biaya untuk menjalankan rencana yang diberikan. Jika rencana yang Tidak Disetujui di bawah perkiraan biaya tersebut, pengoptimisasi akan menggunakannya untuk pernyataan SQL. Anda dapat melihat rencana yang diambil dan statusnya (Disetujui, Tidak Disetujui) di tampilan `dba_plans`. Untuk mempelajari selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| 0 | 0-2147483647 | Perkiraan biaya rencana yang jika tidak tercapai, akan membuat rencana yang Tidak Disetujui digunakan. |

Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL.

## apg_plan_mgmt.use_plan_baselines

Menentukan bahwa pengoptimisasi harus menggunakan salah satu rencana Disetujui yang diambil dan disimpan dalam tampilan `apg_plan_mgmt.dba_plans`. Secara default, parameter ini tidak aktif (false), sehingga menyebabkan pengoptimisasi menggunakan rencana berbiaya minimum yang dihasilkannya tanpa penilaian lebih lanjut. Mengaktifkan parameter ini (mengaturnya ke true) akan memaksa pengoptimisasi untuk memilih rencana eksekusi kueri untuk pernyataan dari acuan dasar (baseline) rencananya. Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL. Untuk menemukan gambar yang menguraikan proses ini, lihat Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom. Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
|---|---|---|
| false | true | Menggunakan rencana yang Disetujui, Pilihan, atau Tidak Disetujui dari `apg_plan_mgmt.dba_plans` . Jika tidak ada yang memenuhi semua kriteria evaluasi untuk pengoptimisasi, maka pengoptimisasi dapat menggunakan rencana berbiaya minimum yang dihasilkannya sendiri. Untuk informasi selengkapnya, lihat Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan. |
| | false | Menggunakan rencana berbiaya minimum yang dihasilkan oleh pengoptimisasi. |

Anda dapat mengevaluasi waktu respons dari berbagai rencana yang diambil dan mengubah status rencana, sesuai kebutuhan. Untuk informasi selengkapnya, lihat Meningkatkan rencana kueri PostgreSQL Aurora.

## auto_explain.hashes

Menentukan apakah output auto_explain menampilkan sql_hash dan plan_hash. Mengubah nilai parameter ini tidak memerlukan boot ulang.

| Default | Nilai yang diizinkan | Deskripsi |
| --- | --- | --- |
| 0 (nonaktif) | 0 (nonaktif) | Hasil `auto_explain` tidak menunjukkan `sql_hash` dan `plan_hash` . |
| | 1 (aktif) | Hasil `auto_explain` menunjukkan `sql_hash` dan `plan_hash` . |

## Referensi fungsi untuk manajemen rencana kueri Aurora PostgreSQL

Ekstensi `apg_plan_mgmt` menyediakan fungsi-fungsi berikut.

Fungsi

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

### apg_plan_mgmt.copy_outline

Menyalin hash rencana dan outline rencana SQL tertentu ke hash rencana dan outline rencana SQL target, sehingga akan menimpa hash rencana dan outline rencana target. Fungsi ini tersedia dalam `apg_plan_mgmt` versi 2.3 dan rilis yang lebih tinggi.

Sintaksis

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
```

```
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

Nilai yang ditampilkan

Menampilkan 0 ketika penyalinan berhasil. Menampilkan pengecualian untuk input yang tidak valid.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `source_sql_hash` | ID `sql_hash` yang terkait dengan `plan_hash` yang akan disalin ke kueri target. |
| `source_plan_hash` | ID `plan_hash` yang akan disalin ke kueri target. |
| `target_sql_hash` | ID `sql_hash` kueri yang akan diperbarui dengan hash rencana dan outline rencana sumber. |
| `target_plan_hash` | ID `plan_hash` kueri yang akan diperbaru i dengan hash rencana dan outline rencana sumber. |
| `force_update_target_plan_hash` | (Opsional) ID `target_plan_hash` kueri diperbarui meskipun rencana sumber tidak dapat direproduksi untuk `target_sql_hash` . Ketika diatur ke true, fungsi ini dapat digunakan untuk menyalin rencana ke seluruh skema yang memiliki nama dan kolom relasi yang konsisten. |

Catatan penggunaan

Fungsi ini memungkinkan Anda untuk menyalin hash rencana dan outline rencana yang menggunakan petunjuk ke pernyataan lain yang serupa, sehingga Anda tidak perlu menggunakan pernyataan petunjuk inline pada setiap peristiwa dalam pernyataan target. Jika kueri target yang

diperbarui menghasilkan rencana yang tidak valid, fungsi ini akan memunculkan kesalahan dan mengembalikan pembaruan yang dicoba.

### apg_plan_mgmt.delete_plan

Menghapus rencana terkelola.

Sintaksis

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

Nilai yang ditampilkan

Menampilkan 0 jika penghapusan berhasil atau -1 jika penghapusan gagal.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. |

### apg_plan_mgmt.evolve_plan_baselines

Memverifikasi apakah rencana yang sudah disetujui lebih cepat atau apakah rencana yang diidentifikasi oleh pengoptimal kueri sebagai rencana berbiaya minimum yang lebih cepat.

Sintaksis

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

Nilai yang ditampilkan

Jumlah rencana yang tidak lebih cepat dari rencana terbaik yang disetujui.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. Gunakan NULL untuk mengindik asikan semua rencana yang memiliki nilai ID `sql_hash` yang sama. |
| `min_speed up_factor` | Faktor percepatan minimum adalah seberapa kali lebih cepat suatu rencana daripada rencana lain yang sudah disetujui agar rencana tersebut dapat disetujui. Alternatifnya, faktor ini dapat berupa berapa kali lebih lambat agar rencana harus ditolak atau dinonaktifkan.<br><br>Ini adalah nilai float positif. |
| `action` | Tindakan yang harus dilakukan oleh fungsi tersebut. Nilai yang valid mencakup hal berikut. Huruf besar/kecil tidak berpengaruh.<br><br>• `'disable'` – Menonaktifkan setiap rencana yang tidak memenuhi faktor percepatan minimum.<br>• `'approve'` – Mengaktifkan setiap rencana yang memenuhi faktor percepatan minimum dan menetapkan statusnya menjadi `approved`.<br>• `'reject'` – Untuk setiap rencana yang tidak memenuhi faktor percepatan minimum, menetapkan statusnya menjadi `rejected`.<br>• NULL – Fungsi hanya menampilkan jumlah rencana yang tidak memiliki keuntungan performa karena tidak memenuhi faktor percepatan minimum. |

Catatan penggunaan

Tetapkan rencana ke status disetujui, ditolak, atau dinonaktifkan berdasarkan apakah waktu perencanaan plus eksekusi lebih cepat daripada rencana yang disetujui berdasarkan faktor yang dapat Anda tetapkan. Parameter tindakan dapat diatur ke `'approve'` atau `'reject'` untuk

secara otomatis menyetujui atau menolak rencana yang memenuhi kriteria performa. Alternatifnya, parameter dapat diatur ke '' (string kosong) untuk melakukan eksperimen performa dan membuat laporan, tetapi tidak mengambil tindakan.

Anda dapat mencegah fungsi `apg_plan_mgmt.evolve_plan_baselines` dijalankan kembali secara sia-sia untuk rencana tempat fungsi ini baru-baru ini dijalankan. Untuk melakukannya, batasi rencana hanya untuk rencana yang baru-baru ini dibuat yang belum disetujui. Alternatifnya, Anda dapat mencegah fungsi `apg_plan_mgmt.evolve_plan_baselines` dijalankan pada rencana yang disetujui yang memiliki stempel waktu `last_verified` baru-baru ini.

Lakukan eksperimen performa untuk membandingkan waktu perencanaan dan eksekusi untuk setiap rencana secara relatif terhadap rencana lain dalam acuan dasar (baseline). Dalam beberapa kasus, hanya ada satu rencana untuk satu pernyataan dan rencana ini disetujui. Dalam kasus seperti itu, bandingkan waktu perencanaan dan eksekusi rencana dengan waktu perencanaan dan eksekusi tanpa menggunakan rencana.

Keuntungan (atau kerugian) tambahan dari setiap rencana dicatat di tampilan `apg_plan_mgmt.dba_plans` dalam kolom `total_time_benefit_ms`. Jika nilainya positif, ada keuntungan performa yang dapat diukur untuk memasukkan rencana ini dalam acuan dasar.

Selain mengumpulkan waktu perencanaan dan eksekusi dari setiap rencana kandidat, kolom `last_verified` pada tampilan `apg_plan_mgmt.dba_plans` diperbarui dengan `current_timestamp`. Stempel waktu `last_verified` dapat digunakan agar fungsi ini tidak lagi dijalankan pada rencana yang performanya baru-baru ini diverifikasi.

## apg_plan_mgmt.get_explain_plan

Menghasilkan teks dari pernyataan `EXPLAIN` untuk pernyataan SQL yang ditentukan.

Sintaksis

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

Nilai yang ditampilkan

Menampilkan statistik runtime untuk pernyataan SQL yang ditentukan. Gunakan tanpa `explainOptionList` untuk menampilkan rencana `EXPLAIN` sederhana.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. |
| `explainOptionList` | Daftar opsi penjelasan yang dipisahkan dengan koma. Nilai yang valid termasuk `'analyze'`, `'verbose'`, `'buffers'`, `'hashes'`, dan `'format json'`. Jika `explainOptionList` adalah NULL atau string kosong (''), fungsi ini menghasilkan pernyataan `EXPLAIN`, tanpa statistik. |

Catatan penggunaan

Untuk `explainOptionList`, Anda dapat menggunakan salah satu dari opsi yang sama yang akan Anda gunakan dengan pernyataan `EXPLAIN`. Pengoptimal Aurora PostgreSQL menggabungkan daftar opsi yang Anda berikan ke pernyataan `EXPLAIN`.

## apg_plan_mgmt.plan_last_used

Menampilkan tanggal `last_used` untuk rencana yang ditetapkan dari memori bersama.

> **Note**
>
> Nilai dalam memori bersama selalu terbaru pada instans DB primer dalam klaster DB. Nilai hanya secara berkala di-flush ke kolom `last_used` dalam tampilan `apg_plan_mgmt.dba_plans`.

Sintaksis

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
```

```
    plan_hash
)
```

Nilai yang ditampilkan

Menampilkan tanggal `last_used`.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. |

## apg_plan_mgmt.reload

Memuat ulang rencana ke dalam memori bersama dari tampilan `apg_plan_mgmt.dba_plans`.

Sintaksis

```
apg_plan_mgmt.reload()
```

Nilai yang ditampilkan

Tidak ada.

Parameter

Tidak ada.

Catatan penggunaan

Memanggil `reload` untuk situasi-situasi berikut:

- Gunakan untuk me-refresh memori bersama replika hanya baca, dan bukan menunggu adanya rencana baru untuk disebarkan ke replika.
- Gunakan setelah mengimpor rencana terkelola.

## apg_plan_mgmt.set_plan_enabled

Mengaktifkan atau menonaktifkan rencana terkelola.

Sintaksis

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

Nilai yang ditampilkan

Menampilkan 0 jika pengaturan berhasil atau -1 jika pengaturan gagal.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. |
| `enabled` | Nilai Boolean benar atau salah:<br>• Nilai `true` mengaktifkan rencana.<br>• Nilai `false` menonaktifkan rencana. |

## apg_plan_mgmt.set_plan_status

Mengatur status rencana terkelola ke `Approved`, `Unapproved`, `Rejected`, atau `Preferred`.

Sintaksis

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
```

```
    status
)
```

Nilai yang ditampilkan

Menampilkan 0 jika pengaturan berhasil atau -1 jika pengaturan gagal.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. |
| `status` | String dengan salah satu nilai berikut:<br>• `'Approved'`<br>• `'Unapproved'`<br>• `'Rejected'`<br>• `'Preferred'`<br><br>Huruf besar/kecil yang Anda gunakan tidak berpengaruh, tetapi nilai status menggunakan huruf besar di awal dalam tampilan `apg_plan_mgmt.dba_plans` . Untuk informasi selengkapnya tentang nilai-nilai tersebut, lihat `status` dalam Referensi untuk tampilan apg_plan_mgmt.dba_plans untuk Aurora PostgreSQL Compatible Edition. |

## apg_plan_mgmt.update_plans_last_used

Segera memperbarui tabel rencana dengan tanggal `last_used` yang disimpan dalam memori bersama.

Sintaksis

```
apg_plan_mgmt.update_plans_last_used()
```

Nilai yang ditampilkan

Tidak ada.

Parameter

Tidak ada.

Catatan penggunaan

Panggil `update_plans_last_used` untuk memastikan kueri terhadap kolom `dba_plans.last_used` menggunakan informasi terbaru. Jika tanggal `last_used` tidak segera diperbarui, proses latar belakang akan memperbarui tabel rencana dengan tanggal `last_used` sekali setiap jam (secara default).

Misalnya, jika pernyataan dengan `sql_hash` tertentu mulai berjalan lambat, Anda dapat menentukan rencana mana yang dieksekusi untuk pernyataan tersebut sejak regresi performa dimulai. Untuk melakukannya, pertama-tama lakukan flushing data dalam memori bersama ke disk sehingga tanggal `last_used` menjadi terkini, lalu jalankan kueri untuk semua rencana `sql_hash` dari pernyataan yang memiliki regresi performa. Dalam kueri, pastikan tanggal `last_used` lebih besar dari atau sama dengan tanggal saat regresi performa dimulai. Kueri tersebut mengidentifikasi rencana atau kumpulan rencana yang mungkin bertanggung jawab atas regresi performa. Anda dapat menggunakan `apg_plan_mgmt.get_explain_plan` dengan `explainOptionList` yang diatur ke `verbose, hashes`. Anda juga dapat menggunakan `apg_plan_mgmt.evolve_plan_baselines` untuk menganalisis rencana dan rencana alternatif apa pun yang mungkin beperforma lebih baik.

Fungsi `update_plans_last_used` hanya berpengaruh pada instans DB primer klaster DB.

## apg_plan_mgmt.validate_plans

Memvalidasi bahwa pengoptimal masih dapat membuat ulang rencana. Pengoptimal memvalidasi rencana `Approved`, `Unapproved`, dan `Preferred`, baik ketika rencana tersebut diaktifkan maupun dinonaktifkan. Rencana `Rejected` tidak divalidasi. Secara opsional, Anda dapat menggunakan fungsi `apg_plan_mgmt.validate_plans` untuk menghapus atau menonaktifkan rencana yang tidak valid.

Sintaksis

```
apg_plan_mgmt.validate_plans(
```

```
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
    action)
```

Nilai yang ditampilkan

Jumlah rencana tidak valid.

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `sql_hash` | ID `sql_hash` pernyataan SQL terkelola milik rencana. |
| `plan_hash` | ID `plan_hash` rencana terkelola. Gunakan NULL untuk merujuk pada semua rencana dengan nilai ID `sql_hash` yang sama. |
| `action` | Tindakan yang harus dilakukan fungsi untuk rencana yang tidak valid. Nilai string yang valid mencakup hal berikut. Huruf besar/kecil tidak berpengaruh.<br>• `'disable'` – Setiap rencana yang tidak valid dinonaktifkan.<br>• `'delete'` – Setiap rencana yang tidak valid dihapus.<br>• `'update_plan_hash'` – Memperbarui ID `plan_hash` untuk rencana yang tidak dapat direproduksi dengan tepat. Hal ini juga memungkinkan Anda memperbaiki rencana dengan menulis ulang SQL. Anda kemudian dapat mendaftarkan rencana yang baik sebagai rencana `Approved` untuk SQL asli.<br>• NULL – Fungsi hanya menampilkan jumlah rencana yang tidak valid. Tidak ada tindakan lain yang dilakukan.<br>• '' – String kosong menghasilkan pesan yang menunjukkan jumlah rencana yang valid dan tidak valid.<br><br>Nilai lainnya diperlakukan seperti string kosong. |

Catatan penggunaan

Gunakan form `validate_plans(action)` untuk memvalidasi semua rencana terkelola untuk semua pernyataan terkelola dalam tampilan `apg_plan_mgmt.dba_plans` secara keseluruhan.

Gunakan form `validate_plans(sql_hash, plan_hash, action)` untuk memvalidasi rencana terkelola yang ditentukan dengan `plan_hash`, untuk pernyataan terkelola yang ditentukan dengan `sql_hash`.

Gunakan form `validate_plans(sql_hash, NULL, action)` untuk memvalidasi semua rencana terkelola untuk laporan terkelola yang ditentukan dengan `sql_hash`.

## Referensi untuk tampilan apg_plan_mgmt.dba_plans untuk Aurora PostgreSQL Compatible Edition

Kolom informasi rencana dalam tampilan `apg_plan_mgmt.dba_plans` mencakup hal-hal berikut ini.

| Kolom dba_plans | Deskripsi |
| --- | --- |
| `cardinality_error` | Pengukuran kesalahan antara kardinalitas yang diperkirakan dan kardinalitas yang sebenarnya. Kardinalitas adalah jumlah baris tabel yang akan diproses oleh rencana. Jika kesalahan pada kardinalitas berukuran besar, hal tersebut akan menambah kemungkinan bahwa rencana tersebut tidak optimal. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines. |
| `compatibility_level` | Parameter ini menunjukkan kapan rencana kueri terakhir divalidasi. Dalam Aurora PostgreSQL versi 12.19, 13.15, 14.12, 15.7, 16.3, dan yang lebih baru, ini menampilkan nomor versi Aurora. Untuk versi sebelumnya, ini menampilkan fitur nomor versi tertentu.<br><br>**Note**<br>Pertahankan nilai parameter ini pada pengatura n defaultnya. Aurora PostgreSQL secara otomatis menetapkan dan memperbarui nilai ini. |

| Kolom dba_plans | Deskripsi |
| --- | --- |
| `created_by` | Pengguna terautentikasi (`session_user` ) yang membuat rencana. |
| `enabled` | Indikator apakah rencana diaktifkan atau dinonaktifkan. Semua rencana diaktifkan secara default. Anda dapat menonaktifkan rencana untuk mencegah pengoptimisasi menggunakannya. Untuk mengubah nilai ini, gunakan fungsi apg_plan_mgmt.set_plan_enabled. |
| `environment_variables` | Parameter dan nilai Grand Unified Configuration (GUC) PostgreSQL yang telah diganti oleh pengoptimisasi pada saat rencana diambil. |
| `estimated_startup_cost` | Perkiraan biaya penyiapan pengoptimisasi sebelum pengoptimisasi menghasilkan baris tabel. |
| `estimated_total_cost` | Perkiraan biaya pengoptimisasi untuk mengirimkan baris tabel akhir. |
| `execution_time_benefit_ms` | Keuntungan waktu eksekusi dalam milidetik untuk mengaktifkan rencana. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines. |
| `execution_time_ms` | Perkiraan waktu dalam milidetik bahwa rencana akan berjalan. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines. |
| `has_side_effects` | Nilai yang menunjukkan bahwa pernyataan SQL adalah pernyataan bahasa manipulasi data (DML) atau pernyataan SELECT yang berisi fungsi VOLATILE. |

| Kolom dba_plans | Deskripsi |
| --- | --- |
| `last_used` | Nilai ini diperbarui ke tanggal saat ini setiap kali rencana dijalankan atau jika rencana adalah rencana berbiaya minimum dari pengoptimisasi kueri. Nilai ini disimpan di memori bersama dan dialirkan secara berkala ke disk. Untuk mendapatk an up-to-date nilai terbanyak, baca tanggal dari memori bersama dengan memanggil fungsi `apg_plan_mgmt.plan _last_used(sql_hash, plan_hash)` alih-alih membaca `last_used` nilainya. Untuk informasi tambahan, lihat parameter apg_plan_mgmt.plan_retention_period. |
| `last_validated` | Tanggal dan waktu terbaru saat terverifikasi bahwa rencana dapat dibuat ulang dengan fungsi apg_plan_mgmt.validate_plan s atau fungsi apg_plan_mgmt.evolve_plan_baselines. |
| `last_verified` | Tanggal dan waktu terbaru saat rencana terverifikasi sebagai rencana beperforma terbaik untuk parameter yang ditentukan oleh fungsi apg_plan_mgmt.evolve_plan_baselines. |
| `origin` | Cara pengambilan rencana dengan parameter apg_plan_ mgmt.capture_plan_baselines. Nilai yang valid mencakup hal berikut:<br><br>`M` – Rencana diambil dengan pengambilan rencana manual.<br><br>`A` – Rencana diambil dengan pengambilan rencana manual. |
| `param_list` | Nilai parameter yang diteruskan ke pernyataan jika pernyataan ini merupakan pernyataan yang disiapkan. |
| `plan_created` | Tanggal dan waktu rencana yang dibuat. |
| `plan_hash` | Pengidentifikasi rencana. Kombinasi dari `plan_hash` dan `sql_hash` mengidentifikasi rencana tertentu secara unik. |

| Kolom dba_plans | Deskripsi |
| --- | --- |
| `plan_outline` | Representasi rencana yang digunakan untuk membuat ulang rencana eksekusi sebenarnya dan yang bersifat independen dari basis data. Operator di hierarki menunjukkan operator yang muncul di output EXPLAIN. |
| `planning_time_ms` | Waktu sebenarnya untuk menjalankan perencana, dalam milidetik. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines. |
| `queryId` | Hash pernyataan, sebagaimana dihitung oleh ekstensi `pg_stat_statements` . Ini bukan pengidentifikasi stabil atau database independen karena tergantung pada pengidentifikasi objek (). OIDs Nilainya adalah `0` jika `compute_query_id` adalah `off` saat mengambil rencana kueri. |
| `sql_hash` | Nilai hash teks pernyataan SQL, yang dinormalisasikan dengan menghapus literal. |
| `sql_text` | Teks lengkap pernyataan SQL. |

| Kolom dba_plans | Deskripsi |
| --- | --- |
| `status` | Status rencana, yang menentukan cara pengoptimisasi menggunakan rencana. Nilai yang valid mencakup hal berikut:<br>• `Approved` – Rencana yang dapat digunakan yang dapat dipilih pengoptimisasi untuk dijalankan. Pengoptimisasi menjalankan rencana berbiaya paling rendah dari kumpulan rencana yang disetujui (acuan dasar) untuk pernyataa n terkelola. Untuk mengatur ulang rencana ke disetujui, gunakan fungsi apg_plan_mgmt.evolve_plan_baselines.<br>• `Unapproved` – Rencana yang diambil dan belum Anda verifikasi untuk digunakan. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.<br>• `Rejected` – Rencana yang tidak akan digunakan oleh pengoptimal. Untuk informasi selengkapnya, lihat Menolak atau menonaktifkan rencana yang lebih lambat.<br>• `Preferred` – Rencana yang telah Anda tentukan sebagai rencana yang lebih disukai untuk digunakan untuk pernyataan terkelola.<br>Jika rencana berbiaya minimum dari pengoptimisasi bukanlah rencana yang disetujui atau yang lebih disukai, Anda dapat mengurangi overhead penerapan rencana. Untuk melakukan nya, buat subkumpulan dari rencana yang disetujui, yaitu `Preferred` . Ketika rencana berbiaya minimum dari pengoptimisasi bukanlah rencana `Approved`, rencana `Preferred` akan dipilih sebelum rencana `Approved`.<br>Untuk mengatur ulang rencana ke `Preferred` , gunakan fungsi apg_plan_mgmt.set_plan_status. |
| `stmt_name` | Nama pernyataan SQL di dalam pernyataan PREPARE. Nilai ini adalah string kosong untuk pernyataan yang disiapkan tanpa nama. Nilai ini adalah NULL untuk pernyataan yang tidak disiapkan. |

| Kolom dba_plans | Deskripsi |
| --- | --- |
| `total_time_benefit_ms` | Keuntungan total waktu dalam milidetik untuk mengaktifkan rencana ini. Nilai ini mempertimbangkan waktu perencanaan dan waktu eksekusi.<br><br>Jika nilai ini negatif, ada kerugian untuk mengaktifkan rencana ini. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines. |

## Fitur lanjutan dalam Manajemen Rencana Kueri

Berikut ini Anda dapat menemukan informasi tentang Aurora Postgre SQL Query Plan Management () fitur canggih: QPM

Topik

- Mengambil rencana eksekusi Aurora PostgreSQL di Replika
- Mendukung partisi tabel

### Mengambil rencana eksekusi Aurora PostgreSQL di Replika

QPM (Query Plan Management) memungkinkan Anda mengambil rencana kueri yang dihasilkan oleh Replika Aurora dan menyimpannya di instans DB primer dari klaster Aurora DB. Anda dapat mengumpulkan rencana kueri dari semua Replika Aurora, dan memelihara serangkaian rencana optimal dalam tabel persisten terpusat pada instans primer. Anda kemudian dapat menerapkan rencana ini pada Replika lain jika diperlukan. Ini membantu Anda menjaga stabilitas rencana eksekusi dan meningkatkan performa kueri di seluruh klaster DB dan versi mesin.

Topik

- Prasyarat
- Mengelola pengambilan rencana untuk Replika Aurora
- Pemecahan Masalah

Prasyarat

Aktifkan **`capture_plan_baselines parameter`** di Replika Aurora - Atur parameter `capture_plan_baselines` ke otomatis atau manual untuk mengambil rencana di Replika Aurora. Untuk informasi selengkapnya, lihat apg_plan_mgmt.capture_plan_baselines.

Instal ekstensi postgres_fdw - Anda harus menginstal ekstensi wrapper data asing `postgres_fdw` untuk mengambil rencana di Replika Aurora. Jalankan perintah berikut di setiap basis data untuk menginstal ekstensi ini.

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

Mengelola pengambilan rencana untuk Replika Aurora

Aktifkan pengambilan rencana untuk Replika Aurora

Anda harus memiliki hak akses `rds_superuser` untuk membuat atau menghapus Pengambilan Rencana di Replika Aurora. Untuk informasi selengkapnya tentang peran dan izin pengguna, lihat Memahami peran dan izin PostgreSQL.

Untuk mengambil rencana, panggil fungsi apg_plan_mgmt.create_replica_plan_capture dalam instans DB penulis, seperti yang ditunjukkan berikut ini:

```
postgres=> CALL apg_plan_mgmt.create_replica_plan_capture('endpoint', 'password');
```

- endpoint - Titik akhir penulis Aurora Global Database atau cluster_endpoint menyediakan dukungan failover untuk Plan Capture di Aurora Replicas.

  Untuk informasi lebih lanjut tentang titik akhir penulis Aurora Global Database, lihat. Melihat titik akhir database global Amazon Aurora

  Untuk informasi selengkapnya tentang titik akhir klaster, lihatTitik akhir cluster untuk Amazon Aurora.
- password - Kami menyarankan Anda untuk mengikuti panduan di bawah ini saat membuat kata sandi untuk meningkatkan keamanan:
  - Kata sandi harus berisi setidaknya 8 karakter.
  - Kata sandi harus berisi setidaknya satu huruf besar, satu huruf kecil, dan satu angka.
  - Kata sandi harus memiliki setidaknya satu karakter khusus (?, !, #, <, >, *, dan sebagainya).

> **Note**
>
> Jika Anda mengubah titik akhir, kata sandi, atau nomor port, Anda harus menjalankan `apg_plan_mgmt.create_replica_plan_capture()` lagi dengan titik akhir dan kata sandi untuk menginisialisasi ulang pengambilan paket. Jika tidak, pengambilan rencana dari Replika Aurora akan gagal.

Nonaktifkan pengambilan rencana untuk Replika Aurora

Anda dapat menonaktifkan parameter `capture_plan_baselines` di Replika Aurora dengan menetapkan nilainya ke `off` dalam grup Parameter.

Hapus pengambilan rencana untuk Replika Aurora

Anda dapat sepenuhnya menghapus Pengambilan Rencana di Replika Aurora tetapi pastikan untuk menonaktifkannya terlebih dahulu. Untuk menghapus pengambilan rencana, panggil `apg_plan_mgmt.remove_replica_plan_capture` seperti yang ditunjukkan berikut:

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

Anda harus memanggil apg_plan_mgmt.create_replica_plan_capture () lagi untuk mengaktifkan pengambilan rencana di Aurora Replicas dengan titik akhir dan kata sandi.

Pemecahan Masalah

Di bagian berikut ini, Anda dapat menemukan ide dan solusi pemecahan masalah jika rencana tidak diambil di Replika Aurora seperti yang diharapkan.

- Pengaturan parameter - Periksa apakah parameter `capture_plan_baselines` diatur ke nilai yang tepat untuk mengaktifkan pengambilan rencana.
- Ekstensi **`postgres_fdw`** diinstal - Gunakan kueri berikut untuk memeriksa apakah `postgres_fdw` diinstal.

  ```
  postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
  ```

- create_replica_plan_capture() dipanggil - Gunakan perintah berikut untuk memeriksa apakah pemetaan pengguna sudah ada. Jika tidak, panggil `create_replica_plan_capture()` untuk menginisialisasi fitur ini.

```
postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- Endpoint dan nomor port - Periksa apakah titik akhir dan nomor port sesuai. Tidak akan ada pesan kesalahan yang ditampilkan jika nilai-nilai ini salah.

  Gunakan perintah berikut untuk memverifikasi titik akhir yang digunakan dalam create() dan untuk memeriksa di basis data mana titik akhir tersebut berada:

```
postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- reload() - Anda harus memanggil apg_plan_mgmt.reload() setelah memanggil apg_plan_mgmt.delete_plan() di Replika Aurora untuk membuat fungsi penghapusan efektif. Hal ini akan memastikan bahwa perubahan telah berhasil diterapkan.
- Kata sandi - Anda harus memasukkan kata sandi di create_replica_plan_capture() sesuai pedoman yang disebutkan. Jika tidak, Anda akan menerima pesan kesalahan. Untuk informasi selengkapnya, lihat Mengelola pengambilan rencana untuk Replika Aurora. Gunakan kata sandi lain yang sesuai dengan persyaratan.
- Koneksi Lintas Wilayah - Pengambilan rencana di Replika Aurora juga didukung di basis data global Aurora, yang memungkinkan instans penulis dan Replika Aurora berada di wilayah yang berbeda. Pastikan Anda menggunakan titik akhir penulis Aurora Global Database untuk mempertahankan konektivitas setelah peristiwa failover atau switchover. Untuk informasi selengkapnya tentang titik akhir Basis Data Global Aurora, lihat Melihat titik akhir Database Global Amazon Aurora. Instans penulis dan Replika lintas Wilayah harus dapat berkomunikasi menggunakan Peering VPC. Untuk informasi selengkapnya, lihat Peering VPC. Jika failover lintas Wilayah terjadi, Anda harus mengonfigurasi ulang titik akhir ke titik akhir klaster DB primer yang baru.

> **Note**
>
> Saat menggunakan titik akhir cluster alih-alih titik akhir penulis Aurora Global Database, Anda harus memperbarui titik akhir cluster setelah melakukan operasi failover atau switchover global.

## Mendukung partisi tabel

Aurora Postgre SQL Query Plan Management (QPM) mendukung partisi tabel deklaratif dalam versi berikut:

- 15.3 dan versi 15 yang lebih tinggi
- 14.8 dan versi 14 yang lebih tinggi
- 13.11 dan versi 13 yang lebih tinggi

Untuk informasi selengkapnya, lihat Pemartisian Tabel.

Topik

- Mengatur partisi tabel
- Mengambil rencana untuk partisi tabel
- Memberlakukan rencana partisi tabel
- Konvensi Penamaan

### Mengatur partisi tabel

Untuk mengatur partisi tabel di Aurora Postgre SQLQPM, lakukan hal berikut:

1. Atur `apg_plan_mgmt.plan_hash_version` ke 3 atau lebih di grup parameter klaster DB.
2. Temukan basis data yang menggunakan Manajemen Rencana Kueri dan memiliki entri dalam tampilan `apg_plan_mgmt.dba_plans`.
3. Panggil `apg_plan_mgmt.validate_plans('update_plan_hash')` untuk memperbarui nilai `plan_hash` dalam tabel rencana.
4. Ulangi langkah 2-3 untuk semua basis data dengan Manajemen Rencana Kueri diaktifkan yang memiliki entri dalam tampilan `apg_plan_mgmt.dba_plans`.

Untuk informasi selengkapnya tentang parameter ini, lihat Referensi parameter untuk manajemen rencana kueri Aurora PostgreSQL.

### Mengambil rencana untuk partisi tabel

DalamQPM, rencana yang berbeda dibedakan berdasarkan `plan_hash` nilainya. Untuk memahami perubahan `plan_hash`, Anda harus terlebih dahulu memahami jenis rencana yang serupa.

Kombinasi metode akses, nama indeks tanpa digit, dan nama partisi tanpa digit, yang terakumulasi pada tingkat simpul Append harus konstan agar rencana dianggap sama. Partisi spesifik yang diakses dalam rencana tidaklah signifikan. Dalam contoh berikut, tabel `tbl_a` dibuat dengan 4 partisi.

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

Rencana berikut dianggap sama karena metode pemindaian tunggal digunakan untuk memindai `tbl_a` terlepas dari jumlah partisi yang dicari kueri.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
```

```
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

Tiga rencana berikut juga dianggap sama karena pada tingkat induk, metode akses, nama indeks tanpa digit, dan nama partisi tanpa digit adalah `SeqScan tbl_a`, `IndexScan (i_idx) tbl_a`.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                               QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                               QUERY PLAN
```

```
------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                              QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

Terlepas dari urutan dan jumlah kemunculan yang berbeda dalam partisi turunan, metode akses, nama indeks tanpa digit, dan nama partisi tanpa digit akan tetap konstan pada tingkat induk untuk masing-masing rencana di atas.

Namun, rencana akan dianggap berbeda jika salah satu dari persyaratan berikut terpenuhi:

- Setiap metode akses tambahan digunakan dalam rencana.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;
```

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

- Salah satu metode akses dalam rencana tidak digunakan lagi.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                             QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

- Indeks yang terkait dengan metode indeks diubah.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                            QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
```

```
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

Memberlakukan rencana partisi tabel

Rencana yang disetujui untuk tabel yang dipartisi diberlakukan dengan kesesuaian posisi. Rencana tidak bersifat spesifik untuk partisi, dan dapat diberlakukan pada partisi selain rencana yang direferensikan dalam kueri asli. Rencana juga memiliki kemampuan untuk diberlakukan untuk kueri yang mengakses jumlah partisi yang berbeda dari garis besar asli yang disetujui.

Misalnya, jika garis besar yang disetujui adalah untuk rencana berikut:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

Kemudian, rencana ini dapat diterapkan pada SQL kueri yang merujuk 2, 4, atau lebih partisi juga. Rencana yang mungkin muncul dari skenario ini untuk akses partisi 2 dan 4 adalah:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
```

```
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a4 tbl_a_4
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
```

```
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

Pertimbangkan rencana lain yang disetujui dengan metode akses berbeda untuk setiap partisi:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

Dalam hal ini, rencana apa pun yang dibaca dari dua partisi akan gagal diberlakukan. Kecuali jika semua kombinasi (metode akses, nama indeks) dari rencana yang disetujui dapat digunakan, rencana tidak dapat diberlakukan. Misalnya, rencana berikut memiliki hash rencana yang berbeda dan rencana yang disetujui tidak dapat diberlakukan dalam kasus ini:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                             QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
```

```
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

Konvensi Penamaan

QPMUntuk menerapkan rencana dengan tabel partisi deklaratif, Anda harus mengikuti aturan penamaan khusus untuk tabel induk, partisi tabel, dan indeks:

- Nama tabel induk — Nama-nama ini harus berbeda menurut huruf atau karakter khusus, dan bukan hanya dengan angka. Misalnya, tA, tB, dan tC adalah nama yang dapat diterima untuk tabel induk terpisah, sedangkan t1, t2, dan t3 tidak.
- Nama tabel partisi individu — Partisi dari induk yang sama harus berbeda satu sama lain dengan digit saja. Misalnya, nama partisi tA yang dapat diterima bisa berupa tA1, tA2, atau t1A, t2A, atau bahkan dengan beberapa angka.

  Perbedaan lain (huruf, karakter khusus) tidak akan menjamin pemberlakuan rencana.
- Nama indeks — Dalam hierarki tabel partisi, pastikan bahwa semua indeks memiliki nama yang unik. Ini berarti bahwa bagian non-numerik dari nama harus berbeda. Misalnya, jika Anda memiliki tabel dipartisi bernama `tA` dengan indeks bernama`tA_col1_idx1`, Anda tidak dapat memiliki indeks lain bernama. `tA_col1_idx2` Namun, Anda dapat memiliki indeks yang dipanggil

`tA_a_col1_idx2` karena bagian non-numerik dari nama itu unik. Aturan ini berlaku untuk indeks yang dibuat pada tabel induk dan tabel partisi individu.

Jika konvensi penamaan di atas tidak dipatuhi, pemberlakuan rencana yang disetujui dapat gagal. Contoh berikut mengilustrasikan pemberlakuan yang gagal:

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;

                           QUERY PLAN
-----------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 1000;

                           QUERY PLAN
-----------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

Meskipun kedua rencana mungkin tampak identik, `Plan Hash` nilainya berbeda karena nama tabel anak. Nama tabel bervariasi menurut karakter alfa, bukan hanya digit yang mengarah ke kegagalan penegakan hukum.

# Menangani ekstensi dan pembungkus data asing

Untuk memperluas fungsionalitas ke klaster Aurora PostgreSQL-Compatible Edition, Anda dapat menginstal dan menggunakan berbagai ekstensi PostgreSQL. Misalnya, jika kasus penggunaan Anda memerlukan entri data intensif di seluruh tabel yang sangat besar, Anda dapat menginstal ekstensi `pg_partman` untuk mempartisi data dan dengan demikian menyebarkan beban kerja.

> **Note**
>
> Mulai versi Aurora PostgreSQL 14.5, Aurora PostgreSQL mendukung Ekstensi Bahasa Tepercaya untuk PostgreSQL. Fitur ini diimplementasikan sebagai ekstensi `pg_tle`, yang dapat Anda tambahkan ke Aurora PostgreSQL Anda. Dengan ekstensi ini, pengembang dapat membuat ekstensi PostgreSQL mereka sendiri di lingkungan yang aman yang menyederhanakan persyaratan penyiapan dan konfigurasi, serta sebagian besar pengujian awal untuk ekstensi baru. Untuk informasi selengkapnya, lihat Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL.

Dalam beberapa kasus, daripada menginstal ekstensi, Anda dapat menambahkan modul tertentu ke daftar `shared_preload_libraries` dalam grup parameter klaster DB khusus klaster DB Aurora PostgreSQL. Biasanya, grup parameter klaster DB default hanya memuat `pg_stat_statements`, tetapi beberapa modul lain tersedia untuk ditambahkan ke daftar. Misalnya, Anda dapat menambahkan kemampuan penjadwalan dengan menambahkan modul `pg_cron`, seperti yang dijelaskan dalam Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL. Sebagai contoh lain, Anda dapat men-log rencana eksekusi kueri dengan memuat modul `auto_explain`. Untuk mempelajari selengkapnya, lihat Mencatat log rencana eksekusi kueri di pusat pengetahuan AWS.

Ekstensi yang memberikan akses ke data eksternal lebih khusus dikenal sebagai pembungkus data asing (FDW). Sebagai contoh, ekstensi `oracle_fdw` memungkinkan klaster DB Aurora PostgreSQL bekerja dengan basis data Oracle.

Anda juga dapat menentukan dengan tepat ekstensi yang dapat diinstal pada instans DB Aurora PostgreSQL, dengan mencantumkannya dalam parameter `rds.allowed_extensions`. Untuk informasi selengkapnya, lihat Membatasi instalasi ekstensi PostgreSQL.

Berikut ini, Anda dapat menemukan informasi tentang pengaturan dan penggunaan beberapa ekstensi, modul, dan FDW yang tersedia untuk Aurora PostgreSQL. Untuk menyederhanakan,

ini semua akan disebut sebagai "ekstensi". Anda dapat menemukan daftar ekstensi yang dapat digunakan dengan versi Aurora PostgreSQL yang tersedia saat ini, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

- Mengelola objek besar dengan modul lo
- Mengelola data spasial dengan GIS ekstensi Post
- Mengelola partisi PostgreSQL dengan ekstensi pg_partman
- Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL
- Menggunakan pgAudit untuk mencatat aktivitas database
- Menggunakan pglogical untuk menyinkronkan data di seluruh instans
- Bekerja dengan basis data Oracle menggunakan ekstensi oracle_fdw
- Bekerja dengan database SQL Server dengan menggunakan ekstensi tds_fdw

## Menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk Postgre SQL

Menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk PostgreSQL, Anda dapat mendelegasikan manajemen ekstensi kepada pengguna yang tidak perlu menjadi file. `rds_superuser` Dengan dukungan ekstensi yang didelegasikan ini, peran baru yang `rds_extension` disebut dibuat dan Anda harus menetapkan ini kepada pengguna untuk mengelola ekstensi lainnya. Peran ini dapat membuat, memperbarui, dan menjatuhkan ekstensi.

Anda dapat menentukan ekstensi yang dapat diinstal pada instans Aurora Postgre SQL DB Anda, dengan mencantumkannya di parameter. `rds.allowed_extensions` Untuk informasi selengkapnya, lihat Menggunakan SQL ekstensi Postgre dengan Amazon RDS untuk Postgre. SQL

Anda dapat membatasi daftar ekstensi yang tersedia yang dapat dikelola oleh pengguna dengan `rds_extension` peran menggunakan `rds.allowed_delegated_extensions` parameter.

Dukungan ekstensi yang didelegasikan tersedia dalam versi berikut:

- Semua versi yang lebih tinggi
- 15.5 dan versi 15 yang lebih tinggi
- 14.10 dan versi 14 yang lebih tinggi
- 13.13 dan 13 versi yang lebih tinggi
- 12.17 dan versi 12 yang lebih tinggi

Topik


- Mengaktifkan dukungan ekstensi delegasi ke pengguna
- Konfigurasi yang digunakan dalam dukungan ekstensi yang didelegasikan Aurora untuk Postgre SQL
- Mematikan dukungan untuk ekstensi yang didelegasikan
- Manfaat menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora
- Batasan dukungan ekstensi yang didelegasikan Aurora untuk Postgre SQL
- Izin diperlukan untuk ekstensi tertentu
- Pertimbangan Keamanan
- Jatuhkan kaskade ekstensi dinonaktifkan
- Contoh ekstensi yang dapat ditambahkan menggunakan dukungan ekstensi yang didelegasikan


## Mengaktifkan dukungan ekstensi delegasi ke pengguna

Anda harus melakukan hal berikut untuk mengaktifkan dukungan ekstensi delegasi kepada pengguna:

1. Berikan **`rds_extension`** peran kepada pengguna - Connect ke database sebagai `rds_superuser` dan jalankan perintah berikut:

   ```
   Postgres => grant rds_extension to user_name;
   ```

2. Mengatur daftar ekstensi yang tersedia bagi pengguna yang didelegasikan untuk dikelola — `rds.allowed_delegated_extensions` Memungkinkan Anda menentukan subset ekstensi yang tersedia menggunakan parameter `rds.allowed_extensions` cluster DB. Anda dapat melakukan ini di salah satu level berikut:

   - Di cluster atau kelompok parameter instance, melalui AWS Management Console atauAPI. Untuk informasi selengkapnya, lihat .
   - Gunakan perintah berikut di tingkat database:

     ```
     alter database database_name set rds.allowed_delegated_extensions =
      'extension_name_1,
                        extension_name_2,...extension_name_n';
     ```

   - Gunakan perintah berikut di tingkat pengguna:

```
alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                    extension_name_2,...extension_name_n';
```

> **Note**
>
> Anda tidak perlu me-restart database setelah mengubah parameter `rds.allowed_delegated_extensions` dinamis.

3. Izinkan akses ke pengguna yang didelegasikan ke objek yang dibuat selama proses pembuatan ekstensi - Ekstensi tertentu membuat objek yang memerlukan izin tambahan untuk diberikan sebelum pengguna dengan `rds_extension` peran dapat mengaksesnya. `rds_superuser`Harus memberikan akses pengguna yang didelegasikan ke objek tersebut. Salah satu opsi adalah menggunakan pemicu peristiwa untuk secara otomatis memberikan izin kepada pengguna yang didelegasikan. Untuk informasi lebih lanjut, lihat contoh pemicu peristiwa diMematikan dukungan untuk ekstensi yang didelegasikan.

## Konfigurasi yang digunakan dalam dukungan ekstensi yang didelegasikan Aurora untuk Postgre SQL

| Nama Konfigsi | Deskripsi | nilai default | Catatan | Siapa yang dapat memodifikasi atau memberikan izin |
|---|---|---|---|---|
| `rds.aed_dted_esions` | Parameter ini membatasi ekstensi yang dapat dikelola peran rds_extension dalam database. Itu harus merupakan bagian dari rds.allowed_extensions. | string kosong | • Secara default, parameter ini adalah string kosong, yang berarti bahwa tidak ada ekstensi yang telah didelegasikan kepada pengguna dengan`rds_extension` . | rds_superuser |

| Nama Konfigsi | Deskripsi | nilai default | Catatan | Siapa yang dapat memodifikasi atau memberikan izin |
|---|---|---|---|---|
| | | | • Ekstensi apa pun yang didukung dapat ditambahkan jika pengguna memiliki izin untuk melakukannya. Untuk melakukan ini, atur `rds.allowed_delegated_extensions` parameter ke string nama ekstensi yang dipisahkan koma. Dengan menambahkan daftar ekstensi ke parameter ini, Anda secara eksplisit mengidentifikasi ekstensi yang dapat dipasang oleh pengguna dengan `rds_extension` peran tersebut.<br>• Ketika diatur ke*, itu berarti bahwa semua ekstensi | |

| Nama Konfigsi | Deskripsi | nilai default | Catatan | Siapa yang dapat memodifikasi atau memberikan izin |
|---|---|---|---|---|
| | | | yang terdaftar di `rds_allowed_extensions` didelegasikan kepada pengguna dengan `rds_extension` peran.<br><br>Untuk mempelajari lebih lanjut tentang pengaturan parameter ini, lihatMengaktifkan dukungan ekstensi delegasi kepengguna. | |

| Nama Konfigurasi | Deskripsi | nilai default | Catatan | Siapa yang dapat memodifikasi atau memberikan izin |
|---|---|---|---|---|
| `rds.allowed_extensions` | Parameter ini memungkinkan pelanggan membatasi ekstensi yang dapat dipasang di instans Aurora SQL Postgre DB. Untuk informasi selengkapnya, lihat Membatasi pemasangan ekstensi SQL Postgre | "*" | Secara default, parameter ini diatur ke “*”, yang berarti bahwa semua ekstensi yang didukung RDS untuk Postgre dan SQL Aurora Postgre SQL diizinkan untuk dibuat oleh pengguna dengan hak istimewa yang diperlukan.<br><br>Kosong berarti tidak ada ekstensi yang dapat dipasang di instans Aurora SQL Postgre DB. | administrator |

| Nama Konfigsi | Deskripsi | nilai default | Catatan | Siapa yang dapat memodifikasi atau memberikan izin |
|---|---|---|---|---|
| `rds-delegated_nsionow_drascad` | Parameter ini mengontrol kemampuan pengguna `rds_extension` untuk menjatuhkan ekstensi menggunakan opsi kaskade. | off | Secara default, `rds-delegated_extensionallow_drop_cascade` diatur ke `off`. Ini berarti bahwa pengguna dengan tidak `rds_extension` diizinkan untuk menjatuhkan ekstensi menggunakan opsi kaskade.<br><br>Untuk memberikan kemampuan itu, `rds.delegated_extensionallow_drop_cascade` parameter harus diatur keon. | rds_superuser |

## Mematikan dukungan untuk ekstensi yang didelegasikan

Mematikan sebagian

Pengguna yang didelegasikan tidak dapat membuat ekstensi baru tetapi masih dapat memperbarui ekstensi yang ada.

- Atur ulang `rds.allowed_delegated_extensions` ke nilai default di grup parameter cluster DB.
- Gunakan perintah berikut di tingkat database:

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- Gunakan perintah berikut di tingkat pengguna:

```
alter user user_name reset rds.allowed_delegated_extensions;
```

### Mematikan sepenuhnya

Mencabut `rds_extension` peran dari pengguna akan mengembalikan pengguna ke izin standar. Pengguna tidak dapat lagi membuat, memperbarui, atau menjatuhkan ekstensi.

```
postgres => revoke rds_extension from user_name;
```

### Contoh pemicu peristiwa

Jika Anda ingin mengizinkan pengguna yang didelegasikan `rds_extension` untuk menggunakan ekstensi yang memerlukan izin pengaturan pada objek mereka yang dibuat oleh pembuatan ekstensi, Anda dapat menyesuaikan contoh pemicu peristiwa di bawah ini dan hanya menambahkan ekstensi yang Anda inginkan agar pengguna yang didelegasikan memiliki akses ke fungsionalitas penuh. Pemicu peristiwa ini dapat dibuat pada template1 (template default), oleh karena itu semua database yang dibuat dari template1 akan memiliki pemicu peristiwa itu. Ketika pengguna yang didelegasikan menginstal ekstensi, pemicu ini akan secara otomatis memberikan kepemilikan pada objek yang dibuat oleh ekstensi.

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;
```

```
BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';

        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );

        databaseowner = (
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
 schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
 schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_gaz TO
 %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_lex TO
 %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_rules
 TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
            EXECUTE format('ALTER TEXT SEARCH DICTIONARY %I.intdict OWNER TO %I;',
 schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname, databaseowner);
```

```
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA topology TO
 %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA topology TO %I
 WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE ON SCHEMA topology TO %I WITH GRANT OPTION;',
 databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE create_ext();
```

## Manfaat menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora

Dengan menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk PostgreSQL, Anda mendelegasikan manajemen ekstensi dengan aman kepada pengguna yang tidak memiliki peran tersebut. `rds_superuser` Fitur ini memberikan manfaat sebagai berikut:

- Anda dapat dengan mudah mendelegasikan manajemen ekstensi kepada pengguna pilihan Anda.
- Ini tidak membutuhkan `rds_superuser` peran.
- Menyediakan kemampuan untuk mendukung kumpulan ekstensi yang berbeda untuk database yang berbeda di cluster DB yang sama.

## Batasan dukungan ekstensi yang didelegasikan Aurora untuk Postgre SQL

- Objek yang dibuat selama proses pembuatan ekstensi mungkin memerlukan hak istimewa tambahan agar ekstensi berfungsi dengan baik.

## Izin diperlukan untuk ekstensi tertentu

Untuk membuat, menggunakan, atau memperbarui ekstensi berikut, pengguna yang didelegasikan harus memiliki hak istimewa yang diperlukan pada fungsi, tabel, dan skema berikut.

| Ekste yang mem an kepe an atau izin | Fungsi | Tabel | Skema | Kamus Pencarian Teks | Komentar |
|---|---|---|---|---|---|
| addre tanda er_da s | | us_gaz, us_lex, us_lex, i.us_rule s | | | |
| amch | bt_index_check, bt_index_ parent_check | | | | |
| dict_i | | | | intdict | |
| pg_p n | | custom_ti me_partisi, part_config, part_config_sub | | | |
| pg_st tatem | | | | | |
| Posti GIS | st_tileamplop | spatial_ref_sys | | | |
| postg aster | | | | | |
| postg opolo | | topologi, lapisan | topologi | | pengguna yang didelegas ikan Harus |

| Ekste yang mem an kepe an atau izin | Fungsi | Tabel | Skema | Kamus Pencarian Teks | Komentar |
|---|---|---|---|---|---|
| | | | | | menjadi pemilik database |
| log_f | create_fo reign_tab le_for_log_file | | | | |
| rds_t | role_pass word_encr yption_type | | | | |
| postg iger_ oder | | geocode_s ettings_default, geocode_s ettings | harimau | | |
| pg_fr acem | pg_freespace | | | | |
| pg_vi lity | pg_visibility | | | | |

## Pertimbangan Keamanan

Perlu diingat bahwa pengguna dengan `rds_extension` peran akan dapat mengelola ekstensi di semua database tempat mereka memiliki hak istimewa terhubung. Jika tujuannya adalah agar pengguna yang didelegasikan mengelola ekstensi pada satu database, praktik yang baik adalah mencabut semua hak istimewa dari publik di setiap database, kemudian secara eksplisit memberikan hak istimewa koneksi untuk database spesifik tersebut kepada pengguna delegasi.

Ada beberapa ekstensi yang dapat memungkinkan pengguna untuk mengakses informasi dari beberapa database. Pastikan pengguna yang Anda berikan `rds_extension` memiliki kemampuan lintas basis data sebelum menambahkan ekstensi ini`rds.allowed_delegated_extensions`. Misalnya, `postgres_fdw` dan `dblink` menyediakan fungsionalitas untuk kueri di seluruh database pada instance yang sama atau instance jarak jauh. `log_fdw`membaca file log mesin postgres, yang untuk semua database dalam instance, berpotensi berisi kueri lambat atau pesan kesalahan dari beberapa database. `pg_cron`memungkinkan menjalankan pekerjaan latar belakang terjadwal pada instans DB dan dapat mengonfigurasi pekerjaan untuk dijalankan di database yang berbeda.

## Jatuhkan kaskade ekstensi dinonaktifkan

Kemampuan untuk menjatuhkan ekstensi dengan opsi kaskade oleh pengguna dengan `rds_extension` peran dikendalikan oleh `rds.delegated_extension_allow_drop_cascade` parameter. Secara default, `rds-delegated_extension_allow_drop_cascade` diatur ke `off`. Ini berarti bahwa pengguna dengan `rds_extension` peran tidak diizinkan untuk menjatuhkan ekstensi menggunakan opsi kaskade seperti yang ditunjukkan pada kueri di bawah ini.

```
DROP EXTENSION CASCADE;
```

Karena ini akan secara otomatis menjatuhkan objek yang bergantung pada ekstensi, dan pada gilirannya semua objek yang bergantung pada objek tersebut. Mencoba menggunakan opsi kaskade akan menghasilkan kesalahan.

Untuk memberikan kemampuan itu, `rds.delegated_extension_allow_drop_cascade` parameter harus diatur keon.

Mengubah parameter `rds.delegated_extension_allow_drop_cascade` dinamis tidak memerlukan restart database. Anda dapat melakukan ini di salah satu level berikut:

- Di cluster atau kelompok parameter instance, melalui AWS Management Console atauAPI.
- Menggunakan perintah berikut di tingkat database:

  ```
  alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

- Menggunakan perintah berikut di tingkat pengguna:

  ```
  alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

## Contoh ekstensi yang dapat ditambahkan menggunakan dukungan ekstensi yang didelegasikan

- `rds_tools`

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- `amcheck`

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- `pg_freespacemap`

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- `pg_visibility`

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- `postgres_fdw`

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

## Mengelola objek besar dengan modul lo

Modul lo (ekstensi) adalah untuk pengguna database dan pengembang yang bekerja dengan SQL database Postgre melalui JDBC atau driver. ODBC Keduanya JDBC dan ODBC mengharapkan database untuk menangani penghapusan objek besar ketika referensi ke mereka berubah. Namun, Postgre SQL tidak bekerja seperti itu. Postgre SQL tidak berasumsi bahwa suatu objek harus dihapus ketika referensinya berubah. Hasilnya adalah bahwa objek tetap berada di disk, tidak direferensikan. Ekstensi lo mencakup fungsi yang Anda gunakan untuk memicu perubahan referensi untuk menghapus objek jika diperlukan.

> **Tip**
>
> Untuk menentukan apakah database Anda dapat memperoleh manfaat dari ekstensi lo, gunakan `vacuumlo` utilitas untuk memeriksa objek besar yatim piatu. Untuk mendapatkan hitungan objek tak berinduk tanpa mengambil tindakan apa pun, jalankan utilitas dengan opsi `-n` (no-op). Untuk mempelajari caranya, lihat vacuumlo utility berikut.

Modul lo tersedia untuk Aurora Postgre SQL 13.7, 12.11, 11.16, 10.21 dan versi minor yang lebih tinggi.

Untuk menginstal modul (ekstensi), Anda memerlukan hak istimewa `rds_superuser`. Menginstal ekstensi lo akan menambahkan berikut ke basis data Anda:

- `lo`— Ini adalah tipe data objek besar (lo) yang dapat Anda gunakan untuk objek besar biner (BLOBs) dan objek besar lainnya. Tipe data `lo` adalah domain dari jenis data `oid`. Dengan kata lain, ini adalah ID objek dengan batasan opsional. Untuk informasi selengkapnya, lihat Pengidentifikasi objek dalam dokumentasi SQL Postgre. Secara sederhana, Anda dapat menggunakan tipe `lo` data untuk membedakan kolom database Anda yang menyimpan referensi objek besar dari pengidentifikasi objek lain (OIDs).

- `lo_manage` – Ini adalah fungsi yang dapat digunakan dalam pemicu pada kolom tabel yang berisi referensi objek besar. Setiap kali Anda menghapus atau mengubah nilai yang mereferensikan objek besar, pemicu akan memutuskan tautan objek (`lo_unlink`) dari referensinya. Gunakan pemicu pada kolom hanya jika kolom tersebut adalah referensi basis data tunggal ke objek besar.

Untuk informasi lebih lanjut tentang modul objek besar, lihat lo di dokumentasi PostgreSQL.

## Menginstal ekstensi lo

Sebelum menginstal ekstensi lo, pastikan bahwa Anda memiliki hak istimewa `rds_superuser`.

Untuk menginstal ekstensi lo.

1. Gunakan `psql` untuk terhubung ke instans DB utama cluster Aurora SQL Postgre DB Anda.

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   Saat diminta, masukkan kata sandi Anda. Klien `psql` menghubungkan dan menampilkan basis data koneksi administratif default, `postgres=>`, sebagai perintah.

2. Instal ekstensi seperti berikut.

   ```
   postgres=> CREATE EXTENSION lo;
   CREATE EXTENSION
   ```

Anda kini dapat menggunakan jenis data `lo` untuk menentukan kolom dalam tabel Anda. Misalnya, Anda dapat membuat tabel (`images`) yang berisi data citra raster. Anda dapat menggunakan jenis data `lo` untuk kolom `raster`, seperti yang ditunjukkan pada contoh berikut, yang membuat tabel.

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

## Menggunakan fungsi pemicu lo_manage untuk menghapus objek

Anda dapat menggunakan fungsi `lo_manage` dalam pemicu pada `lo` atau kolom objek besar lainnya untuk membersihkan (dan mencegah objek tak berinduk) ketika `lo` diperbarui atau dihapus.

Untuk menyiapkan pemicu pada kolom yang mereferensikan objek besar

- Lakukan salah satu langkah berikut:
  - Buat DELETE pemicu BEFORE UPDATE OR pada setiap kolom untuk berisi referensi unik ke objek besar, menggunakan nama kolom untuk argumen.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

  - Terapkan pemicu hanya ketika kolom sedang diperbarui.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

Fungsi `lo_manage` pemicu hanya berfungsi saat memasukkan atau menghapus data kolom, bergantung pada bagaimana Anda mendefinisikan pemicu. Ini tidak berpengaruh ketika Anda melakukan operasi `DROP` atau `TRUNCATE` pada basis data. Berarti Anda harus menghapus kolom objek dari tabel apa pun sebelum hilang, untuk mencegah pembuatan objek tak berinduk.

Misalnya, anggaplah Anda ingin menghilangkan basis data yang berisi tabel `images`. Anda menghapus kolom sebagai berikut.

```
postgres=> DELETE FROM images COLUMN raster
```

Dengan asumsi bahwa fungsi `lo_manage` didefinisikan pada kolom itu untuk menangani penghapusan, Anda sekarang dapat dengan aman menghilangkan tabel.

### Menghapus benda-benda besar yatim piatu menggunakan **`vacuumlo`**

`vacuumlo`Utilitas mengidentifikasi dan menghapus objek besar yatim piatu dari database. Utilitas ini telah tersedia sejak Postgre SQL 9.1.24. Jika pengguna database Anda secara rutin bekerja dengan objek besar, kami sarankan Anda menjalankan `vacuumlo` sesekali untuk membersihkan objek besar yatim piatu.

Sebelum menginstal ekstensi lo, Anda dapat menggunakan `vacuumlo` untuk menilai apakah cluster Aurora Postgre SQL DB Anda dapat memperoleh manfaat. Untuk melakukannya, jalankan `vacuumlo` dengan opsi `-n` (no-op) untuk menunjukkan apa yang akan dihapus, seperti yang ditunjukkan berikut:

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

Seperti yang ditunjukkan output, objek besar tak berinduk tidak menjadi masalah untuk basis data khusus ini.

Untuk informasi lebih lanjut tentang utilitas ini, lihat `vacuumlo`di dokumentasi PostgreSQL.

## Memahami cara kerja **`vacuumlo`**

`vacuumlo`Perintah menghapus objek besar yatim piatu (LOs) dari SQL database Postgre Anda tanpa mempengaruhi atau bertentangan dengan tabel pengguna Anda.

Perintah berfungsi sebagai berikut:

1. `vacuumlo`dimulai dengan membuat tabel sementara yang berisi semua Object IDs (OIDs) dari objek besar dalam database Anda.
2. `vacuumlo`kemudian memindai melalui setiap kolom dalam database yang menggunakan tipe data `oid` atau`lo`. Jika `vacuumlo` menemukan pencocokan OID di kolom ini, itu menghapus OID dari tabel sementara. `vacuumlo`memeriksa hanya kolom yang diberi nama khusus `oid` atau`lo`, bukan domain berdasarkan jenis ini.
3. Entri yang tersisa dalam tabel sementara mewakili yatim piatuLOs, yang `vacuumlo` kemudian dihapus dengan aman.

## Meningkatkan **`vacuumlo`** kinerja

Anda berpotensi dapat meningkatkan kinerja `vacuumlo` dengan meningkatkan ukuran batch menggunakan `-l` opsi. Ini memungkinkan `vacuumlo` untuk memproses lebih banyak LOs sekaligus.

Jika sistem Anda memiliki memori yang cukup dan Anda dapat mengakomodasi tabel sementara sepenuhnya dalam memori, meningkatkan `temp_buffers` pengaturan di tingkat database dapat meningkatkan kinerja. Hal ini memungkinkan tabel untuk berada sepenuhnya dalam memori, yang dapat meningkatkan kinerja secara keseluruhan.

Berikut query memperkirakan ukuran tabel sementara:

```
SELECT
    pg_size_pretty(SUM(pg_column_size(oid))) estimated_lo_temp_table_size
FROM
    pg_largeobject_metadata;
```

### Pertimbangan untuk benda besar

Berikut ini Anda dapat menemukan beberapa pertimbangan penting untuk dicatat ketika bekerja dengan objek besar:

- `Vacuumlo`adalah satu-satunya solusi karena saat ini tidak ada metode lain untuk menghapus yatim piatuLOs.
- Alat seperti pglogis, replikasi logis asli, dan AWS DMS yang menggunakan teknologi replikasi tidak mendukung replikasi objek besar.
- Saat merancang skema database Anda, hindari menggunakan objek besar bila memungkinkan dan pertimbangkan untuk menggunakan tipe data alternatif seperti `bytea` sebagai gantinya.
- Jalankan `vacuumlo` secara teratur, setidaknya setiap minggu, untuk mencegah masalah dengan yatim piatuLOs.
- Gunakan pemicu dengan `lo_manage` fungsi pada tabel yang menyimpan objek besar untuk membantu mencegah anak yatim piatu dibuatLOs.

## Mengelola data spasial dengan GIS ekstensi Post

Post GIS adalah perpanjangan dari Postgre SQL untuk menyimpan dan mengelola informasi spasial. Untuk mempelajari lebih lanjut tentang PostGIS, lihat GISPost.net.

Dimulai dengan versi 10.5, Postgre SQL mendukung pustaka libprotobuf 1.3.0 yang digunakan oleh Post GIS untuk bekerja dengan data ubin vektor kotak peta.

Menyiapkan GIS ekstensi Post membutuhkan `rds_superuser` hak istimewa. Kami menyarankan Anda membuat pengguna (peran) untuk mengelola GIS ekstensi Post dan data spasial Anda. GISEkstensi Post dan komponen terkaitnya menambahkan ribuan fungsi ke PostgreSQL. Pertimbangkan untuk membuat GIS ekstensi Post dalam skemanya sendiri jika itu masuk akal untuk kasus penggunaan Anda. Contoh berikut menunjukkan cara menginstal ekstensi dalam basis data tersendiri, tetapi ini tidak wajib dilakukan.

Topik

- Langkah 1: Buat pengguna (peran) untuk mengelola GIS ekstensi Post
- Langkah 2: Muat GIS ekstensi Post
- Langkah 3: Transfer kepemilikan skema ekstensi
- Langkah 4: Transfer kepemilikan GIS tabel Post
- Langkah 5: Menguji ekstensi
- Langkah 6: Tingkatkan GIS ekstensi Post
- Versi GIS ekstensi posting
- Upgrade Post GIS 2 ke Post GIS 3


## Langkah 1: Buat pengguna (peran) untuk mengelola GIS ekstensi Post

Pertama, sambungkan ke instans Postgre SQL DB Anda RDS sebagai pengguna yang memiliki `rds_superuser` hak istimewa. Jika tidak mengubah nama default saat menyiapkan instans, Anda akan terhubung sebagai `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

Buat peran terpisah (pengguna) untuk mengelola GIS ekstensi Post.

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

Berikan hak akses `rds_superuser` kepada peran ini agar dapat menginstal ekstensi.

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

Buat database untuk digunakan untuk GIS artefak Post Anda. Langkah ini bersifat opsional. Atau Anda dapat membuat skema dalam database pengguna Anda untuk GIS ekstensi Post, tetapi ini juga tidak diperlukan.

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

Berikan semua hak akses `gis_admin` pada basis data `lab_gis`.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

Keluar dari sesi dan sambungkan kembali ke instans Postgre SQL DB Anda RDS sebagai. `gis_admin`

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

Lanjutkan menyiapkan ekstensi seperti yang dijelaskan pada langkah berikutnya.

## Langkah 2: Muat GIS ekstensi Post

GISEkstensi Post mencakup beberapa ekstensi terkait yang bekerja sama untuk menyediakan fungsionalitas geospasial. Tergantung pada kasus penggunaannya, Anda mungkin tidak memerlukan semua ekstensi yang dibuat pada langkah ini.

Gunakan `CREATE EXTENSION` pernyataan untuk memuat GIS ekstensi Post.

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

Anda dapat memverifikasi hasil dengan menjalankan SQL kueri yang ditampilkan dalam contoh berikut, yang mencantumkan ekstensi dan pemiliknya.

```
SELECT n.nspname AS "Name",
```

```
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

## Langkah 3: Transfer kepemilikan skema ekstensi

Gunakan ALTER SCHEMA pernyataan untuk mentransfer kepemilikan skema ke `gis_admin` peran.

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

Anda dapat mengonfirmasi perubahan kepemilikan dengan menjalankan SQL kueri berikut. Bisa juga dengan menggunakan metacommand \dn dari baris perintah psql.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

      List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
(4 rows)
```

## Langkah 4: Transfer kepemilikan GIS tabel Post

> **Note**
>
> Jangan mengubah kepemilikan GIS fungsi Post. Operasi yang tepat dan peningkatan Post di masa depan GIS memerlukan fungsi-fungsi ini untuk mempertahankan kepemilikan asli. Untuk informasi selengkapnya tentang GIS izin Post, lihat Keamanan SQLPostgre.

Gunakan fungsi berikut untuk mentransfer kepemilikan GIS tabel Post ke `gis_admin` peran. Jalankan pernyataan berikut dari perintah psql untuk membuat fungsinya.

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

Selanjutnya, jalankan kueri berikut untuk menjalankan fungsi `exec` yang nantinya akan menjalankan pernyataan dan mengubah izin.

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

## Langkah 5: Menguji ekstensi

Agar Anda tidak harus menentukan nama skema, tambahkan skema `tiger` ke jalur pencarian menggunakan perintah berikut.

```
SET search_path=public,tiger;
SET
```

Uji `tiger` skema dengan menggunakan SELECT pernyataan berikut.

```
SELECT address, streetname, streettypeabbrev, zip
```

```
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

Untuk mempelajari lebih lanjut tentang ekstensi ini, lihat Tiger Geocoder di dokumentasi PostGIS.

Uji akses ke skema `topology` menggunakan pernyataan `SELECT` berikut. Tindakan ini akan memanggil fungsi `createtopology` untuk mendaftarkan objek topologi baru (my_new_topo) dengan pengidentifikasi referensi spasial (26986) dan toleransi default (0,5) yang ditentukan. Untuk mempelajari lebih lanjut, lihat CreateTopologydi GIS dokumentasi Post.

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
(1 row)
```

### Langkah 6: Tingkatkan GIS ekstensi Post

Setiap rilis baru Postgre SQL mendukung satu atau lebih versi GIS ekstensi Post yang kompatibel dengan rilis itu. Memutakhirkan SQL mesin Postgre ke versi baru tidak secara otomatis meningkatkan ekstensi Post. GIS Sebelum memutakhirkan SQL mesin Postgre, Anda biasanya meng-upgrade Post GIS ke versi terbaru yang tersedia untuk versi Postgre saat ini. SQL Untuk detailnya, lihat Versi GIS ekstensi posting.

Setelah upgrade SQL mesin Postgre, Anda kemudian meng-upgrade GIS ekstensi Post lagi, ke versi yang didukung untuk versi mesin SQL Postgre yang baru ditingkatkan. Untuk informasi lebih lanjut tentang memutakhirkan SQL mesin Postgre, lihat. Menguji peningkatan klaster DB produksi ke versi mayor baru

Anda dapat memeriksa pembaruan versi GIS ekstensi Post yang tersedia di cluster SQL DB Aurora Postgre Anda RDS untuk instans Postgre DB saja. Untuk melakukannya, jalankan perintah berikut. Fungsi ini tersedia dengan Post GIS 2.5.0 dan versi yang lebih tinggi.

```
SELECT postGIS_extensions_upgrade();
```

Jika aplikasi Anda tidak mendukung GIS versi Post terbaru, Anda dapat menginstal versi Post yang lebih lama GIS yang tersedia di versi utama Anda sebagai berikut.

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

Jika Anda ingin meningkatkan ke GIS versi Post tertentu dari versi yang lebih lama, Anda juga dapat menggunakan perintah berikut.

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

Tergantung pada versi sebelum peningkatan, Anda mungkin perlu menggunakan fungsi ini lagi. Hasil dari menjalankan fungsi pertama akan menentukan perlu atau tidaknya fungsi peningkatan lain. Misalnya, ini adalah kasus untuk meningkatkan dari Post GIS 2 ke Post GIS 3. Untuk informasi selengkapnya, lihat Upgrade Post GIS 2 ke Post GIS 3.

Jika Anda memutakhirkan ekstensi ini untuk mempersiapkan peningkatan versi utama SQL mesin Postgre, Anda dapat melanjutkan tugas awal lainnya. Untuk informasi selengkapnya, lihat Menguji peningkatan klaster DB produksi ke versi mayor baru.

## Versi GIS ekstensi posting

Kami menyarankan Anda menginstal versi semua ekstensi seperti Post GIS seperti yang tercantum dalam versi Ekstensi untuk Aurora Postgre SQL -Compatible Edition di Catatan Rilis untuk Aurora Postgre. SQL Untuk mendapatkan daftar versi yang tersedia dalam rilis Anda, gunakan perintah berikut.

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

Anda dapat menemukan informasi versi di bagian berikut di Catatan Rilis untuk Aurora Postgre: SQL

- Versi ekstensi untuk Aurora Postgre 14 SQL
- Versi ekstensi untuk Aurora SQL Postgre -Compatible Edition 13
- Versi ekstensi untuk Aurora SQL Postgre -Compatible Edition 12
- Versi ekstensi untuk Aurora SQL Postgre -Compatible Edition 11
- Versi ekstensi untuk Aurora SQL Postgre -Compatible Edition 10
- Versi ekstensi untuk Aurora SQL Postgre -Compatible Edition 9.6

## Upgrade Post GIS 2 ke Post GIS 3

Dimulai dengan versi 3.0, fungsionalitas Post GIS raster sekarang merupakan ekstensi terpisah,`postgis_raster`. Ekstensi ini memiliki jalur instalasi dan peningkatan tersendiri. Ekstensi ini menghapus berbagai fungsi, tipe data, dan artefak lain yang diperlukan untuk pemrosesan gambar raster dari ekstensi `postgis` inti. Artinya, jika kasus penggunaan Anda tidak memerlukan pemrosesan raster, Anda tidak perlu menginstal ekstensi `postgis_raster`.

Dalam contoh peningkatan berikut, perintah peningkatan pertama mengekstrak fungsionalitas raster ke dalam ekstensi `postgis_raster`. Selanjutnya, perintah peningkatan kedua diperlukan untuk meningkatkan `postgis_raster` ke versi baru.

Untuk meng-upgrade dari Post GIS 2 ke Post GIS 3

1. Identifikasi versi default Post GIS yang tersedia untuk SQL versi Postgre di cluster Aurora Postgre DB Anda. SQL Untuk melakukannya, jalankan kueri berikut.

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                          comment
---------+-----------------+-------------------
+------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. Identifikasi versi Post yang GIS diinstal di setiap database pada instance penulis cluster Aurora SQL Postgre DB Anda. Dengan kata lain, buat kueri setiap basis data pengguna sebagai berikut.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
```

```
    1;
  Name   | Version | Schema |                                Description
---------+---------+--------
+-------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

Ketidakcocokan antara versi default (Post GIS 3.1.4) dan versi yang diinstal (Post GIS 2.3.7) berarti Anda perlu memutakhirkan ekstensi Post. GIS

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. Jalankan kueri berikut untuk memverifikasi bahwa fungsi raster kini telah berada dalam paketnya tersendiri.

```
SELECT
    probin,
    count(*)
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

Output-nya menunjukkan bahwa masih ada perbedaan antarversi. GISFungsi Post adalah versi 3 (postgis-3), sedangkan fungsi raster (rtpostgis) adalah versi 2 (rtpostgis-2.3). Untuk menyelesaikan peningkatan, Anda dapat menjalankan perintah peningkatan lagi sebagai berikut.

```
postgres=> SELECT postgis_extensions_upgrade();
```

Anda dapat dengan aman mengabaikan pesan peringatan. Jalankan lagi kueri berikut untuk memverifikasi bahwa peningkatan telah selesai. Upgrade selesai ketika Post GIS dan semua ekstensi terkait tidak ditandai sebagai perlu upgrade.

```
SELECT postgis_full_version();
```

4. Gunakan kueri berikut untuk melihat proses peningkatan yang telah selesai dan ekstensi yang dikemas secara terpisah, lalu verifikasi bahwa versinya telah sesuai.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                            Description
----------------+---------+--------
+------------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

Output menunjukkan bahwa ekstensi Post GIS 2 ditingkatkan ke Post GIS 3, dan keduanya `postgis` dan `postgis_raster` ekstensi yang sekarang terpisah adalah versi 3.1.5.

Setelah peningkatan ini selesai, Anda dapat menghapus ekstensi seperti berikut jika tidak berencana menggunakan fungsionalitas raster.

```
DROP EXTENSION postgis_raster;
```

# Mengelola partisi PostgreSQL dengan ekstensi pg_partman

Partisi tabel PostgreSQL menyediakan kerangka kerja untuk penanganan input data dan laporan performa tinggi. Gunakan partisi untuk basis data yang memerlukan input data dalam jumlah besar dengan cepat. Partisi juga menyediakan kueri tabel besar yang lebih cepat. Partisi membantu memelihara data tanpa memengaruhi instans basis data karena sumber daya I/O yang diperlukan lebih sedikit.

Dengan partisi, Anda dapat membagi data menjadi beberapa bagian berukuran kustom untuk diproses. Misalnya, Anda dapat membagi data deret waktu ke dalam berbagai rentang seperti per jam, harian, mingguan, bulanan, triwulanan, tahunan, kustom, atau kombinasinya. Untuk contoh data deret waktu, jika Anda membagi tabel berdasarkan jam, setiap partisi akan berisi data per satu jam. Jika Anda membagi tabel deret waktu berdasarkan hari, partisi akan berisi data per hari, dan seterusnya. Kunci partisi mengontrol ukuran partisi.

Saat Anda menggunakan perintah SQL `INSERT` atau `UPDATE` pada tabel yang dipartisi, mesin basis data merutekan data ke partisi yang sesuai. Partisi tabel PostgreSQL yang menyimpan data adalah tabel turunan dari tabel utama.

Selama pembacaan kueri basis data, pengoptimal PostgreSQL memeriksa klausul `WHERE` pada kueri dan, jika memungkinkan, mengarahkan pemindaian basis data hanya untuk partisi yang relevan.

Mulai versi 10, PostgreSQL menggunakan partisi deklaratif untuk mengimplementasikan partisi tabel. Ini juga dikenal sebagai partisi PostgreSQL asli. Sebelum PostgreSQL versi 10, Anda menggunakan pemicu untuk mengimplementasikan partisi.

Partisi tabel PostgreSQL menyediakan fitur berikut:

- Pembuatan partisi baru setiap saat.
- Rentang partisi bervariasi.
- Partisi yang dapat dilepas dan dapat dipasang kembali menggunakan pernyataan bahasa definisi data (DDL).

  Sebagai contoh, partisi yang dapat dilepas berguna untuk menghapus data historis dari partisi utama, tetapi menyimpan data historis untuk analisis.
- Partisi baru mewarisi properti tabel basis data induk, termasuk yang berikut ini:
  - Indeks
  - Kunci primer, yang harus berisi kolom kunci partisi

  - Kunci asing
  - Batasan pemeriksaan
  - Referensi
- Membuat indeks untuk seluruh tabel atau partisi tertentu.

Anda tidak dapat mengubah skema partisi individual. Namun, Anda dapat mengubah tabel induk (seperti menambahkan kolom baru), yang disebarkan ke partisi.

Topik

- Ikhtisar ekstensi pg_partman PostgreSQL
- Mengaktifkan ekstensi pg_partman
- Mengonfigurasi partisi menggunakan fungsi create_parent
- Mengonfigurasi pemeliharaan partisi menggunakan fungsi run_maintenance_proc

## Ikhtisar ekstensi pg_partman PostgreSQL

Anda dapat menggunakan ekstensi `pg_partman` PostgreSQL untuk mengotomatiskan pembuatan dan pemeliharaan partisi tabel. Untuk informasi umum selengkapnya, lihat PG Partition Manager dalam dokumentasi `pg_partman`.

> **Note**
>
> Ekstensi `pg_partman` didukung di Aurora for PostgreSQL versi 12.6 dan yang lebih tinggi.

Alih-alih membuat setiap partisi secara manual, Anda dapat mengonfigurasi `pg_partman` dengan pengaturan berikut:

- Tabel yang akan dipartisi
- Jenis partisi
- Kunci partisi
- Granularitas partisi
- Opsi pra-pembuatan dan manajemen partisi

Setelah membuat tabel yang dipartisi PostgreSQL, daftarkan dengan pg_partman dengan memanggil fungsi create_parent. Tindakan ini akan membuat partisi yang diperlukan berdasarkan parameter yang Anda teruskan ke fungsi.

Ekstensi pg_partman juga menyediakan fungsi run_maintenance_proc, yang dapat Anda panggil sesuai jadwal untuk secara otomatis mengelola partisi. Untuk memastikan bahwa partisi yang tepat dibuat sesuai kebutuhan, jadwalkan fungsi ini untuk berjalan secara berkala (seperti per jam). Anda juga dapat memastikan bahwa partisi secara otomatis dibatalkan.

## Mengaktifkan ekstensi pg_partman

Jika Anda memiliki beberapa basis data di dalam instans DB PostgreSQL yang partisinya ingin Anda kelola, aktifkan ekstensi pg_partman secara terpisah untuk setiap basis data. Untuk mengaktifkan ekstensi pg_partman untuk basis data tertentu, buat skema pemeliharaan partisi, kemudian buat ekstensi pg_partman seperti berikut.

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> Untuk membuat ekstensi pg_partman, pastikan Anda memiliki hak istimewa rds_superuser.

Jika Anda menerima kesalahan seperti berikut, berikan hak istimewa rds_superuser untuk akun tersebut atau gunakan akun pengguna super Anda.

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

Untuk memberikan hak istimewa rds_superuser, hubungkan dengan akun pengguna super Anda dan jalankan perintah berikut.

```
GRANT rds_superuser TO user-or-role;
```

Untuk contoh yang menunjukkan penggunaan ekstensi pg_partman, kita gunakan contoh tabel dan partisi basis data berikut. Basis data ini menggunakan tabel yang dipartisi berdasarkan stempel

waktu. Skema `data_mart` berisi tabel bernama `events` dengan kolom bernama `created_at`. Pengaturan berikut disertakan dalam tabel `events`:

- Kunci primer `event_id` dan `created_at`, yang harus memiliki kolom yang digunakan untuk memandu partisi.
- Batasan pemeriksaan `ck_valid_operation` guna menerapkan nilai untuk kolom tabel `operation`.
- Dua kunci asing, yang salah satunya (`fk_orga_membership`) menunjuk ke tabel eksternal `organization` dan kunci lainnya (`fk_parent_event_id`) adalah kunci asing referensi mandiri.
- Dua indeks, yang salah satunya (`idx_org_id`) untuk kunci asing dan indeks lainnya (`idx_event_type`) untuk jenis peristiwa.

Pernyataan DDL berikut membuat objek ini, yang secara otomatis disertakan pada setiap partisi.

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

### Mengonfigurasi partisi menggunakan fungsi create_parent

Setelah mengaktifkan ekstensi `pg_partman`, gunakan fungsi `create_parent` untuk mengonfigurasi partisi di dalam skema pemeliharaan partisi. Contoh berikut menggunakan contoh tabel `events` yang dibuat di Mengaktifkan ekstensi pg_partman. Panggil fungsi `create_parent` seperti berikut.

```
SELECT partman.create_parent(
 p_parent_table => 'data_mart.events',
 p_control      => 'created_at',
 p_type         => 'range',
 p_interval     => '1 day',
 p_premake      => 30);
```

Parameternya adalah sebagai berikut:

- `p_parent_table` – Tabel induk yang dipartisi. Tabel ini harus sudah ada dan sepenuhnya memenuhi syarat, termasuk skemanya.
- `p_control` – Kolom yang menjadi dasar pembuatan partisi. Jenis data harus bilangan bulat atau berbasis waktu.
- `p_type` – Jenisnya adalah `'range'` atau `'list'`.
- `p_interval` – Interval waktu atau rentang bilangan bulat untuk setiap partisi. Contoh nilai meliputi `1 day1 hour`,, dan sebagainya.
- `p_premake` – Jumlah partisi yang akan dibuat terlebih dahulu untuk mendukung sisipan baru.

Untuk keterangan lengkap tentang fungsi `create_parent`, lihat Creation Functions dalam dokumentasi `pg_partman`.

### Mengonfigurasi pemeliharaan partisi menggunakan fungsi run_maintenance_proc

Anda dapat menjalankan operasi pemeliharaan partisi untuk secara otomatis membuat partisi baru, melepaskan partisi, atau menghapus partisi lama. Pemeliharaan partisi bergantung pada fungsi `run_maintenance_proc` pada ekstensi `pg_partman` dan `pg_cron`, yang memulai penjadwal internal. Penjadwal `pg_cron` secara otomatis mengeksekusi pernyataan, fungsi, dan prosedur SQL yang ditetapkan dalam basis data Anda.

Contoh berikut menggunakan contoh tabel `events` yang dibuat di Mengaktifkan ekstensi pg_partman untuk mengatur operasi pemeliharaan partisi agar berjalan secara otomatis. Sebagai prasyarat,

tambahkan pg_cron ke parameter shared_preload_libraries dalam grup parameter instans DB.

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

Berikut ini, Anda dapat menemukan step-by-step penjelasan dari contoh sebelumnya:

1. Modifikasi grup parameter yang terkait dengan instans DB Anda dan tambahkan pg_cron ke nilai parameter shared_preload_libraries. Untuk menerapkan perubahan, instans DB harus dimulai ulang. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB di .
2. Jalankan perintah CREATE EXTENSION pg_cron; menggunakan akun yang memiliki izin rds_superuser. Tindakan ini akan mengaktifkan ekstensi pg_cron. Untuk informasi selengkapnya, lihat Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL.
3. Jalankan perintah UPDATE partman.part_config guna menyesuaikan pengaturan pg_partman untuk tabel data_mart.events.
4. Jalankan perintah SET . . . untuk mengonfigurasi tabel data_mart.events dengan klausul berikut:
   a. infinite_time_partitions = true, – Mengonfigurasi tabel untuk dapat secara otomatis membuat partisi baru tanpa batas.
   b. retention = '3 months', – Mengonfigurasi tabel agar memiliki retensi maksimum tiga bulan.
   c. retention_keep_table=true – Mengonfigurasi tabel agar ketika periode retensi sudah habis, tabel tidak akan dihapus secara otomatis. Sebaliknya, partisi yang lebih lama dari periode retensi hanya dilepaskan dari tabel induk.
5. Jalankan perintah SELECT cron.schedule . . . untuk membuat panggilan fungsi pg_cron. Panggilan ini menetapkan frekuensi penjadwal menjalankan prosedur pemeliharaan pg_partman, partman.run_maintenance_proc. Untuk contoh ini, prosedur berjalan setiap jam.

Untuk keterangan lengkap tentang fungsi `run_maintenance_proc`, lihat Maintenance Functions dalam dokumentasi `pg_partman`.

## Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL

Anda dapat menggunakan ekstensi `pg_cron` PostgreSQL untuk menjadwalkan perintah pemeliharaan dalam basis data PostgreSQL. Untuk informasi selengkapnya tentang ekstensi ini, lihat What is pg_cron? dalam dokumentasi pg_cron.

Ekstensi `pg_cron` didukung pada mesin Aurora PostgreSQL versi 12.6 dan yang lebih tinggi

Untuk mempelajari selengkapnya tentang penggunaan `pg_cron`, lihat Menjadwalkan pekerjaan dengan pg_cron di RDS for PostgreSQL atau basis data Aurora Edisi Kompatibel PostgreSQL.

Topik

- Menyiapkan ekstensi pg_cron
- Mengizinkan pengguna basis data untuk menggunakan pg_cron
- Menjadwalkan pekerjaan pg_cron
- Referensi untuk ekstensi pg_cron

### Menyiapkan ekstensi pg_cron

Siapkan ekstensi `pg_cron` sebagai berikut:

1. Modifikasi grup parameter kustom yang terkait dengan instans DB PostgreSQL Anda dengan menambahkan `pg_cron` ke nilai parameter `shared_preload_libraries`.

   Mulai ulang instans DB PostgreSQL untuk menerapkan perubahan pada grup parameter. Untuk mempelajari selengkapnya tentang penggunaan grup parameter, lihat Parameter Amazon Aurora PostgreSQL.
2. Setelah instans DB PostgreSQL dimulai ulang, jalankan perintah berikut menggunakan akun yang memiliki izin `rds_superuser`. Misalnya, jika Anda menggunakan pengaturan default saat membuat klaster DB Aurora PostgreSQL, hubungkan sebagai pengguna `postgres` dan buat ekstensi.

   ```
   CREATE EXTENSION pg_cron;
   ```

Penjadwal pg_cron diatur dalam basis data PostgreSQL default bernama postgres. Objek pg_cron dibuat dalam basis data postgres ini dan semua tindakan penjadwalan berjalan dalam basis data ini.

3. Anda dapat menggunakan pengaturan default, atau Anda dapat menjadwalkan pekerjaan untuk berjalan di basis data lain dalam instans DB PostgreSQL Anda. Untuk menjadwalkan pekerjaan basis data lain dalam instans DB PostgreSQL Anda, lihat contoh di Menjadwalkan pekerjaan cron untuk basis data selain basis data default.

## Mengizinkan pengguna basis data untuk menggunakan pg_cron

Menginstal ekstensi pg_cron membutuhkan hak istimewa rds_superuser. Namun, izin untuk menggunakan pg_cron dapat diberikan (oleh anggota grup/peran rds_superuser) kepada pengguna basis data lain, sehingga mereka dapat menjadwalkan pekerjaannya sendiri. Sebaiknya hanya berikan izin yang diperlukan ke skema cron sesuai kebutuhan ketika skema ini dapat meningkatkan operasi di lingkungan produksi Anda.

Untuk memberikan izin pengguna basis data dalam skema cron, jalankan perintah berikut:

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

Ini memberikan *db-user* izin untuk mengakses cron skema untuk menjadwalkan pekerjaan cron untuk objek yang mereka memiliki izin untuk mengakses. Jika pengguna basis data tidak memiliki izin, tugas akan gagal setelah memposting pesan kesalahan ke file postgresql.log, seperti yang ditunjukkan berikut:

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

Dengan kata lain, pastikan bahwa pengguna database yang diberikan izin pada cron skema juga memiliki izin pada objek (tabel, skema, dan sebagainya) yang mereka rencanakan untuk dijadwalkan.

Detail pekerjaan cron dan keberhasilan atau kegagalannya juga ditangkap dalam cron.job_run_details tabel. Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .

## Menjadwalkan pekerjaan pg_cron

Bagian berikut menunjukkan bagaimana Anda dapat menjadwalkan berbagai tugas manajemen menggunakan pekerjaan pg_cron.

> **Note**
>
> Saat membuat pekerjaan pg_cron, periksa apakah pengaturan jumlah max_worker_processes lebih besar dari cron.max_running_jobs. Pekerjaan pg_cron akan gagal jika kehabisan proses pekerja latar belakang. Jumlah default pekerjaan pg_cron adalah 5. Untuk informasi selengkapnya, lihat Parameter untuk mengelola ekstensi pg_cron.

Topik

- Mengosongkan tabel
- Membersihkan tabel riwayat pg_cron
- Mencatat log kesalahan ke file postgresql.log saja
- Menjadwalkan pekerjaan cron untuk basis data selain basis data default

### Mengosongkan tabel

Pengosongan otomatis menangani pemeliharaan vakum untuk kebanyakan kasus. Namun, Anda mungkin ingin menjadwalkan pengosongan tabel pada waktu yang Anda pilih.

Berikut adalah contoh penggunaan fungsi cron.schedule guna menyiapkan pekerjaan untuk menggunakan VACUUM FREEZE pada tabel tertentu setiap hari pada pukul 22.00 (GMT).

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

Setelah contoh sebelumnya berjalan, Anda dapat memeriksa riwayatnya di tabel cron.job_run_details seperti berikut.

```
postgres=> SELECT * FROM cron.job_run_details;
```

```
jobid  | runid | job_pid | database | username | command                          |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+-------------------------------
+-----------+---------------+------------------------------
+-------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

Berikut ini adalah query dari `cron.job_run_details` tabel untuk melihat pekerjaan gagal.

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-----------------------------------------------+-------------------------------
+-------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
(1 row)
```

Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .

Membersihkan tabel riwayat pg_cron

Tabel `cron.job_run_details` berisi riwayat pekerjaan cron yang ukurannya bisa menjadi sangat besar dari waktu ke waktu. Sebaiknya jadwalkan pekerjaan untuk membersihkan tabel ini. Sebagai contoh, menyimpan entri selama seminggu mungkin sudah cukup untuk tujuan pemecahan masalah.

Contoh berikut ini menggunakan fungsi cron.schedule untuk menjadwalkan pekerjaan yang berjalan setiap hari pada tengah malam untuk membersihkan tabel `cron.job_run_details`. Pekerjaan ini hanya menyimpan tujuh hari terakhir. Gunakan akun `rds_superuser` untuk menjadwalkan pekerjaan seperti berikut.

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .

Mencatat log kesalahan ke file postgresql.log saja

Untuk mencegah penulisan ke tabel `cron.job_run_details`, modifikasi grup parameter yang terkait dengan instans DB PostgreSQL dan nonaktifkan parameter `cron.log_run`. Ekstensi `pg_cron` tidak lagi menulis ke tabel dan menangkap kesalahan ke file `postgresql.log` saja. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB di .

Gunakan perintah berikut untuk memeriksa nilai parameter `cron.log_run`.

```
postgres=> SHOW cron.log_run;
```

Untuk informasi selengkapnya, lihat Parameter untuk mengelola ekstensi pg_cron.

Menjadwalkan pekerjaan cron untuk basis data selain basis data default

Semua metadata untuk `pg_cron` disimpan dalam basis data default PostgreSQL bernama `postgres`. Karena pekerja latar belakang digunakan untuk menjalankan pekerjaan pemeliharaan cron, Anda dapat menjadwalkan pekerjaan di salah satu basis data Anda dalam instans DB PostgreSQL:

> **Note**
>
> Hanya pengguna dengan `rds_superuser` peran atau `rds_superuser` hak istimewa yang dapat mencantumkan semua pekerjaan cron dalam database. Pengguna lain hanya dapat melihat pekerjaan mereka sendiri di `cron.job` tabel.

1. Dalam basis data cron, jadwalkan pekerjaan seperti yang biasa Anda lakukan menggunakan cron.schedule.

   ```
   postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
    freeze test_table');
   ```

2. Sebagai pengguna dengan peran `rds_superuser`, perbarui kolom basis data untuk pekerjaan yang baru saja Anda buat agar berjalan di basis data lain dalam instans DB PostgreSQL Anda.

   ```
   postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
   ```

3. Verifikasi dengan membuat kueri tabel `cron.job`.

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                          | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+----------------------------------+-----------+----------
+----------+----------+--------+-----------------------
106   | 29 03 * * * | vacuum freeze test_table       | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1    | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> Dalam beberapa situasi, Anda mungkin menambahkan pekerjaan cron yang ingin Anda jalankan di basis data yang berbeda. Dalam kasus tersebut, pekerjaan mungkin mencoba untuk dijalankan dalam basis data default (`postgres`) sebelum kolom basis data ditentukan dengan benar. Jika nama pengguna memiliki izin, berarti pekerjaan berhasil dijalankan di basis data default.

## Referensi untuk ekstensi pg_cron

Anda dapat menggunakan parameter, fungsi, dan tabel berikut dengan ekstensi `pg_cron`. Untuk informasi selengkapnya, lihat What is pg_cron? dalam dokumentasi pg_cron.

Topik

- Parameter untuk mengelola ekstensi pg_cron
- Referensi fungsi: cron.schedule
- Referensi fungsi: cron.unschedule
- Tabel untuk menjadwalkan pekerjaan dan menangkap status

### Parameter untuk mengelola ekstensi pg_cron

Berikut adalah daftar parameter yang mengontrol perilaku ekstensi `pg_cron`.

| Parameter | Deskripsi |
| --- | --- |
| cron.database_name | Basis data penyimpanan metadata pg_cron. |
| cron.host | Nama host untuk menghubungkan ke PostgreSQL. Anda tidak dapat mengubah nilai ini. |
| cron.log_run | Catat log setiap pekerjaan yang berjalan di tabel job_run_details . Nilainya adalah on atau off. Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status . |
| cron.log_statement | Catat log semua pernyataan cron sebelum menjalankannya. Nilainya adalah on atau off. |
| cron.max_running_jobs | Jumlah maksimum pekerjaan yang dapat dijalankan secara bersamaan. |
| cron.use_background_workers | Gunakan pekerja latar belakang, bukan sesi klien. Anda tidak dapat mengubah nilai ini. |

Gunakan perintah SQL berikut untuk menampilkan parameter ini dan nilainya.

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

Referensi fungsi: cron.schedule

Fungsi ini menjadwalkan pekerjaan cron. Pada mulanya, pekerjaan dijadwalkan di basis data postgres default. Fungsi tersebut mengembalikan nilai bigint yang mewakili pengidentifikasi pekerjaan. Untuk menjadwalkan pekerjaan agar berjalan di basis data lain dalam instans DB PostgreSQL Anda, lihat contoh di Menjadwalkan pekerjaan cron untuk basis data selain basis data default.

Fungsi ini memiliki dua format sintaks.

### Sintaks

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

### Parameter

| Parameter | Deskripsi |
| --- | --- |
| `job_name` | Nama pekerjaan cron. |
| `schedule` | Teks yang menunjukkan jadwal pekerjaan cron. Formatnya adalah format cron standar. |
| `command` | Teks perintah yang akan dijalankan. |

### Contoh

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

Referensi fungsi: cron.unschedule

Fungsi ini menghapus pekerjaan cron. Anda dapat menentukan `job_name` atau `job_id`. Suatu kebijakan memastikan bahwa Anda adalah pemilik guna menghapus jadwal pekerjaan. Fungsi akan mengembalikan Boolean yang menunjukkan keberhasilan atau kegagalan.

Fungsi memiliki format sintaks berikut.

Sintaks

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

Parameter

| Parameter | Deskripsi |
| --- | --- |
| `job_id` | Pengidentifikasi pekerjaan yang dikembalikan dari fungsi `cron.schedule` saat pekerjaan cron dijadwalkan. |
| `job_name` | Nama pekerjaan cron yang dijadwalkan dengan fungsi `cron.schedule` . |

Contoh

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

Tabel untuk menjadwalkan pekerjaan dan menangkap status

Tabel berikut digunakan untuk menjadwalkan pekerjaan cron dan mencatat bagaimana pekerjaan tersebut diselesaikan.

| Tabel | Deskripsi |
| --- | --- |
| `cron.job` | Berisi metadata tentang setiap pekerjaan yang dijadwalkan. Sebagian besar interaksi dengan tabel ini harus dilakukan menggunakan fungsi `cron.schedule` dan `cron.unschedule` .<br><br>**Important**<br>Sebaiknya jangan memberikan pembaruan atau memasukkan hak istimewa secara langsung ke tabel ini. Hal tersebut akan memungkinkan pengguna untuk memperbarui kolom `username` agar berjalan sebagai `rds-superuser` . |
| `cron.job_run_details` | Berisi informasi historis tentang pekerjaan terjadwal yang telah dijalankan sebelumnya. Hal ini berguna untuk menyelidiki status, pesan kembali, dan waktu mulai dan akhir dari pekerjaan yang berjalan.<br><br>**Note**<br>Agar ukuran tabel ini tidak bertambah besar, bersihkan secara rutin. Sebagai contoh, lihat Membersihkan tabel riwayat pg_cron. |

## Menggunakan pgAudit untuk mencatat aktivitas database

Lembaga keuangan, lembaga pemerintah, dan banyak industri perlu menyimpan log audit untuk memenuhi persyaratan regulasi. Dengan menggunakan ekstensi Postgre SQL Audit (pgAudit) dengan klaster SQL DB Aurora Postgre Anda , Anda dapat menangkap catatan terperinci yang

biasanya dibutuhkan oleh auditor atau untuk memenuhi persyaratan peraturan. Misalnya, Anda dapat mengatur pgAudit ekstensi untuk melacak perubahan yang dibuat pada database dan tabel tertentu, untuk merekam pengguna yang membuat perubahan, dan banyak detail lainnya.

pgAudit Ekstensi dibangun di atas fungsionalitas infrastruktur SQL logging Postgre asli dengan memperluas pesan log dengan lebih detail. Dengan kata lain, Anda menggunakan pendekatan yang sama untuk melihat log audit seperti untuk melihat pesan log lain. Untuk informasi lebih lanjut tentang SQL pencatatan Postgre, lihat. File log basis data Aurora PostgreSQL

pgAudit Ekstensi menyunting data sensitif seperti kata sandi cleartext dari log. Jika cluster Aurora Postgre SQL DB Anda dikonfigurasi untuk mencatat pernyataan bahasa manipulasi data (DML) seperti yang dijelaskan dalamMengaktifkan pencatatan log kueri untuk klaster DB Aurora PostgreSQL, Anda dapat menghindari masalah kata sandi cleartext dengan menggunakan ekstensi Postgre Audit. SQL

Anda dapat mengonfigurasi audit pada basis data instans dengan tingkat kekhususan yang tinggi. Anda dapat mengaudit semua basis data dan semua pengguna. Atau, Anda dapat memilih untuk mengaudit hanya basis data, pengguna, dan objek tertentu lainnya. Anda juga dapat secara eksplisit mengecualikan pengguna dan basis data tertentu agar tidak diaudit. Untuk informasi selengkapnya, lihat Mengecualikan pengguna atau basis data dari pencatatan log audit.

Mengingat jumlah detail yang dapat ditangkap, kami menyarankan jika Anda menggunakannyapgAudit, Anda memantau konsumsi penyimpanan Anda.

pgAudit Ekstensi ini didukung pada semua versi Aurora SQL Postgre yang tersedia. Untuk daftar versi yang didukung oleh pgAudit versi Aurora Postgre, lihat SQL Versi ekstensi untuk Amazon Aurora Postgre SQL di Catatan Rilis untuk Aurora Postgre. SQL

Topik

- Menyiapkan pgAudit ekstensi
- Mengaudit objek basis data
- Mengecualikan pengguna atau basis data dari pencatatan log audit
- Referensi untuk pgAudit ekstensi

## Menyiapkan pgAudit ekstensi

Untuk menyiapkan pgAudit ekstensi pada cluster DB instans Postgre SQL DB, Anda terlebih dahulu menambahkan pgAudit ke pustaka bersama pada Anda. RDS RDS SQL grup parameter cluster DB

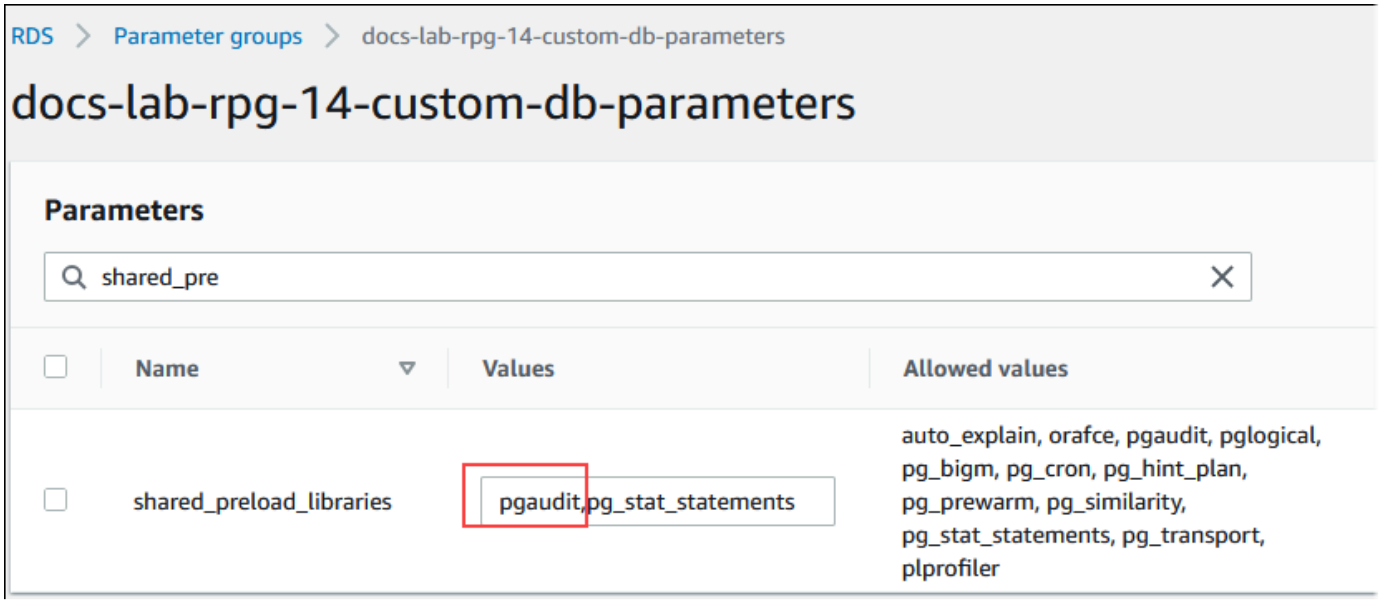
kustom untuk cluster Aurora SQL Postgre DB Anda. Untuk informasi cara membuat Grup parameter klaster DB, lihat .  Selanjutnya, Anda menginstal pgAudit ekstensi. Terakhir, Anda menentukan basis data dan objek yang ingin diaudit. Prosedur di bagian ini menunjukkan caranya kepada Anda. Anda dapat menggunakan AWS Management Console atau AWS CLI.

Anda harus memiliki izin sebagai peran `rds_superuser` untuk melakukan semua tugas ini.

Konsol

Untuk mengatur pgAudit ekstensi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih instance Writer cluster Aurora SQL Postgre DB Anda untuk Postgre DB. SQL
3. Buka tab Konfigurasi untuk instance penulis cluster Aurora Postgre SQL DB Anda. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter khusus yang terkait dengan cluster Aurora SQL Postgre DB Anda.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pgaudit` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.




8. Reboot instance penulis cluster Aurora Postgre SQL DB Anda RDS sehingga perubahan Anda pada parameter berlaku. `shared_preload_libraries`
9. Ketika instance tersedia, verifikasi yang pgAudit telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda, dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. Dengan pgAudit diinisialisasi, Anda sekarang dapat membuat ekstensi. Anda perlu membuat ekstensi setelah menginisialisasi pustaka karena `pgaudit` ekstensi menginstal pemicu peristiwa untuk mengaudit pernyataan bahasa definisi data (). DDL

```
CREATE EXTENSION pgaudit;
```

11. Tutup sesi `psql`.

```
labdb=> \q
```

12. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

13. Temukan parameter `pgaudit.log` dalam daftar lalu atur ke nilai yang sesuai untuk kasus penggunaan Anda. Misalnya, menyetel parameter `pgaudit.log` ke `write` seperti yang ditunjukkan pada gambar berikut menangkap sisipan, pembaruan, penghapusan, dan beberapa jenis lainnya yang berubah pada log.

Anda juga dapat memilih salah satu nilai berikut untuk parameter `pgaudit.log`.

- none – Ini adalah default. Tidak ada perubahan basis data yang dibuat log.
- semua – Semua dibuat log (baca, tulis, fungsi, peran, ddl, misc).
- ddl — Mencatat semua pernyataan bahasa definisi data (DDL) yang tidak termasuk dalam `ROLE` kelas.
- fungsi – Membuat log panggilan fungsi dan blok `DO`.
- misc – Membuat log berbagai perintah, seperti `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, dan `SET`.
- baca – Membuat log `SELECT` dan `COPY` saat sumbernya adalah relasi (seperti tabel) atau kueri.
- peran – Membuat log pernyataan yang terkait dengan peran dan hak akses, seperti `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, dan `DROP ROLE`.
- write – Membuat log `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, dan `COPY` saat tujuannya adalah relasi (tabel).

14. Pilih Simpan perubahan.
15. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

16. Pilih instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance dari daftar Database.

AWS CLI

Untuk pengaturan pgAudit

Untuk pengaturan pgAudit menggunakan AWS CLI, Anda memanggil modify-db-parameter-groupoperasi untuk memodifikasi parameter log audit di grup parameter kustom Anda, seperti yang ditunjukkan dalam prosedur berikut.

1. Gunakan yang berikut ini AWS CLI perintah untuk `pgaudit` menambah `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan yang berikut ini AWS CLI perintah untuk me-reboot instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance Postgre DB diinisialisasi.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Saat instans tersedia, verifikasikan bahwa `pgaudit` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda,  dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

Dengan pgAudit diinisialisasi, Anda sekarang dapat membuat ekstensi.

```
CREATE EXTENSION pgaudit;
```

4. Tutup `psql` sesi sehingga Anda dapat menggunakan AWS CLI.

```
labdb=> \q
```

5. Gunakan yang berikut ini AWS CLI perintah untuk menentukan kelas pernyataan yang ingin dicatat oleh sesi audit logging. Contoh ini menetapkan parameter `pgaudit.log` ke `write`, yang menangkap sisipan, pembaruan, dan penghapusan ke log.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

   Anda juga dapat memilih salah satu nilai berikut untuk parameter `pgaudit.log`.

   - none – Ini adalah default. Tidak ada perubahan basis data yang dibuat log.
   - semua – Semua dibuat log (baca, tulis, fungsi, peran, ddl, misc).
   - ddl — Mencatat semua pernyataan bahasa definisi data (DDL) yang tidak termasuk dalam `ROLE` kelas.
   - fungsi – Membuat log panggilan fungsi dan blok `DO`.
   - misc – Membuat log berbagai perintah, seperti `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, dan `SET`.
   - baca – Membuat log `SELECT` dan `COPY` saat sumbernya adalah relasi (seperti tabel) atau kueri.
   - peran – Membuat log pernyataan yang terkait dengan peran dan hak akses, seperti `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, dan `DROP ROLE`.
   - write – Membuat log `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, dan `COPY` saat tujuannya adalah relasi (tabel).

   Nyalakan ulang instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance Postgre DB ini AWS CLI perintah.

```
aws rds reboot-db-instance \
```

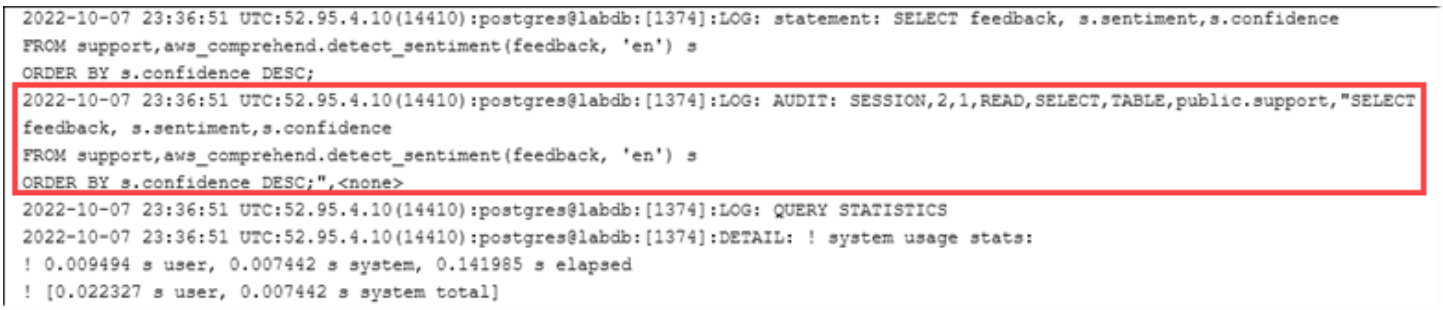
```
    --db-instance-identifier writer-instance \
    --region aws-region
```

## Mengaudit objek basis data

Dengan pgAudit pengaturan pada cluster Aurora Postgre SQL DB Anda dan dikonfigurasi untuk kebutuhan Anda, informasi yang lebih rinci ditangkap dalam log Postgre. SQL Misalnya, sementara konfigurasi SQL logging Postgre default mengidentifikasi tanggal dan waktu perubahan dibuat dalam tabel database, dengan pgAudit ekstensi entri log dapat mencakup skema, pengguna yang membuat perubahan, dan detail lainnya tergantung pada bagaimana parameter ekstensi dikonfigurasi. Anda dapat menyiapkan audit untuk melacak perubahan dengan cara berikut.

- Untuk setiap sesi, oleh pengguna. Untuk tingkat sesi, Anda dapat menangkap teks perintah yang sepenuhnya memenuhi syarat.
- Untuk setiap objek, oleh pengguna dan basis data.

Kemampuan audit objek diaktifkan saat Anda membuat peran `rds_pgaudit` pada sistem, lalu menambahkan peran ini ke parameter `pgaudit.role` dalam grup parameter kustom Anda. Secara default, parameter `pgaudit.role` tidak ditetapkan dan satu-satunya nilai yang diizinkan adalah `rds_pgaudit`. Langkah-langkah berikut mengasumsikan bahwa `pgaudit` telah diinisialisasi dan bahwa Anda telah membuat ekstensi `pgaudit` dengan mengikuti prosedur di Menyiapkan pgAudit ekstensi.



```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

Seperti yang ditunjukkan dalam contoh ini, baris "LOGAUDIT::SESSION" memberikan informasi tentang tabel dan skemanya, di antara detail lainnya.

### Menyiapkan audit objek

1. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

```
psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
username=postgrespostgres --password --dbname=labdb
```

2. Buat peran basis data yang dinamai `rds_pgaudit` menggunakan perintah berikut.

```
labdb=> CREATE ROLE rds_pgaudit;
CREATE ROLE
labdb=>
```

3. Tutup sesi `psql`.

```
labdb=> \q
```

Dalam beberapa langkah berikutnya, gunakan AWS CLI untuk memodifikasi parameter log audit di grup parameter kustom Anda.

4. Gunakan yang berikut AWS CLI perintah untuk mengatur `pgaudit.role` parameter ke`rds_pgaudit`. Secara default, parameter ini kosong, dan `rds_pgaudit` merupakan satu-satunya nilai yang diizinkan.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
 \
   --region aws-region
```

5. Gunakan yang berikut AWS CLI perintah untuk me-reboot instance penulis cluster Aurora Postgre SQL DB Anda RDS sehingga perubahan Anda pada parameter berlaku.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. Jalankan perintah berikut untuk mengonfirmasi bahwa `pgaudit.role` diatur ke `rds_pgaudit`.

```
SHOW pgaudit.role;
pgaudit.role
-----------------
rds_pgaudit
```

Untuk menguji pgAudit logging, Anda dapat menjalankan beberapa contoh perintah yang ingin Anda audit. Misalnya, Anda dapat menjalankan perintah berikut.

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

Log basis data harus berisi entri yang serupa dengan entri berikut.

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

Untuk informasi tentang menampilkan log, lihat Memantau log Amazon Aurora Amazon.

Untuk mempelajari lebih lanjut tentang pgAudit ekstensi, lihat pgAuditdi GitHub.

## Mengecualikan pengguna atau basis data dari pencatatan log audit

Seperti dibahas dalamFile log basis data Aurora PostgreSQL, SQL log Postgre mengkonsumsi ruang penyimpanan. Menggunakan pgAudit ekstensi menambah volume data yang dikumpulkan di log Anda ke berbagai tingkat, tergantung pada perubahan yang Anda lacak. Anda mungkin tidak perlu mengaudit setiap pengguna atau database di cluster Aurora SQL Postgre DB Anda.

Untuk meminimalkan dampak pada penyimpanan dan untuk menghindari pengambilan catatan audit yang tidak perlu, Anda dapat mengecualikan pengguna dan basis data agar tidak diaudit. Anda juga dapat mengubah pencatatan log dalam sesi tertentu. Contoh berikut menunjukkan caranya kepada Anda.

> **Note**
>
> Pengaturan parameter pada tingkat sesi lebih diutamakan daripada pengaturan dalam grup parameter cluster DB kustom untuk grup parameter DB khusus instance penulis Aurora Postgre SQL DB cluster untuk instance . RDS SQL Jika Anda tidak ingin pengguna basis data melewati pengaturan konfigurasi pencatatan audit, pastikan untuk mengubah izin mereka.

Misalkan cluster Aurora Postgre SQL DB Anda dikonfigurasi untuk mengaudit tingkat aktivitas yang sama untuk semua pengguna dan database. Anda kemudian memutuskan bahwa Anda tidak ingin mengaudit pengguna `myuser`. Anda dapat menonaktifkan audit `myuser` dengan SQL perintah berikut.

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

Kemudian, Anda dapat menggunakan kueri berikut untuk memeriksa kolom `user_specific_settings` untuk `pgaudit.log` guna mengonfirmasi bahwa parameter ditetapkan ke `NONE`.

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

Anda akan melihat output seperti berikut ini.

```
 user_name | user_specific_settings
-----------+-----------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

Anda dapat menonaktifkan pencatatan log untuk pengguna tertentu di tengah-tengah sesi mereka menggunakan basis data dengan perintah berikut.

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

Gunakan kueri berikut untuk memeriksa kolom pengaturan pgaudit.log untuk kombinasi pengguna dan basis data tertentu.

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
```

```
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

Anda akan melihat output yang mirip dengan berikut ini.

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

Setelah menonaktifkan audit `myuser`, Anda memutuskan bahwa Anda tidak ingin melacak perubahan ke `mydatabase`. Anda menonaktifkan audit untuk basis data spesifik tersebut menggunakan perintah berikut.

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

Kemudian, gunakan kueri berikut untuk memeriksa kolom database_specific_settings untuk mengonfirmasi bahwa pgaudit.log disetel ke. NONE

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

Anda akan melihat output seperti berikut ini.

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
```

```
(1 row)
```

Untuk mengembalikan pengaturan ke pengaturan default untuk myuser, gunakan perintah berikut:

```
ALTER USER myuser RESET pgaudit.log;
```

Untuk mengembalikan pengaturan ke pengaturan default untuk basis data, gunakan perintah berikut ini.

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

Untuk mengatur ulang pengguna dan basis data ke pengaturan default, gunakan perintah berikut.

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

Anda juga dapat menangkap peristiwa tertentu ke log dengan menyetel `pgaudit.log` ke salah satu nilai lain yang diizinkan untuk parameter `pgaudit.log`. Untuk informasi selengkapnya, lihat Daftar pengaturan yang diizinkan untuk parameter `pgaudit.log`.

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

## Referensi untuk pgAudit ekstensi

Anda dapat menentukan tingkat detail yang diinginkan untuk log audit Anda dengan mengubah satu atau beberapa parameter yang tercantum di bagian ini.

### Mengontrol pgAudit perilaku

Anda dapat mengontrol pencatatan log audit dengan mengubah satu atau beberapa parameter yang tercantum pada tabel berikut.

| Parameter | Deskripsi |
| --- | --- |
| `pgaudit.log` | Menentukan kelas pernyataan yang akan di-log oleh sesi audit pencatatan log. Nilai yang diizinkan termasuk ddl, fungsi, misc, |

| Parameter | Deskripsi |
| --- | --- |
| | baca, peran, tulis, tidak ada, semua. Untuk informasi selengkap nya, lihat Daftar pengaturan yang diizinkan untuk parameter `pgaudit.log` . |
| `pgaudit.log_catalog` | Saat diaktifkan (diatur ke 1), tambahkan pernyataan ke jejak audit jika semua relasi dalam pernyataan ada di pg_catalog. |
| `pgaudit.log_level` | Menentukan tingkat log untuk digunakan dalam entri log. Nilai yang diizinkan: debug5, debug4, debug3, debug2, debug1, info, notifikasi, peringatan, log |
| `pgaudit.log_parameter` | Saat diaktifkan (diatur ke 1), parameter yang diteruskan dengan pernyataan ditangkap dalam log audit. |
| `pgaudit.log_relation` | Ketika diaktifkan (diatur ke 1), log audit untuk sesi membuat entri log terpisah untuk setiap relasi (TABLE,VIEW, dan seterusnya) direferensikan dalam DML pernyataan SELECT atau. |
| `pgaudit.log_statement_once` | Menentukan apakah pencatatan log akan mencakup teks pernyataan dan parameter dengan entri log pertama untuk kombinasi pernyataan/subpernyataan atau dengan setiap entri. |
| `pgaudit.role` | Menentukan peran utama yang akan digunakan untuk pencatatan log audit objek. Satu-satunya entri yang diizinkan adalah `rds_pgaudit` . |

Daftar pengaturan yang diizinkan untuk parameter **`pgaudit.log`**

| Nilai | Deskripsi |
| --- | --- |
| Tidak ada | Ini adalah opsi default. Tidak ada perubahan basis data yang dibuat log. |
| semua | Membuat log untuk semua (baca, tulis, fungsi, peran, ddl, misc). |
| ddl | Mencatat semua pernyataan bahasa definisi data (DDL) yang tidak disertakan dalam `ROLE` kelas. |

| Nilai | Deskripsi |
| --- | --- |
| Fungsi | Membuat log panggilan fungsi dan blok `DO`. |
| misc | Membuat log berbagai perintah, seperti `DISCARD`, `FETCH`, `CHECKPOINT` , `VACUUM` dan `SET`. |
| baca | Membuat log `SELECT` dan `COPY` saat sumbernya adalah relasi (seperti tabel) atau kueri. |
| peran | Membuat log pernyataan terkait peran dan hak akses, seperti `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, dan `DROP ROLE`. |
| tulis | Membuat log `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, dan `COPY` saat tujuannya adalah relasi (tabel). |

Untuk mencatat beberapa jenis peristiwa dengan audit sesi, gunakan daftar yang dipisahkan koma. Untuk membuat log semua jenis peristiwa, atur `pgaudit.log` ke `ALL`. Boot ulang instans DB Anda untuk menerapkan perubahan.

Dengan objek audit, Anda dapat memperbaiki pencatatan log audit agar berfungsi dengan relasi tertentu. Misalnya, Anda dapat menentukan bahwa Anda ingin pencatatan log audit untuk operasi `READ` di satu tabel atau lebih.

## Menggunakan pglogical untuk menyinkronkan data di seluruh instans

Semua versi Aurora Postgre yang tersedia saat ini  mendukung ekstensi. `pglogical` Ekstensi pglogical mendahului fitur replikasi logis yang serupa secara fungsional yang diperkenalkan oleh Postgre di versi 10. SQL Untuk informasi selengkapnya, lihat Ikhtisar replikasi SQL logis Postgre dengan Aurora.

`pglogical`Ekstensi ini mendukung replikasi logis antara dua atau lebih cluster Aurora SQL Postgre DB. Ini juga mendukung replikasi antara SQL versi Postgre yang berbeda, dan antara database yang berjalan RDS untuk instans Postgre DB dan cluster Aurora Postgre SQL DB. SQL Ekstensi `pglogical` menggunakan model terbitkan-dan-langganan untuk mereplikasi perubahan pada tabel dan objek lain, seperti urutan, dari penerbit ke pelanggan. Itu bergantung pada slot replikasi untuk memastikan bahwa perubahan disinkronkan dari simpul penerbit ke simpul pelanggan, didefinisikan sebagai berikut.

- Node penerbit adalah yang merupakan sumber data yang akan direplikasi ke node lain. Simpul penerbit mendefinisikan tabel yang akan direplikasi dalam set publikasi.
- Node pelanggan adalah . WAL Pelanggan membuat langganan untuk terhubung ke penerbit dan mendapatkan data yang diterjemahkanWAL. Saat pelanggan membuat langganan, slot replikasi dibuat pada simpul penerbit.

Berikut ini, Anda dapat menemukan informasi tentang cara mengatur ekstensi `pglogical`.

Topik

- Persyaratan dan batasan untuk ekstensi pglogical
- Menyiapkan ekstensi pglogical
- Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL
- Membangun kembali replikasi logis setelah peningkatan besar
- 
- Referensi parameter untuk ekstensi pglogical

## Persyaratan dan batasan untuk ekstensi pglogical

Semua rilis Aurora Postgre SQL  yang tersedia saat ini mendukung ekstensi. `pglogical`

Baik simpul penerbit maupun simpul pelanggan harus disiapkan untuk replikasi logis.

Tabel yang ingin Anda replikasi dari penerbit ke pelanggan harus memiliki nama dan skema yang sama. Tabel ini juga harus berisi kolom yang sama, dan kolom harus menggunakan jenis data yang sama. Tabel penerbit dan pelanggan harus memiliki kunci primer yang sama. Kami menyarankan Anda hanya menggunakan PRIMARY KEY sebagai kendala unik.

Tabel pada node pelanggan dapat memiliki kendala yang lebih permisif daripada yang ada di node penerbit untuk kendala dan kendala. CHECK NOT NULL

`pglogical`Ekstensi ini menyediakan fitur seperti replikasi dua arah yang tidak didukung oleh fitur replikasi logis yang dibangun ke dalam Postgre SQL (versi 10 dan lebih tinggi). Untuk informasi lebih lanjut, lihat Replikasi dua SQLarah Postgre menggunakan pglogical.

## Menyiapkan ekstensi pglogical

Untuk menyiapkan `pglogical` ekstensi pada cluster DB instance Postgre SQL DB, Anda menambahkan `pglogical` ke pustaka bersama pada Anda. RDS RDS SQL grup parameter cluster

DB kustom untuk cluster Aurora SQL Postgre DB Anda. Anda juga perlu mengatur nilai parameter `rds.logical_replication` ke `1`, untuk mengaktifkan penguraian kode logis. Terakhir, Anda membuat ekstensi di basis data. Anda dapat menggunakan AWS Management Console atau AWS CLI untuk tugas-tugas ini.

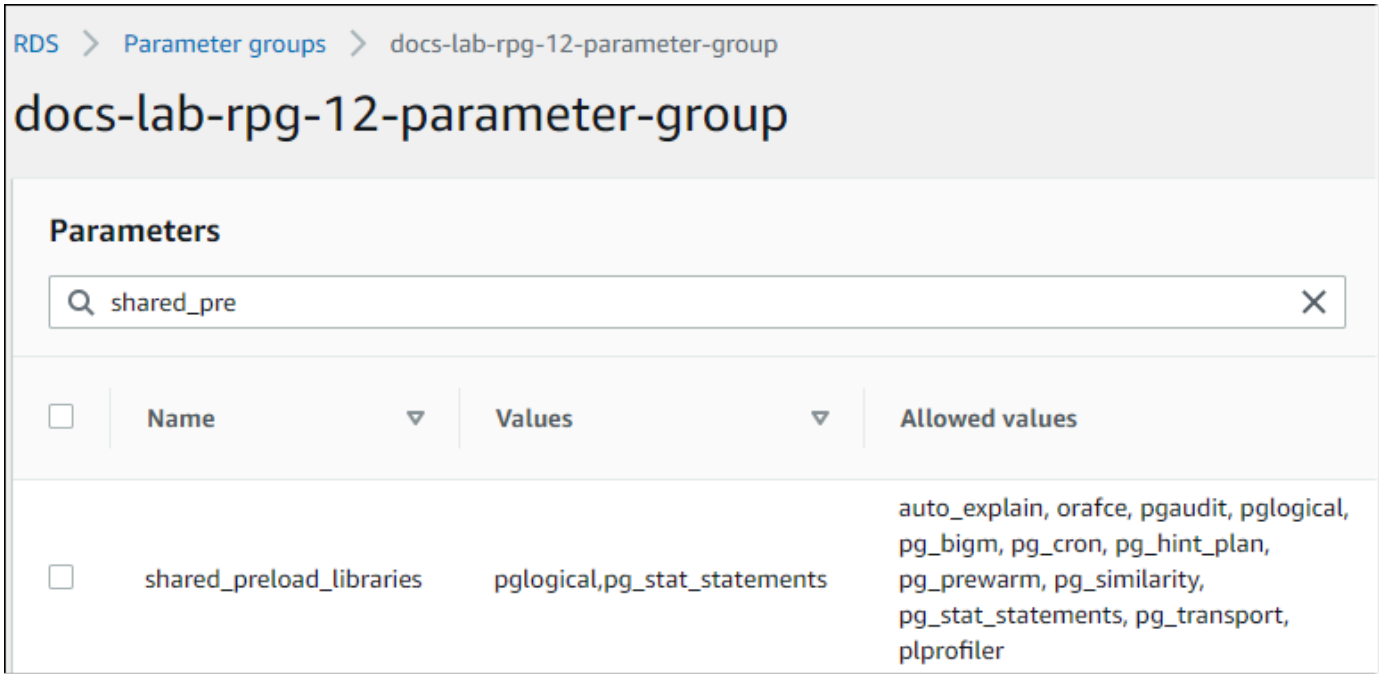
Anda harus memiliki izin sebagai peran `rds_superuser` untuk melakukan semua tugas ini.

Untuk informasi cara membuat Grup parameter klaster DB, lihat .

Konsol

Menyiapkan ekstensi pglogical

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih instance Writer cluster Aurora SQL Postgre DB Anda untuk Postgre DB. SQL
3. Buka tab Konfigurasi untuk instance penulis cluster Aurora Postgre SQL DB Anda. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter khusus yang terkait dengan cluster Aurora SQL Postgre DB Anda.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pglogical` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.




8. Temukan parameter `rds.logical_replication` dan atur ke 1, untuk mengaktifkan replikasi logis.
9. Nyalakan ulang instance penulis cluster Aurora Postgre SQL DB Anda RDS sehingga perubahan Anda berlaku.
10. Ketika instance tersedia, Anda dapat menggunakan `psql` (ataupgAdmin) untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. Untuk memverifikasi bahwa pglogical telah diinisialisasi, jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. Verifikasikan pengaturan yang memungkinkan penguraian kode logis, sebagai berikut.

```
SHOW wal_level;
wal_level
-----------
 logical
```

```
(1 row)
```

13. Buat ekstensi, sebagai berikut.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. Pilih Simpan perubahan.
15. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
16. Pilih instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance dari daftar Database untuk memilihnya, lalu pilih Reboot dari menu Tindakan.

AWS CLI

Menyiapkan ekstensi pglogical

Untuk mengatur pglogical menggunakan AWS CLI, Anda memanggil modify-db-parameter-groupoperasi untuk memodifikasi parameter tertentu dalam grup parameter kustom Anda seperti yang ditunjukkan dalam prosedur berikut.

1. Gunakan AWS CLI perintah berikut `pglogical` untuk menambah `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan AWS CLI perintah berikut untuk mengatur `rds.logical_replication` 1 untuk mengaktifkan kemampuan decoding logis untuk instance penulis cluster Aurora Postgre DB. SQL

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. Gunakan AWS CLI perintah berikut untuk me-reboot instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance Postgre DB diinisialisasi.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. Ketika instance tersedia, gunakan `psql` untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. Buat ekstensi, sebagai berikut.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. Reboot instance penulis cluster Aurora Postgre SQL DB Anda menggunakan perintah berikut. AWS CLI

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

## Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL

Prosedur berikut menunjukkan cara memulai replikasi logis di antara dua klaster DB Aurora PostgreSQL Langkah-langkah mengasumsikan bahwa sumber (penerbit) dan target (pelanggan) memiliki ekstensi `pglogical` yang disiapkan seperti yang dijelaskan dalam Menyiapkan ekstensi pglogical.

> **Note**
>
> node_nameNode pelanggan tidak dapat memulai dengan`rds`.

Untuk membuat simpul penerbit dan menentukan tabel untuk direplikasi

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL memiliki instans penulis dengan basis data yang memiliki satu atau beberapa tabel yang ingin direplikasi ke simpul lain. Anda perlu membuat ulang struktur tabel dari penerbit pada pelanggan, jadi pertama-tama, dapatkan struktur tabel jika perlu. Anda dapat melakukannya dengan menggunakan `psql` metacommand `\d` *`tablename`*, lalu membuat tabel yang sama pada instans pelanggan. Prosedur berikut membuat tabel contoh pada penerbit (sumber) untuk tujuan demonstrasi.

1. Gunakan `psql` untuk terhubung ke instans yang memiliki tabel yang ingin Anda gunakan sebagai sumber untuk pelanggan.

   ```
   psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password --dbname=labdb
   ```

   Jika tidak memiliki tabel yang ingin ditiru, Anda dapat membuat tabel contoh sebagai berikut.

   a. Buat contoh tabel menggunakan pernyataan SQL berikut.

      ```
      CREATE TABLE docs_lab_table (a int PRIMARY KEY);
      ```

   b. Mengisi tabel dengan data yang dihasilkan menggunakan pernyataan SQL berikut.

      ```
      INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
      INSERT 0 5000
      ```

   c. Verifikasikan bahwa data ada dalam tabel menggunakan pernyataan SQL berikut.

      ```
      SELECT count(*) FROM docs_lab_table;
      ```

2. Identifikasi klaster DB Aurora PostgreSQL ini sebagai simpul penerbit, sebagai berikut.

   ```
   SELECT pglogical.create_node(
       node_name := 'docs_lab_provider',
       dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
    dbname=labdb');
    create_node
   -------------
      3410995529
   (1 row)
   ```

3. Tambahkan tabel yang ingin Anda replikasi ke set replikasi default. Untuk informasi set replikasi selengkapnya, lihat Replication sets dalam dokumentasi pglogical.

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  ---------------------------
  t
  (1 row)
```

Penyiapan simpul penerbit selesai. Anda sekarang dapat menyiapkan simpul pelanggan untuk menerima informasi terkini dari penerbit.

Untuk menyiapkan simpul pelanggan dan membuat langganan untuk menerima informasi terkini

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL telah disiapkan dengan ekstensi `pglogical`. Untuk informasi selengkapnya, lihat Menyiapkan ekstensi pglogical.

1. Gunakan `psql` untuk terhubung ke instans yang ingin Anda terima informasi terkini dari penerbit.

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Pada pelanggan klaster DB Aurora PostgreSQL, , buat tabel yang sama yang ada pada penerbit. Untuk contoh ini, tabelnya adalah `docs_lab_table`. Anda dapat membuat tabel sebagai berikut.

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. Verifikasi bahwa tabel ini kosong.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. Identifikasi klaster DB Aurora PostgreSQL ini sebagai simpul pelanggan, sebagai berikut.

```
SELECT pglogical.create_node(
```

```
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. Buat langganan.

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
1038357190
(1 row)
```

Saat Anda menyelesaikan langkah ini, data dari tabel pada penerbit akan dibuat dalam tabel pada pelanggan. Anda dapat memverifikasi bahwa tabel telah dibuat menggunakan query SQL berikut.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

Mulai dari titik ini, perubahan yang dilakukan pada tabel pada penerbit direplikasi ke tabel pada pelanggan.

## Membangun kembali replikasi logis setelah peningkatan besar

Sebelum Anda dapat melakukan upgrade versi utama dari cluster Aurora Postgre SQL DB RDS yang diatur sebagai node penerbit untuk replikasi logis, Anda harus menghapus semua slot replikasi, bahkan yang tidak aktif. Kami menyarankan Anda mengalihkan sementara transaksi database

dari node penerbit, menjatuhkan slot replikasi, meningkatkan cluster Aurora Postgre SQL DB, meningkatkan instance Postgre DB RDS untuk SQL Postgre, kembali dan memulai ulang replikasi.

Slot replikasi hanya di-hosting di simpul penerbit. Node SQL pelanggan Aurora Postgre dalam skenario replikasi logis tidak memiliki slot untuk dijatuhkan. Proses upgrade versi SQL utama Aurora Postgre mendukung peningkatan pelanggan ke versi utama baru SQL Postgre independen dari node penerbit. Namun, proses upgrade memang mengganggu proses replikasi dan mengganggu sinkronisasi WAL data antara node penerbit dan node pelanggan. Anda perlu membangun kembali replikasi logis antara penerbit dan pelanggan setelah peningkatan penerbit, pelanggan, atau keduanya. Prosedur berikut menunjukkan kepada Anda cara menentukan bahwa replikasi telah terganggu dan cara mengatasi masalah tersebut.

Menentukan bahwa replikasi logis telah terganggu

Anda dapat menentukan bahwa proses replikasi telah terganggu dengan menanyakan simpul penerbit atau simpul pelanggan, sebagai berikut.

Untuk memeriksa simpul penerbit

- Gunakan `psql` untuk terhubung ke simpul penerbit, lalu buat kueri fungsi `pg_replication_slots`. Perhatikan nilai di kolom aktif. Biasanya, ini akan kembali `t` (benar), menunjukkan bahwa replikasi aktif. Jika kueri kembali `f` (salah), ini merupakan indikasi bahwa replikasi ke pelanggan telah berhenti.

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

Untuk memeriksa simpul pelanggan

Pada simpul pelanggan, Anda dapat memeriksa status replikasi dengan tiga cara berbeda.

- Lihat melalui SQL log Postgre pada node pelanggan untuk menemukan pesan kegagalan. Log dapat mengidentifikasi kegagalan dengan pesan yang menyertakan kode keluar 1, sebagaimana ditunjukkan berikut.

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
```

```
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- Buat kueri fungsi `pg_replication_origin`. Hubungkan ke basis data pada simpul pelanggan menggunakan `psql` dan buat kueri fungsi `pg_replication_origin`, sebagai berikut.

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

  Set hasil kosong berarti replikasi telah terganggu. Biasanya, Anda akan melihat output seperti berikut.

```
   roident |                       roname
  ---------+-----------------------------------------------------
         1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
  (1 row)
```

- Buat kueri fungsi `pglogical.show_subscription_status` sebagaimana ditunjukkan pada contoh berikut.

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
---====----------------+--------+-----------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

  Output ini menunjukkan bahwa replikasi telah terganggu. Statusnya adalah `down`. Biasanya, output menunjukkan status sebagai `replicating`.

Jika proses replikasi logis terganggu, Anda dapat membangun kembali replikasi dengan mengikuti langkah-langkah berikut.

Untuk membangun kembali replikasi logis antara simpul penerbit dan pelanggan

Untuk membangun kembali replikasi, pertama-tama Anda memutuskan sambungan pelanggan dari simpul penerbit, lalu membangun kembali langganan, seperti yang diuraikan dalam langkah-langkah ini.

1. Hubungkan ke simpul pelanggan menggunakan `psql`, sebagai berikut.

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Nonaktifkan langganan menggunakan fungsi `pglogical.alter_subscription_disable`.

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. Dapatkan identifier simpul penerbit dengan menanyakan `pg_replication_origin`, sebagai berikut.

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+-------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. Gunakan respons dari langkah sebelumnya dengan perintah `pg_replication_origin_create` untuk menetapkan pengenal yang dapat digunakan oleh langganan saat dibuat kembali.

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                             1
(1 row)
```

5. Aktifkan langganan dengan meneruskan namanya menggunakan status `true`, sebagaimana ditunjukkan dalam contoh berikut.

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

Periksa status simpul. Statusnya harus `replicating` sebagaimana ditunjukkan dalam contoh ini.

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+-----------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

Periksa status slot replikasi pelanggan pada simpul penerbit. Kolom `active` slot harus kembali `t` (benar), menunjukkan bahwa replikasi telah dibuat kembali.

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                   slot_name              |      plugin      | slot_type | active
------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

Sebelum Anda dapat melakukan upgrade versi mayor pada instance penulis cluster Aurora Postgre SQL DB RDS untuk instance . Proses pra-pemeriksaan peningkatan versi utama akan memberi tahu Anda bahwa peningkatan tidak dapat dilanjutkan sampai slot dihapuskan sementara.

Untuk mengidentifikasi slot replikasi yang dibuat menggunakan ekstensi `pglogical`, masuk ke setiap basis data dan dapatkan nama simpul. Bila membuat kueri simpul pelanggan, Anda akan mendapatkan simpul penerbit dan pelanggan dalam output, sebagaimana ditunjukkan dalam contoh ini.

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

Anda dapat memperoleh detail tentang langganan dengan kueri berikut.

```
SELECT sub_name,sub_slot_name,sub_target
```

```
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

Anda sekarang dapat menghapus sementara langganan, sebagai berikut.

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

Setelah menghapus sementara langganan, Anda dapat menghapus simpul.

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

Anda dapat memverifikasi bahwa simpul tidak ada lagi, sebagai berikut.

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

## Referensi parameter untuk ekstensi pglogical

Dalam tabel, Anda dapat menemukan parameter yang terkait dengan ekstensi `pglogical`. Parameter seperti `pglogical.conflict_log_level` dan `pglogical.conflict_resolution` digunakan untuk menangani konflik pembaruan. Konflik dapat muncul saat perubahan dilakukan secara lokal ke tabel yang sama yang berlangganan terhadap perubahan dari penerbit. Konflik juga dapat terjadi selama berbagai skenario, seperti replikasi dua arah atau saat beberapa pelanggan mereplikasi dari penerbit yang sama. Untuk informasi lebih lanjut, lihat Replikasi dua SQLarah Postgre menggunakan pglogical.

| Parameter | Deskripsi |
| --- | --- |
| pglogical.batch_inserts | Melakukan penyisipan batch jika memungkinkan. Tidak diatur secara default. Ubah ke '1' untuk mengaktifkan, '0' untuk menonaktifkan. |
| pglogical.conflict_log_level | Menetapkan tingkat log yang digunakan untuk mencatat log konflik yang diselesaikan. Nilai string yang didukung adalah debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic. |
| pglogical.conflict_resolution | Menetapkan metode yang digunakan untuk menyelesaikan konflik saat konflik dapat diselesaikan. Nilai string yang didukung adalah kesalahan, apply_remote, keep_local, last_update_wins, first_update_wins. |
| pglogical.extra_connection_options | Opsi koneksi untuk ditambahkan ke semua koneksi simpul peer. |
| pglogical.synchronous_commit | nilai komit sinkron spesifik pglogical |
| pglogical.use_spi | Gunakan SPI (antarmuka pemrograman server) alih-alih tingkat rendah API untuk menerapkan perubahan. Atur ke '1' untuk mengaktifkan, '0' untuk menonaktifkan. Untuk informasi selengkapnyaSPI, lihat Antarmuka Pemrograman Server di dokumentasi PostgreSQL. |

Pembungkus data asing (FDW) adalah jenis ekstensi khusus yang menyediakan akses ke data eksternal. Misalnya, `oracle_fdw` ekstensi memungkinkan instans Aurora Postgre SQL DB Anda RDS untuk cluster Postgre DB .

Berikut ini, Anda dapat menemukan informasi tentang beberapa pembungkus data SQL asing Postgre yang didukung.

Topik

- Menggunakan ekstensi log_fdw untuk mengakses log DB menggunakan SQL
- Menggunakan ekstensi postgres_fdw untuk mengakses data eksternal

- Bekerja dengan SQL database Saya dengan menggunakan ekstensi mysql_fdw
- Bekerja dengan basis data Oracle menggunakan ekstensi oracle_fdw
- Bekerja dengan database SQL Server dengan menggunakan ekstensi tds_fdw

## Menggunakan ekstensi log_fdw untuk mengakses log DB menggunakan SQL

Cluster Aurora Postgre SQL DB RDS mendukung `log_fdw` ekstensi, yang dapat Anda gunakan untuk mengakses log mesin database Anda menggunakan antarmuka. SQL Ekstensi `log_fdw` ini memberikan dua fungsi yang memudahkan pembuatan tabel asing untuk log basis data:

- `list_postgres_log_files` – Mencantumkan file di direktori log basis data dan ukuran file dalam byte.
- `create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text)` – Membangun tabel asing untuk file yang ditentukan dalam basis data saat ini.

Semua fungsi yang dibuat oleh `log_fdw` dimiliki oleh `rds_superuser`. Anggota dengan peran `rds_superuser` dapat memberikan akses ke fungsi ini kepada pengguna basis data lain.

Secara default, file log dihasilkan oleh Amazon Aurora dalam format `stderr` (kesalahan standar), seperti yang ditentukan dalam `log_destination` parameter. Hanya ada dua opsi untuk parameter ini, `stderr` dan `csvlog` (nilai yang dipisahkan koma,CSV). Jika Anda menambahkan `csvlog` opsi ke parameter, Amazon Aurora menghasilkan keduanya `stderr` dan `csvlog` log. Hal ini dapat memengaruhi kapasitas penyimpanan pada klaster DB sehingga Anda perlu mengetahui parameter lain yang memengaruhi penanganan log. Untuk informasi selengkapnya, lihat Mengatur tujuan log (`stderr, csvlog`).

Salah satu manfaat pembuatan log `csvlog` adalah ekstensi `log_fdw` memungkinkan Anda membangun tabel asing dengan data yang terbagi dengan rapi menjadi beberapa kolom. Untuk melakukannya, instans Anda harus terkait dengan grup parameter DB kustom sehingga Anda dapat mengubah pengaturan `log_destination`. Untuk informasi selengkapnya tentang cara melakukannya, lihat .

Contoh berikut mengasumsikan bahwa parameter `log_destination` mencakup `cvslog`.

Untuk menggunakan ekstensi log_fdw

1. Instal ekstensi `log_fdw`.

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. Buat server log sebagai pembungkus data asing.

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. Pilih semua dari daftar file log.

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

Respons sampel sebagai berikut.

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. Membuat tabel dengan satu kolom 'log_entry' untuk file yang dipilih.

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

Respons tersebut tidak memberikan detail selain memberitahukan bahwa tabel telah ada.

```
----------------------------------
(1 row)
```

5. Pilih sampel file log. Kode berikut mengambil waktu log dan deskripsi pesan kesalahan.

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

Respons sampel sebagai berikut.

```
            log_time             |                                 message
```

```
---------------------------------
+-------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

## Menggunakan ekstensi postgres_fdw untuk mengakses data eksternal

Anda dapat mengakses data dalam tabel di server basis data jarak jauh dengan ekstensi postgres_fdw. Jika Anda mengatur koneksi jarak jauh dari instans DB PostgreSQL, akses juga tersedia untuk replika baca Anda.

Untuk menggunakan postgres_fdw agar dapat mengakses server basis data jarak jauh

1. Instal ekstensi postgres_fdw.

   ```
   CREATE EXTENSION postgres_fdw;
   ```

2. Buat server data asing menggunakan CREATE SERVER.

   ```
   CREATE SERVER foreign_server
   FOREIGN DATA WRAPPER postgres_fdw
   OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
   ```

3. Buat pemetaan pengguna untuk mengidentifikasi peran yang akan digunakan pada server jarak jauh.

   > **Important**
   >
   > Untuk menyunting kata sandi sehingga tidak muncul di log, atur `log_statement=none` pada tingkat sesi. Pengaturan pada tingkat parameter tidak menyunting kata sandi.

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. Buat tabel yang memetakan ke tabel pada server jarak jauh.

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

## Bekerja dengan SQL database Saya dengan menggunakan ekstensi mysql_fdw

Untuk mengakses database My SQL -compatible dari , Anda dapat menginstal dan menggunakan ekstensi. `mysql_fdw` Pembungkus data asing ini memungkinkan Anda bekerja dengan RDS untuk MySQL, Aurora My, MariaDBSQL, dan database My -compatible lainnya. SQL Koneksi dari cluster Aurora Postgre SQL DB RDS ke SQL database Saya dienkripsi berdasarkan upaya terbaik, tergantung pada konfigurasi klien dan server. Namun, Anda dapat menerapkan enkripsi jika diinginkan. Untuk informasi selengkapnya, lihat Menggunakan enkripsi bergerak dengan ekstensi.

`mysql_fdw`Ekstensi ini didukung di Amazon Aurora Postgre SQL versi 15.4, 14.9, 13.12, 12.16, versi 14.2, 13.6, dan rilis yang lebih tinggi. Ini mendukung memilih, menyisipkan, memperbarui, dan menghapus dari RDS untuk Postgre SQL DB ke tabel pada instance database My -kompatibel. SQL

Topik

- Menyiapkan Aurora Postgre Anda untuk Postgre DB SQL RDS untuk menggunakan
- 
- Menggunakan enkripsi bergerak dengan ekstensi

Menyiapkan Aurora Postgre Anda untuk Postgre DB SQL RDS untuk menggunakan

Menyiapkan `mysql_fdw` ekstensi pada cluster Aurora Postgre SQL DB Anda DB cluster DB Anda dan kemudian membuat titik koneksi ke Saya. SQL Untuk tugas itu, Anda harus memiliki detail berikut tentang instance My SQL DB:

- Nama host atau titik akhir. Untuk cluster Aurora My SQL DB RDS , Anda dapat menemukan titik akhir dengan menggunakan Konsol. Pilih tab Konektivitas & keamanan, lalu lihat di bagian "Titik akhir dan port".
- Nomor port. Nomor port default untuk My SQL adalah 3306.
- Nama basis data. Pengidentifikasi DB.

Anda juga perlu menyediakan akses pada grup keamanan atau daftar kontrol akses (ACL) untuk SQL port Saya, 3306. Baik cluster Aurora Postgre SQL DB dan cluster Aurora My DB untuk instance Postgre SQL DB . Jika akses tidak dikonfigurasi dengan benar, saat Anda mencoba menyambung ke tabel yang SQL kompatibel dengan saya, Anda akan melihat pesan kesalahan yang mirip dengan berikut ini:

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

Dalam prosedur berikut, Anda (sebagai akun `rds_superuser`) akan membuat server asing. Kemudian, Anda akan memberikan akses ke server asing untuk pengguna tertentu. Pengguna ini kemudian membuat pemetaan mereka sendiri ke akun SQL pengguna Saya yang sesuai untuk bekerja dengan instans SQL DB Saya.

Untuk menggunakan mysql_fdw untuk mengakses server database Saya SQL

1. Connect ke instans Postgre SQL DB Anda menggunakan akun yang memiliki peran. `rds_superuser` Jika Anda menerima default saat membuat cluster Aurora Postgre SQL DB Postgre DB, nama penggunanya adalah`postgres`, dan Anda dapat terhubung menggunakan alat baris perintah sebagai berikut: SQL `psql`

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Instal ekstensi `mysql_fdw` sebagai berikut:

   ```
   postgres=> CREATE EXTENSION mysql_fdw;
   CREATE EXTENSION
   ```

Setelah ekstensi diinstal pada cluster Aurora Postgre SQL DB Anda , Anda mengatur server asing yang menyediakan koneksi ke database Saya. SQL

Untuk membuat server asing

Lakukan tugas-tugas ini pada cluster Aurora Postgre SQL DB instance Postgre DB. SQL Langkah-langkah ini mengasumsikan bahwa Anda terhubung sebagai pengguna dengan hak istimewa `rds_superuser`, seperti `postgres`.

1. Buat server asing di cluster Aurora Postgre SQL DB instance Postgre DB: SQL

```
postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-name.111122223333.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Berikan akses pengguna yang sesuai ke server asing. Pengguna yang diberi akses harus pengguna non-administrator, yaitu pengguna tanpa peran `rds_superuser`.

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

SQLPengguna Postgre membuat dan mengelola koneksi mereka sendiri ke SQL database Saya melalui server asing.

Misalkan Anda memiliki tabel sederhana pada instance Aurora Postgre SQL DB instance Postgre DB. SQL Aurora Postgre Anda SQL ingin menanyakan (`SELECT`),, `INSERTUPDATE`, dan item di tabel itu. `DELETE` Asumsikan bahwa `mysql_fdw` ekstensi dibuat pada instans Postgre SQL DB AndaRDS, seperti yang dijelaskan dalam prosedur sebelumnya. Setelah Anda terhubung ke instans RDS for Postgre SQL DB sebagai pengguna yang memiliki `rds_superuser` hak istimewa, Anda dapat melanjutkan dengan langkah-langkah berikut.

1. Pada instans Aurora Postgre SQL DB RDS untuk instance , buat server asing:

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Izinkan penggunaan kepada pengguna yang tidak memiliki izin `rds_superuser`, misalnya `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. Connect sebagai *user1*, lalu buat pemetaan ke SQL pengguna Saya:

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. Buat tabel asing yang ditautkan ke SQL tabel Saya:

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. Jalankan kueri sederhana pada tabel asing:

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. Anda dapat menambahkan, mengubah, dan menghapus data dari SQL tabel Saya. Sebagai contoh:

```
test=> INSERT INTO mytab values (2, 'mango');
INSERT 0 1
```

   Jalankan lagi kueri `SELECT` untuk melihat hasilnya:

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

Menggunakan enkripsi bergerak dengan ekstensi

Koneksi ke My SQL dari Aurora Postgre SQL menggunakan enkripsi dalam transit (/) secara default. TLS SSL Namun, koneksi akan kembali ke non-enkripsi saat konfigurasi klien dan server berbeda. Anda dapat menerapkan enkripsi untuk semua koneksi keluar dengan menentukan `REQUIRE SSL`

opsi pada RDS untuk akun pengguna SayaSQL. Pendekatan yang sama ini juga berfungsi untuk akun pengguna MariaDB dan Aurora MySQL.

Untuk akun SQL pengguna saya yang dikonfigurasi`REQUIRE SSL`, upaya koneksi gagal jika koneksi aman tidak dapat dibuat.

Untuk menerapkan enkripsi untuk akun pengguna SQL database Saya yang ada, Anda dapat menggunakan `ALTER USER` perintah. Sintaksnya bervariasi, tergantung pada SQL versi Saya, seperti yang ditunjukkan pada tabel berikut. Untuk informasi lebih lanjut, lihat ALTERUSERdi Manual SQL Referensi Saya.

| SQL5.7 saya, 8.0 saya SQL | SQL5.6 saya |
| --- | --- |
| `ALTER USER 'user'@'%' REQUIRE SSL;` | `GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;` |

Untuk informasi selengkapnya tentang ekstensi `mysql_fdw`, lihat dokumentasi mysql_fdw.

## Bekerja dengan basis data Oracle menggunakan ekstensi oracle_fdw

Untuk mengakses basis data Oracle dari klaster DB Aurora PostgreSQL , Anda dapat menginstal dan menggunakan ekstensi `oracle_fdw`. Ekstensi ini adalah pembungkus data asing untuk basis data Oracle. Untuk mempelajari selengkapnya tentang ekstensi ini, lihat dokumentasi oracle_fdw.

Ekstensi `oracle_fdw` didukung pada Aurora PostgreSQL 12.7 (Amazon Aurora rilis 4.2) dan versi yang lebih baru.

Topik

- Mengaktifkan ekstensi oracle_fdw
- Contoh: Menggunakan server asing yang terhubung ke basis data Amazon RDS for Oracle
- Bekerja dengan enkripsi bergerak
- Memahami tampilan dan izin pg_user_mappings

### Mengaktifkan ekstensi oracle_fdw

Untuk menggunakan ekstensi oracle_fdw, lakukan prosedur berikut.

Untuk mengaktifkan ekstensi oracle_fdw

- Jalankan perintah berikut menggunakan akun yang memiliki izin `rds_superuser`.

```
CREATE EXTENSION oracle_fdw;
```

Contoh: Menggunakan server asing yang terhubung ke basis data Amazon RDS for Oracle

Contoh berikut ini menunjukkan penggunaan server asing yang terhubung ke basis data Amazon RDS for Oracle.

Untuk membuat server asing yang terhubung ke basis data RDS for Oracle

1. Perhatikan hal-hal berikut ini pada instans DB RDS for Oracle:
   - Titik Akhir
   - Port
   - Nama basis data
2. Membuat server asing.

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. Izinkan penggunaan kepada pengguna yang tidak memiliki hak istimewa `rds_superuser`, misalnya `user1`.

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. Hubungkan sebagai `user1`, lalu buat pemetaan kepada pengguna Oracle.

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. Buat tabel asing yang dihubungkan ke tabel Oracle.

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
```

```
CREATE FOREIGN TABLE
```

6. Buat kueri tabel asing.

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

Jika kueri melaporkan kesalahan berikut, periksa grup keamanan dan daftar kontrol akses (ACL) untuk memastikan bahwa kedua instans dapat berkomunikasi.

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

Bekerja dengan enkripsi bergerak

PostgreSQL-to-Oracle enkripsi dalam perjalanan didasarkan pada kombinasi parameter konfigurasi klien dan server. Untuk contoh menggunakan Oracle 21c, lihat Tentang Nilai untuk Melakukan Negosiasi Enkripsi dan Integritas pada dokumentasi Oracle. Klien yang digunakan untuk oracle_fdw di Amazon RDS dikonfigurasi `ACCEPTED` dengan, artinya enkripsi bergantung pada konfigurasi server database Oracle dan menggunakan Oracle Security Library (libnnz) untuk enkripsi.

Jika basis data Anda menggunakan RDS for Oracle, lihat Enkripsi jaringan asli Oracle untuk mengonfigurasi enkripsi.

Memahami tampilan dan izin pg_user_mappings

Katalog PostgreSQL `pg_user_mapping` menyimpan pemetaan dari pengguna Aurora PostgreSQL ke pengguna di server data asing (jarak jauh). Akses ke katalog dibatasi, tetapi Anda menggunakan tampilan `pg_user_mappings` untuk melihat pemetaan. Berikut ini Anda dapat menemukan contoh yang menunjukkan bagaimana izin berlaku dengan contoh basis data Oracle, tetapi informasi ini biasanya berlaku untuk pembungkus data asing apa pun.

Pada output berikut, Anda dapat menemukan peran dan izin yang dipetakan kepada tiga pengguna contoh yang berbeda. Pengguna `rdssu1` dan `rdssu2` adalah anggota dengan peran `rds_superuser`, sedangkan `user1` tidak. Contoh menggunakan metacommand `psql \du` untuk daftar peran yang ada.

```
test=>  \du
                                                              List of roles
    Role name    |                         Attributes                         |
                 Member of
-----------------+------------------------------------------------------------
+-----------------------------------------------------------
 rdssu1          |                                                            |
 {rds_superuser}
 rdssu2          |                                                            |
 {rds_superuser}
 user1           |                                                            | {}
```

Semua pengguna, termasuk pengguna yang memiliki hak istimewa `rds_superuser`, diizinkan untuk melihat pemetaan pengguna (`umoptions`) mereka sendiri di tabel `pg_user_mappings`. Seperti yang ditunjukkan pada contoh berikut, saat `rdssu1` mencoba untuk mendapatkan semua pemetaan pengguna, kesalahan ditampilkan meskipun hak istimewa `rdssu1` `rds_superuser`:

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

Berikut adalah beberapa contoh.

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)
```

```
test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

Karena perbedaan implementasi antara `information_schema._pg_user_mappings` dan `pg_catalog.pg_user_mappings`, maka `rds_superuser` yang dibuat secara manual memerlukan izin tambahan untuk dapat melihat kata sandi di `pg_catalog.pg_user_mappings`.

`rds_superuser` tidak memerlukan izin tambahan untuk dapat melihat kata sandi di `information_schema._pg_user_mappings`.

Pengguna yang tidak memiliki peran `rds_superuser` dapat melihat kata sandi `pg_user_mappings` hanya dalam kondisi berikut:

- Pengguna saat ini adalah pengguna yang dipetakan dan memiliki server atau memegang hak istimewa `USAGE` atas server tersebut.
- Pengguna saat ini adalah pemilik server dan pemetaannya untuk `PUBLIC`.

## Bekerja dengan database SQL Server dengan menggunakan ekstensi tds_fdw

Anda dapat menggunakan SQL `tds_fdw` ekstensi Postgre untuk mengakses database yang mendukung protokol aliran data tabular (TDS), seperti database Sybase dan Microsoft Server. SQL Pembungkus data asing ini memungkinkan Anda terhubung dari cluster SQL DB Postgre DB ke database yang menggunakan protokol, TDS termasuk Amazon untuk Microsoft Server. RDS RDS SQL Untuk informasi selengkapnya, lihat dokumentasi tds-fdw/tds_fdw di. GitHub

`tds_fdw`Ekstensi ini didukung pada Amazon Aurora Postgre SQL versi 13.6 dan rilis yang lebih tinggi.

Menyiapkan Aurora Postgre SQL DB Anda untuk menggunakan ekstensi tds_fdw

Dalam prosedur berikut, Anda dapat menemukan contoh pengaturan dan penggunaan `tds_fdw` dengan cluster. SQL Sebelum Anda dapat terhubung ke database SQL Server menggunakan`tds_fdw`, Anda perlu mendapatkan rincian berikut untuk contoh:

- Nama host atau titik akhir. Untuk instans RDS for SQL Server DB, Anda dapat menemukan titik akhir dengan menggunakan Console. Pilih tab Konektivitas & keamanan, lalu lihat di bagian "Titik akhir dan port".
- Nomor port. Nomor port default untuk Microsoft SQL Server adalah 1433.
- Nama basis data. Pengidentifikasi DB.

Anda juga perlu menyediakan akses pada grup keamanan atau daftar kontrol akses (ACL) untuk port SQL Server, 1433. Baik cluster Aurora Postgre SQL DB RDS RDS untuk SQL Server DB memerlukan akses ke port 1433. Jika akses tidak dikonfigurasi dengan benar, ketika Anda mencoba untuk query Microsoft SQL Server Anda melihat pesan galat berikut:

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

Untuk menggunakan tds_fdw untuk terhubung ke database Server SQL

1. Connect ke instans Postgre SQL DB cluster utama menggunakan akun yang memiliki peran: `rds_superuser`

   ```
   psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=test --password
   ```

2. Instal ekstensi `tds_fdw`:

   ```
   test=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

Setelah ekstensi diinstal pada cluster Aurora Postgre SQL DB Anda , Anda mengatur server asing.

Untuk membuat server asing

Lakukan tugas-tugas ini di cluster Aurora Postgre SQL DB RDS menggunakan akun yang memiliki hak istimewa. `rds_superuser`

1. Buat server asing di cluster Aurora Postgre SQL DB instance Postgre DB: SQL

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
 (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
 'tds_fdw_testing');
CREATE SERVER
```

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. Berikan izin kepada pengguna yang tidak memiliki hak istimewa peran `rds_superuser`, misalnya `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. Connect as user1 dan buat pemetaan ke pengguna SQL Server:

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. Buat tabel asing yang ditautkan ke tabel SQL Server:

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. Buat kueri tabel asing:

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

Gunakan enkripsi bergerak untuk koneksi

Koneksi dari Aurora Postgre SQL RDS ke SQL Server menggunakan enkripsi dalam transit (TLS/SSL) tergantung pada konfigurasi database Server. SQL Jika SQL Server tidak dikonfigurasi untuk

enkripsi, SQL klien Postgre RDS untuk membuat permintaan ke database SQL Server akan kembali ke tidak terenkripsi.

Anda dapat menerapkan enkripsi untuk koneksi ke RDS instans SQL Server DB dengan menyetel parameter. `rds.force_ssl` Untuk mempelajari caranya, lihat Memaksa koneksi ke instans DB Anda untuk digunakan SSL. Untuk informasi selengkapnya tentangSSL/TLSkonfigurasi RDS untuk SQL Server, lihat Menggunakan SSL dengan instans Microsoft SQL Server DB.

# Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL

Ekstensi Bahasa Tepercaya untuk PostgreSQL adalah kit pengembangan sumber terbuka untuk mendesain ekstensi PostgreSQL. Ini memungkinkan Anda untuk membangun ekstensi PostgreSQL performa tinggi dan menjalankannya dengan aman di klaster DB Aurora PostgreSQL. Dengan menggunakan Ekstensi Bahasa Tepercaya (TLE) untuk PostgreSQL, Anda dapat membuat ekstensi PostgreSQL yang mengikuti pendekatan terdokumentasi untuk memperluas fungsionalitas PostgreSQL. Lihat informasi selengkapnya, lihat Packaging Related Objects into an Extension dalam dokumentasi PostgreSQL.

Salah satu manfaat utama TLE adalah Anda dapat menggunakannya di lingkungan yang tidak menyediakan akses ke sistem file yang mendasari instans PostgreSQL. Sebelumnya, penginstalan ekstensi baru memerlukan akses ke sistem file. TLE menghilangkan batasan ini. Ini menyediakan lingkungan pengembangan untuk membuat ekstensi baru untuk basis data PostgreSQL apa pun, termasuk yang berjalan di klaster DB Aurora PostgreSQL.

TLE dirancang untuk mencegah akses ke sumber daya yang tidak aman untuk ekstensi yang Anda buat menggunakan TLE. Lingkungan runtime-nya membatasi dampak kerusakan ekstensi apa pun ke koneksi basis data tunggal. TLE juga memberi administrator basis data kontrol terperinci atas siapa saja yang dapat menginstal ekstensi, dan memberikan model izin untuk menjalankannya.

TLE didukung pada Aurora PostgreSQL versi 14.5 dan yang lebih tinggi.

Lingkungan pengembangan dan runtime Ekstensi Bahasa Tepercaya dikemas sebagai ekstensi PostgreSQL `pg_tle`, versi 1.0.1. Ini mendukung pembuatan ekstensi di JavaScript, Perl, Tcl, PL/PGSQL, dan SQL. Anda menginstal ekstensi `pg_tle` di klaster DB Aurora PostgreSQL dengan cara yang sama seperti Anda menginstal ekstensi PostgreSQL lainnya. Setelah `pg_tle` disiapkan, developer dapat menggunakannya untuk membuat ekstensi PostgreSQL baru, yang dikenal sebagai ekstensi TLE.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara mengatur Ekstensi Bahasa Tepercaya dan cara memulai membuat ekstensi TLE Anda sendiri.

Topik


- Terminologi
- Persyaratan untuk menggunakan Ekstensi Bahasa Tepercaya untuk Postgre SQL
-

- Ikhtisar Ekstensi Bahasa Tepercaya untuk Postgre SQL
- 
- Menjatuhkan TLE ekstensi Anda dari database
- Menghapus Instalasi Ekstensi Bahasa Tepercaya untuk Postgre SQL
- Menggunakan hook PostgreSQL dengan ekstensi TLE
- Referensi fungsi untuk Ekstensi Bahasa Tepercaya untuk Postgre SQL
- Referensi hook untuk Trusted Language Extensions for PostgreSQL

## Terminologi

Untuk membantu Anda lebih memahami Ekstensi Bahasa Tepercaya, lihat glosarium berikut untuk istilah yang digunakan dalam topik ini.

Ekstensi Bahasa Tepercaya untuk Postgre SQL

Ekstensi Bahasa Tepercaya untuk Postgre SQL adalah nama resmi kit pengembangan open source yang dikemas sebagai ekstensi. `pg_tle` Ini tersedia untuk digunakan pada sistem Postgre SQL apa pun. Untuk informasi selengkapnya, lihat aws/pg_tle di. GitHub

Ekstensi Bahasa Terpercaya

Trusted Language Extensions adalah singkatan dari Trusted Language Extensions for PostgreSQL. Nama singkat ini dan singkatannya (TLE) juga digunakan dalam dokumentasi ini.

bahasa tepercaya

Bahasa tepercaya adalah bahasa pemrograman atau skrip yang memiliki atribut keamanan tertentu. Misalnya, bahasa tepercaya biasanya membatasi akses ke sistem file, dan membatasi penggunaan properti jaringan tertentu. Kit TLE pengembangan dirancang untuk mendukung bahasa tepercaya. Postgre SQL mendukung beberapa bahasa berbeda yang digunakan untuk membuat ekstensi tepercaya atau tidak tepercaya. Sebagai contoh, lihat PL/Perl Tepercaya dan Tidak Tepercaya di dokumentasi Postgre. SQL Saat Anda membuat ekstensi menggunakan Ekstensi Bahasa Tepercaya, ekstensi tersebut akan menggunakan mekanisme bahasa tepercaya secara inheren.

TLEekstensi

TLEEkstensi adalah ekstensi Postgre SQL yang dibuat dengan menggunakan kit pengembangan Trusted Language Extensions (TLE).

# Persyaratan untuk menggunakan Ekstensi Bahasa Tepercaya untuk Postgre SQL

Berikut ini adalah persyaratan untuk menyiapkan dan menggunakan kit TLE pengembangan.

- Versi Aurora Postgre RDS untuk SQL versi  14.5 RDS untuk Postgre SQL versi 13.12 dan versi 13 yang lebih tinggi, 14.5 .
  - Jika Anda perlu memutakhirkan cluster Aurora Postgre SQL DB Anda, RDS untuk contoh SQL Postgre, . Meningkatkan klaster DB Amazon Aurora PostgreSQL
  - Jika Anda belum memiliki DB yang menjalankan PostgreSQL, Anda dapat membuatnya. Untuk mengetahui informasi selengkapnya, lihat Membuat dan terhubung ke klaster DB Aurora PostgreSQL.
- Memerlukan hak istimewa **`rds_superuser`** – Untuk menyiapkan dan mengonfigurasi ekstensi `pg_tle`, peran pengguna basis data Anda harus memiliki izin peran `rds_superuser`. Secara default, peran ini diberikan kepada `postgres` pengguna yang membuat cluster Aurora SQL Postgre DB.
- Memerlukan grup parameter DB khusus - Kluster DB Aurora Postgre SQL Anda harus dikonfigurasi dengan grup parameter DB khusus. Anda menggunakan grup parameter DB kustom untuk instance penulis cluster Aurora SQL Postgre DB Anda.
  - Jika cluster Aurora Postgre SQL DB Anda tidak dikonfigurasi dengan grup parameter DB khusus, Anda harus membuatnya dan mengaitkannya dengan instance penulis cluster Aurora Postgre DB Anda. SQL Untuk ringkasan langkah-langkahnya, lihat Membuat dan menerapkan grup parameter DB kustom.
  - Jika cluster Aurora Postgre SQL DB RDS sudah dikonfigurasi menggunakan grup parameter DB kustom, Anda dapat mengatur Ekstensi Bahasa Tepercaya. Untuk detailnya, lihat .

## Membuat dan menerapkan grup parameter DB kustom

Gunakan langkah-langkah berikut untuk membuat grup parameter DB kustom dan konfigurasikan cluster Aurora Postgre SQL DB Anda untuk instans SQL Postgre DB RDS menggunakannya.

Konsol

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

2. Pilih grup Parameter dari RDS menu Amazon.
3. Pilih Buat grup parameter.
4. Di halaman Detail grup parameter, masukkan informasi berikut.
   - Untuk Keluarga grup Parameter, pilih aurora-postgresql14.
   - Untuk Jenis, pilih Grup Parameter DB.
   - Untuk Nama grup, berikan grup parameter Anda nama yang bermakna dalam konteks operasi Anda.
   - Untuk Deskripsi, masukkan deskripsi yang berguna sehingga orang lain di tim Anda dapat dengan mudah menemukannya.
5. Pilih Buat. Grup parameter DB kustom Anda dibuat di Wilayah AWS. Anda sekarang dapat memodifikasi cluster Aurora Postgre SQL DB Anda untuk menggunakannya dengan mengikuti langkah selanjutnya.
6. Pilih Database dari RDS menu Amazon.
7. Pilih cluster Aurora Postgre SQL DB RDS yang ingin Anda gunakan TLE dari antara yang terdaftar, lalu pilih Modify.
8. Di halaman Ubah pengaturan klaster DB, cari Opsi basis data dan gunakan pemilih untuk memilih grup parameter DB kustom Anda.
9. Pilih Lanjutkan untuk menyimpan perubahan.
10. Pilih Terapkan segera sehingga Anda dapat melanjutkan pengaturan cluster Aurora Postgre SQL DB untuk instans Postgre DB RDS digunakan. SQL TLE

Untuk melanjutkan penyiapan sistem untuk Ekstensi Bahasa Tepercaya, lihat .

Untuk informasi selengkapnya tentang cara menggunakan Grup parameter DB dan klaster DB, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

AWS CLI

Anda dapat menghindari menentukan `--region` argumen ketika Anda menggunakan CLI perintah dengan mengkonfigurasi AWS CLI dengan default Anda Wilayah AWS. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi di AWS Command Line Interface Panduan Pengguna.

1. Gunakan create-db-parameter-group AWS CLI Wilayah AWS. Perhatikan bahwa dalam langkah ini Anda membuat grup parameter DB untuk diterapkan ke instance penulis cluster Aurora SQL Postgre DB Anda.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Untuk Windows:

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
     --db-parameter-group-family aurora-postgresql14 ^
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Grup parameter DB kustom Anda tersedia di Wilayah AWS, sehingga Anda dapat memodifikasi instance penulis cluster Aurora Postgre SQL DB Anda untuk instance Postgre DB RDS .

2. Gunakan modify-db-instance AWS CLI perintah untuk menerapkan grup parameter DB kustom Anda ke instance penulis cluster Aurora SQL Postgre DB Anda. Perintah ini langsung mem-boot ulang instans yang aktif.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds modify-db-instance \
     --region aws-region \
     --db-instance-identifier your-writer-instance-name \
     --db-parameter-group-name custom-params-for-pg-tle \
     --apply-immediately
   ```

   Untuk Windows:

   ```
   aws rds modify-db-instance ^
     --region aws-region ^
     --db-instance-identifier your-writer-instance-name ^
   ```

```
    --db-parameter-group-name custom-params-for-pg-tle ^
    --apply-immediately
```

Untuk melanjutkan penyiapan sistem untuk Ekstensi Bahasa Tepercaya, lihat .

Untuk informasi selengkapnya, lihat Grup parameter DB untuk instans Aurora DB Amazon .

Anda dapat menggunakan AWS Management Console atau AWS CLI untuk langkah-langkah ini.

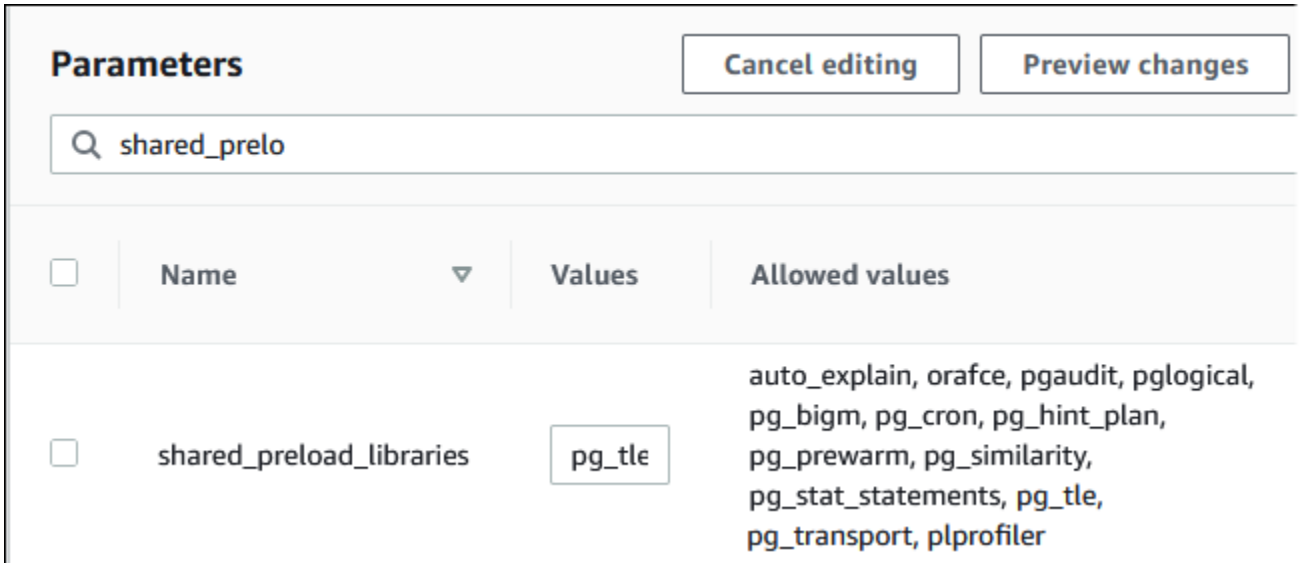
Saat menyiapkan Ekstensi Bahasa Tepercaya di klaster SQL DB Aurora Postgre RDS , Anda menginstalnya di database tertentu untuk digunakan oleh pengguna database yang memiliki izin pada database tersebut.

Konsol

Untuk menyiapkan Ekstensi Bahasa Tepercaya

Lakukan langkah-langkah berikut menggunakan akun yang merupakan anggota grup `rds_superuser` (peran).

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih instance Writer cluster Aurora SQL Postgre DB Anda untuk Postgre DB. SQL
3. Buka tab Konfigurasi untuk instance penulis cluster Aurora Postgre SQL DB Anda. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter khusus yang terkait dengan cluster Aurora SQL Postgre DB Anda.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pg_tle` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.




8. Reboot instance penulis cluster Aurora Postgre SQL DB Anda RDS sehingga perubahan Anda pada parameter berlaku. `shared_preload_libraries`
9. Ketika instans tersedia, verifikasi bahwa `pg_tle` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda,  dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. Dengan ekstensi `pg_tle` yang diinisialisasi, Anda kini dapat membuat ekstensi.

```
CREATE EXTENSION pg_tle;
```

Anda dapat memverifikasi bahwa ekstensi diinstal dengan menggunakan metacommand `psql` berikut.

```
labdb=> \dx
                        List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+-------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. Berikan `pgtle_admin` peran ke nama pengguna utama yang Anda buat untuk cluster RDS Aurora Postgre SQL DB Anda untuk instans Postgre DB . Jika Anda menerima opsi default-nya, berarti nilainya `postgres`.

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

Anda dapat memverifikasi bahwa pemberian telah terjadi dengan menggunakan metacommand `psql` seperti yang ditunjukkan pada contoh berikut. Hanya peran `pgtle_admin` dan `postgres` yang ditampilkan dalam output. Untuk informasi selengkapnya, lihat Memahami peran dan izin PostgreSQL.

```
labdb=> \du
                          List of roles
    Role name    |           Attributes            |               Member of
-----------------+---------------------------------
+-----------------------------------
pgtle_admin     | Cannot login                     | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. Tutup sesi `psql` menggunakan metacommand `\q`.

```
\q
```

Untuk mulai membuat TLE ekstensi, lihatContoh: Membuat ekstensi bahasa tepercaya menggunakan SQL.

AWS CLI

Anda dapat menghindari menentukan `--region` argumen ketika Anda menggunakan CLI perintah dengan mengkonfigurasi AWS CLI dengan default Anda Wilayah AWS. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi di AWS Command Line Interface Panduan Pengguna.

Untuk menyiapkan Ekstensi Bahasa Tepercaya

1. Gunakan modify-db-parameter-group AWS CLI perintah untuk `pg_tle` menambah `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan reboot-db-instance AWS CLI perintah untuk me-reboot instance penulis cluster Aurora Postgre SQL DB Anda RDS untuk instance Postgre DB perpustakaan. `pg_tle`

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Saat instans tersedia, verifikasikan bahwa `pg_tle` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora Postgre SQL DB Anda, dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

   Dengan `pg_tle` diinisialisasi, Anda sekarang dapat membuat ekstensi.

```
CREATE EXTENSION pg_tle;
```

4. Berikan `pgtle_admin` peran ke nama pengguna utama yang Anda buat untuk cluster RDS Aurora Postgre SQL DB Anda untuk instans Postgre DB . Jika Anda menerima opsi default-nya, berarti nilainya `postgres`.

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

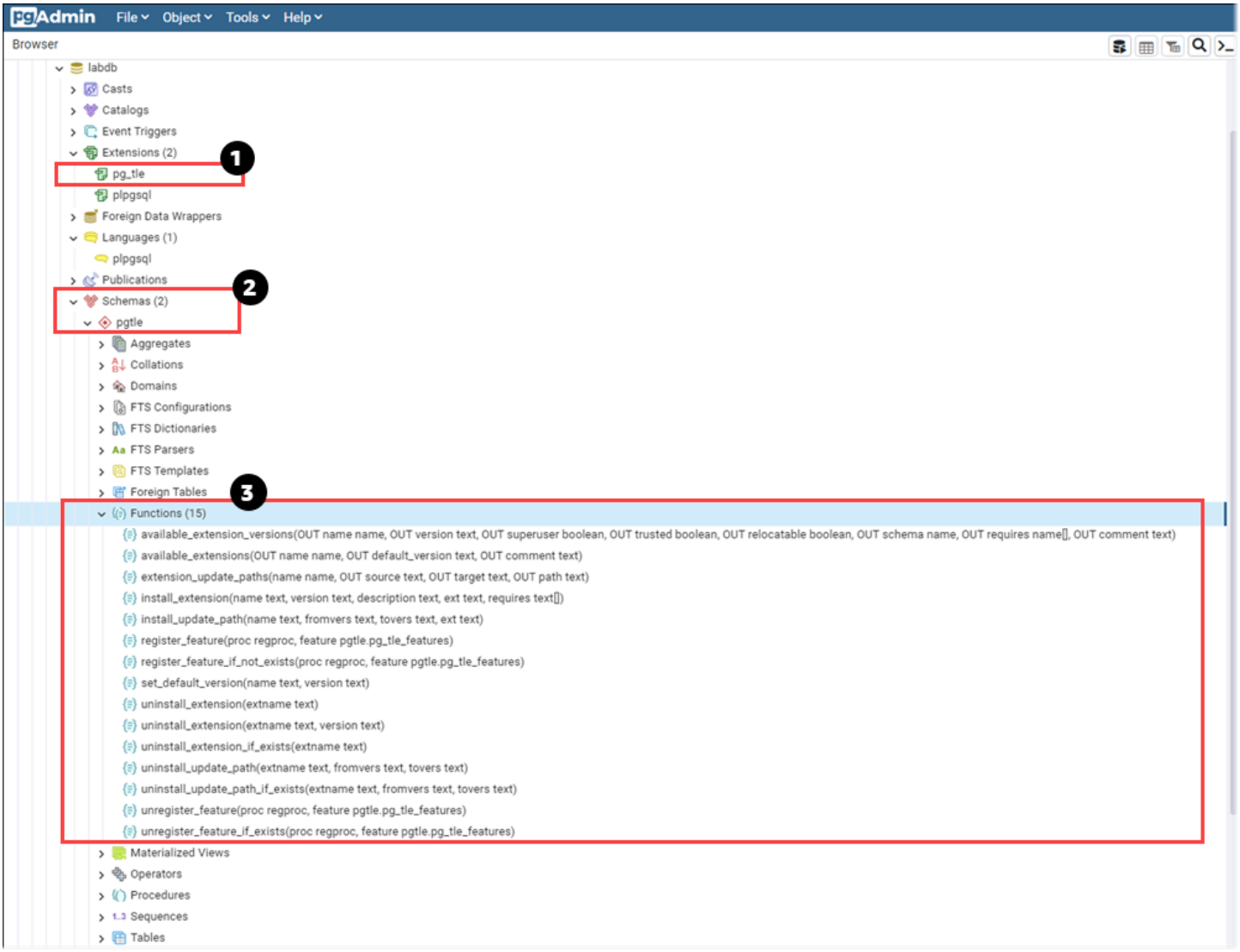
5. Tutup sesi `psql` seperti berikut.

```
labdb=> \q
```

Untuk mulai membuat TLE ekstensi, lihatContoh: Membuat ekstensi bahasa tepercaya menggunakan SQL.

## Ikhtisar Ekstensi Bahasa Tepercaya untuk Postgre SQL

Ekstensi Bahasa Tepercaya untuk Postgre SQL adalah ekstensi Postgre yang Anda instal di cluster Aurora Postgre DB Anda RDS untuk instans Postgre SQL DB lainnya. SQL Pada gambar berikut dari database contoh di alat pgAdmin klien, Anda dapat melihat beberapa komponen yang terdiri dari `pg_tle` ekstensi.




Anda dapat melihat detail berikut.

1. Kit SQL pengembangan Trusted Language Extensions (TLE) untuk Postgre dikemas sebagai ekstensi. `pg_tle` Dengan demikian, `pg_tle` ditambahkan ke ekstensi yang tersedia untuk basis data tempat kit ini diinstal.
2. TLEmemiliki skema sendiri,`pgtle`. Skema ini berisi fungsi pembantu (3) untuk menginstal dan mengelola ekstensi yang Anda buat.
3. TLEmenyediakan lebih dari selusin fungsi pembantu untuk menginstal, mendaftar, dan mengelola ekstensi Anda. Untuk mempelajari selengkapnya tentang fungsi ini, lihat Referensi fungsi untuk Ekstensi Bahasa Tepercaya untuk Postgre SQL.

Komponen lain dari ekstensi `pg_tle` meliputi:

- Peran **`pgtle_admin`** – Peran `pgtle_admin` dibuat jika ekstensi `pg_tle` diinstal. Peran ini diistimewakan dan harus diperlakukan semestinya. Sebaiknya Anda mengikuti prinsip hak akses paling rendah saat memberikan peran `pgtle_admin` kepada pengguna basis data. Dengan kata lain, berikan `pgtle_admin` peran hanya kepada pengguna database yang diizinkan untuk membuat, menginstal, dan mengelola TLE ekstensi baru, sepertipostgres.
- **`pgtle.feature_info`**Tabel — `pgtle.feature_info` Tabel adalah tabel yang dilindungi yang berisi informasi tentang AndaTLEs, kait, dan prosedur dan fungsi tersimpan khusus yang mereka gunakan. Jika Anda memiliki hak istimewa `pgtle_admin`, gunakan fungsi Ekstensi Bahasa Tepercaya berikut untuk menambahkan dan memperbarui informasi tersebut dalam tabel.
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

Anda dapat menginstal ekstensi apa pun yang Anda buat TLE di cluster Aurora Postgre SQL DB apa pun DB yang memiliki ekstensi yang diinstal. `pg_tle` `pg_tle`Ekstensi dicakup ke SQL database Postgre di mana ia diinstal. Ekstensi yang Anda buat menggunakan TLE cakupan ke database yang sama.

Gunakan berbagai `pgtle` fungsi untuk menginstal kode yang membentuk TLE ekstensi Anda. Semua fungsi Ekstensi Bahasa Tepercaya berikut memerlukan peran `pgtle_admin`.

- pgtle.install_extension
- pgtle.install_update_path

- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(nama)
- pgtle.uninstall_extension (nama, versi)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

## Contoh: Membuat ekstensi bahasa tepercaya menggunakan SQL

Contoh berikut menunjukkan cara membuat TLE ekstensi bernama `pg_distance` yang berisi beberapa SQL fungsi untuk menghitung jarak menggunakan rumus yang berbeda. Dalam daftar, Anda dapat menemukan fungsi untuk menghitung jarak Manhattan dan fungsi untuk menghitung jarak Euclidean. Untuk informasi selengkapnya tentang perbedaan antara formula ini, lihat Geometri taksi dan Geometri Euclidean di Wikipedia.

Anda dapat menggunakan contoh ini di cluster Aurora Postgre SQL DB Anda sendiri jika Anda memiliki `pg_tle` ekstensi yang disiapkan seperti yang dijelaskan dalam.

> **Note**
>
> Untuk mengikuti prosedur ini, Anda harus memiliki hak istimewa peran `pgtle_admin`.

Untuk membuat contoh TLE ekstensi

Langkah-langkah berikut menggunakan contoh basis data bernama `labdb`. Basis data ini milik pengguna utama `postgres`. Peran `postgres` juga memiliki izin peran `pgtle_admin`.

1. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Buat TLE ekstensi bernama `pg_distance` dengan menyalin kode berikut dan menempelkannya ke konsol `psql` sesi Anda.

```
SELECT pgtle.install_extension
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

Anda akan melihat output seperti berikut.

```
install_extension
---------------
 t
(1 row)
```

Artefak yang membentuk ekstensi `pg_distance` sekarang diinstal di basis data Anda. Artefak ini mencakup file kontrol dan kode untuk ekstensi, yang merupakan item yang harus ada sehingga ekstensi dapat dibuat menggunakan perintah `CREATE EXTENSION`. Dengan kata lain, Anda masih perlu membuat ekstensi agar fungsinya tersedia bagi pengguna basis data.

3. Untuk membuat ekstensi, gunakan perintah `CREATE EXTENSION` seperti yang Anda lakukan untuk ekstensi lainnya. Seperti ekstensi lainnya, pengguna basis data harus memiliki izin `CREATE` dalam basis data.

```
CREATE EXTENSION pg_distance;
```

4. Untuk menguji `pg_distance` TLE ekstensi, Anda dapat menggunakannya untuk menghitung jarak Manhattan antara empat titik.

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

Untuk menghitung Jarak Euclidean antara kumpulan titik yang sama, Anda dapat menggunakan berikut.

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

Ekstensi `pg_distance` memuat fungsi dalam basis data dan membuatnya tersedia bagi setiap pengguna dengan izin pada basis data.

## Memodifikasi ekstensi Anda TLE

Untuk meningkatkan kinerja kueri untuk fungsi yang dikemas dalam TLE ekstensi ini, tambahkan dua SQL atribut Postgre berikut ke spesifikasinya.

- `IMMUTABLE` – Atribut `IMMUTABLE` memastikan bahwa pengoptimal kueri dapat menggunakan pengoptimalan untuk meningkatkan waktu respons kueri. Untuk informasi selengkapnya, lihat Kategori Volatilitas Fungsi dalam dokumentasi SQL Postgre.
- `PARALLEL SAFE`— `PARALLEL SAFE` Atribut adalah atribut lain yang memungkinkan Postgre SQL menjalankan fungsi dalam mode paralel. Untuk informasi lebih lanjut, lihat CREATEFUNCTIONdi dokumentasi PostgreSQL.

Dalam contoh berikut, Anda dapat melihat bagaimana `pgtle.install_update_path` fungsi digunakan untuk menambahkan atribut ini ke setiap fungsi untuk membuat `0.2` versi `pg_distance` TLE ekstensi. Untuk informasi selengkapnya tentang fungsi ini, lihat pgtle.install_update_path. Anda harus memiliki peran `pgtle_admin` untuk melakukan tugas ini.

Untuk memperbarui TLE ekstensi yang ada dan menentukan versi default

1. Connect ke instance penulis cluster Aurora Postgre SQL DB Anda RDS lain, seperti. pgAdmin

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Ubah TLE ekstensi yang ada dengan menyalin kode berikut dan menempelkannya ke konsol `psql` sesi Anda.

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

Anda akan melihat hasil yang mirip dengan berikut ini.

```
install_update_path
```

```
---------------------
 t
(1 row)
```

Anda dapat menjadikan versi ekstensi ini sebagai versi default, sehingga pengguna basis data tidak perlu menentukan versi saat mereka membuat atau memperbarui ekstensi di basis data mereka.

3. Untuk menentukan bahwa versi modifikasi (versi 0.2) TLE ekstensi Anda adalah versi default, gunakan `pgtle.set_default_version` fungsi seperti yang ditunjukkan pada contoh berikut.

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

Untuk informasi selengkapnya tentang fungsi ini, lihat pgtle.set_default_version.

4. Dengan kode di tempat, Anda dapat memperbarui TLE ekstensi yang diinstal dengan cara biasa, dengan menggunakan `ALTER EXTENSION ... UPDATE` perintah, seperti yang ditunjukkan di sini:

```
ALTER EXTENSION pg_distance UPDATE;
```

## Menjatuhkan TLE ekstensi Anda dari database

Anda dapat menjatuhkan TLE ekstensi Anda dengan menggunakan `DROP EXTENSION` perintah dengan cara yang sama seperti yang Anda lakukan untuk ekstensi Postgre SQL lainnya. Menghapus ekstensi tidak menghapus file penginstalan yang membentuk ekstensi, yang memungkinkan pengguna membuat ulang ekstensi. Untuk menghapus ekstensi dan file penginstalannya, lakukan proses dua langkah berikut.

Untuk menjatuhkan TLE ekstensi dan menghapus file instalasinya

1. Gunakan `psql` atau alat klien lain untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. Jatuhkan ekstensi seperti yang Anda lakukan pada ekstensi Postgre SQL apa pun.

```
DROP EXTENSION your-TLE-extension
```

Misalnya, jika Anda membuat ekstensi `pg_distance` seperti yang dijelaskan dalamContoh: Membuat ekstensi bahasa tepercaya menggunakan SQL, Anda dapat menghilangkan ekstensi sebagai berikut.

```
DROP EXTENSION pg_distance;
```

Anda melihat output yang mengonfirmasi bahwa ekstensi telah dihilangkan, sebagai berikut.

```
DROP EXTENSION
```

Pada titik ini, ekstensi tidak lagi aktif dalam basis data. Namun, file penginstalan dan file kontrolnya masih tersedia di basis data, sehingga pengguna basis data dapat membuat ekstensi lagi jika ingin.

- Jika Anda ingin membiarkan file ekstensi utuh sehingga pengguna database dapat membuat TLE ekstensi Anda, Anda dapat berhenti di sini.
- Jika Anda ingin menghapus semua file yang membentuk ekstensi, lanjutkan ke langkah berikutnya.

3. Untuk menghapus semua file penginstalan untuk ekstensi Anda, gunakan fungsi `pgtle.uninstall_extension`. Fungsi ini menghapus semua kode dan file kontrol untuk ekstensi Anda.

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

Misalnya, untuk menghapus semua file penginstalan `pg_distance`, gunakan perintah berikut.

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

# Menghapus Instalasi Ekstensi Bahasa Tepercaya untuk Postgre SQL

Jika Anda tidak lagi ingin membuat TLE ekstensi Anda sendiri menggunakanTLE, Anda dapat menjatuhkan `pg_tle` ekstensi dan menghapus semua artefak. Tindakan ini termasuk menjatuhkan TLE ekstensi apa pun dalam database dan menjatuhkan skema. `pgtle`

Untuk menghilangkan ekstensi **`pg_tle`** dan skema dari basis data

1. Gunakan `psql` atau alat klien lain untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda.

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password --dbname=dbname
   ```

2. Hilangkan ekstensi `pg_tle` dari basis data. Jika database memiliki TLE ekstensi Anda sendiri yang masih berjalan di database, Anda juga harus menghapus ekstensi tersebut. Untuk melakukannya, Anda dapat menggunakan kata kunci `CASCADE`, seperti yang ditunjukkan berikut ini.

   ```
   DROP EXTENSION pg_tle CASCADE;
   ```

   Jika ekstensi `pg_tle` masih tidak aktif dalam basis data, Anda tidak perlu menggunakan kata kunci `CASCADE`.

3. Hilangkan skema `pgtle`. Tindakan ini akan menghapus semua fungsi manajemen dari basis data.

   ```
   DROP SCHEMA pgtle CASCADE;
   ```

   Perintah ini menampilkan hasil berikut setelah proses selesai.

   ```
   DROP SCHEMA
   ```

   Ekstensi `pg_tle`, skema dan fungsinya, serta semua artefak dihapus. Untuk membuat ekstensi baru menggunakanTLE, melalui proses pengaturan lagi. Untuk informasi selengkapnya, lihat .

# Menggunakan hook PostgreSQL dengan ekstensi TLE

Hook adalah mekanisme callback yang tersedia di PostgreSQL yang memungkinkan developer memanggil fungsi kustom atau rutinitas lainnya selama operasi basis data reguler. Kit pengembangan TLE mendukung hook PostgreSQL sehingga Anda dapat mengintegrasikan fungsi kustom dengan perilaku PostgreSQL saat runtime. Misalnya, Anda dapat menggunakan hook untuk mengaitkan proses autentikasi dengan kode kustom Anda sendiri, atau mengubah proses perencanaan dan eksekusi kueri untuk kebutuhan spesifik Anda.

Ekstensi TLE Anda dapat menggunakan hook. Jika hook cakupannya global, ini berlaku di semua basis data. Oleh karena itu, jika ekstensi TLE Anda menggunakan hook global, Anda perlu membuat ekstensi TLE di semua basis data yang dapat diakses pengguna.

Saat menggunakan ekstensi `pg_tle` untuk membuat Ekstensi Bahasa Tepercaya Anda sendiri, Anda dapat menggunakan hook yang tersedia dari SQL API untuk membangun fungsi ekstensi. Anda harus mendaftarkan hook apa pun di `pg_tle`. Untuk beberapa hook, Anda mungkin juga perlu mengatur berbagai parameter konfigurasi. Misalnya, hook pemeriksaan `passcode` dapat diatur ke aktif, nonaktif, wajib. Untuk informasi selengkapnya tentang persyaratan khusus untuk hook `pg_tle` yang tersedia, lihat Referensi hook untuk Trusted Language Extensions for PostgreSQL.

## Contoh: Membuat ekstensi yang menggunakan hook PostgreSQL

Contoh yang dibahas di bagian ini menggunakan hook PostgreSQL untuk memeriksa kata sandi yang diberikan selama operasi SQL tertentu dan mencegah pengguna basis data mengatur kata sandi mereka ke salah satu basis data yang tercantum dalam tabel `password_check.bad_passwords`. Tabel berisi sepuluh besar pilihan kata sandi yang paling umum digunakan, tetapi mudah dipecahkan.

Untuk menyiapkan contoh ini di klaster DB Aurora PostgreSQL, Anda harus sudah menginstal Ekstensi Bahasa Tepercaya. Untuk detailnya, lihat .

Untuk menyiapkan contoh hook pemeriksaan kata sandi

1. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Salin kode dari Daftar kode hook pemeriksaan kata sandi dan tempel ke basis data Anda.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
```

```
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Setelah ekstensi dimuat ke basis data, Anda akan melihat output seperti berikut.

```
 install_extension
-------------------
 t
(1 row)
```

3. Jika masih terhubung ke basis data, Anda kini dapat membuat ekstensi.

```
CREATE EXTENSION my_password_check_rules;
```

4. Anda dapat mengonfirmasi bahwa ekstensi telah dibuat dalam basis data dengan menggunakan metacommand `psql` berikut.

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
-------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. Buka sesi terminal lain untuk bekerja dengan AWS CLI. Anda perlu mengubah grup parameter DB kustom untuk mengaktifkan hook pemeriksaan kata sandi. Untuk melakukannya, gunakan perintah modify-db-parameter-groupCLI seperti yang ditunjukkan pada contoh berikut.

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

   Mungkin diperlukan waktu beberapa menit agar perubahan pada pengaturan grup parameter diterapkan. Akan tetapi, parameter ini bersifat dinamis, jadi Anda tidak perlu memulai ulang instans penulis klaster DB Aurora PostgreSQL untuk menerapkan pengaturan.

6. Buka sesi `psql` dan kirim kueri ke basis data untuk memverifikasi bahwa hook password_check telah diaktifkan.

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

Hook password_check kini aktif. Anda dapat mengujinya dengan membuat peran baru dan menggunakan salah satu kata sandi yang buruk, seperti pada contoh berikut.

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

Output ini telah diformat agar mudah dibaca.

Contoh berikut menunjukkan bahwa perilaku `\password` metacommand interaktif `pgsql` juga dipengaruhi oleh hook password_check.

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

Anda dapat menghilangkan ekstensi TLE ini dan menghapus instalan file sumbernya jika ingin. Untuk informasi selengkapnya, lihat Menjatuhkan TLE ekstensi Anda dari database.

Daftar kode hook pemeriksaan kata sandi

Kode contoh yang ditampilkan di sini menentukan spesifikasi untuk ekstensi TLE `my_password_check_rules`. Jika kode ini disalin dan ditempelkan ke basis data, kode untuk ekstensi `my_password_check_rules` akan dimuat ke dalam basis data, dan hook `password_check` akan didaftarkan untuk digunakan oleh ekstensi.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
```

```
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

# Referensi fungsi untuk Ekstensi Bahasa Tepercaya untuk Postgre SQL

Lihat dokumentasi referensi berikut tentang fungsi yang tersedia di Ekstensi Bahasa Tepercaya untuk PostgreSQL. Gunakan fungsi-fungsi ini untuk menginstal, mendaftar, memperbarui, dan mengelola TLEekstensi Anda, yaitu SQL ekstensi Postgre yang Anda kembangkan menggunakan kit pengembangan Ekstensi Bahasa Tepercaya.

Fungsi


- pgtle.available_extensions
- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(nama)
- pgtle.uninstall_extension (nama, versi)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists


## pgtle.available_extensions

Fungsi `pgtle.available_extensions` adalah fungsi pengembalian set. Ia mengembalikan semua TLE ekstensi yang tersedia dalam database. Setiap baris yang dikembalikan berisi informasi tentang TLE ekstensi tunggal.

Prototipe fungsi

```
pgtle.available_extensions()
```

Peran

Tidak ada.

Argumen

Tidak ada.

Output

- `name`— Nama TLE ekstensi.
- `default_version`— Versi TLE ekstensi yang akan digunakan saat `CREATE EXTENSION` dipanggil tanpa versi yang ditentukan.
- `description`— Penjelasan yang lebih rinci tentang TLE ekstensi.

Contoh penggunaan

```
SELECT * FROM pgtle.available_extensions();
```

## pgtle.available_extension_versions

Fungsi `available_extension_versions` adalah fungsi pengembalian set. Ini mengembalikan daftar semua TLE ekstensi yang tersedia dan versinya. Setiap baris berisi informasi tentang versi tertentu dari TLE ekstensi yang diberikan, termasuk apakah itu memerlukan peran tertentu.

Prototipe fungsi

```
pgtle.available_extension_versions()
```

Peran

Tidak ada.

Argumen

Tidak ada.

Output

- `name`— Nama TLE ekstensi.
- `version`— Versi TLE ekstensi.

- `superuser`— Nilai ini selalu `false` untuk TLE ekstensi Anda. Izin yang diperlukan untuk membuat TLE ekstensi atau memperbaruinya sama dengan membuat objek lain dalam database yang diberikan.
- `trusted`— Nilai ini selalu `false` untuk TLE ekstensi.
- `relocatable`— Nilai ini selalu `false` untuk TLE ekstensi.
- `schema`— Menentukan nama skema di mana TLE ekstensi diinstal.
- `requires`— Array yang berisi nama-nama ekstensi lain yang dibutuhkan oleh TLE ekstensi ini.
- `description`— Penjelasan rinci tentang TLE ekstensi.

Untuk informasi selengkapnya tentang nilai keluaran, lihat Mengemas Objek Terkait ke dalam Ekstensi > File Ekstensi dalam dokumentasi PostgreSQL.

Contoh penggunaan

```
SELECT * FROM pgtle.available_extension_versions();
```

## pgtle.extension_update_paths

Fungsi `extension_update_paths` adalah fungsi pengembalian set. Ini mengembalikan daftar semua jalur pembaruan yang mungkin untuk TLE ekstensi. Setiap baris menyertakan peningkatan atau penurunan versi yang tersedia untuk ekstensi itu. TLE

Prototipe fungsi

```
pgtle.extension_update_paths(name)
```

Peran

Tidak ada.

Argumen

`name`— Nama TLE ekstensi dari mana untuk mendapatkan jalur upgrade.

Output

- `source` – Versi sumber untuk pembaruan.
- `target` – Versi target untuk pembaruan.

- `path`— Jalur pemutakhiran yang digunakan untuk memperbarui TLE ekstensi dari `source` `target` versi ke versi, misalnya,`0.1--0.2`.

Contoh penggunaan

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

## pgtle.install_extension

`install_extension`Fungsi ini memungkinkan Anda menginstal artefak yang membentuk TLE ekstensi Anda di database, setelah itu dapat dibuat menggunakan `CREATE EXTENSION` perintah.

Prototipe fungsi

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

Peran

Tidak ada.

Argumen

- `name`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `version`— Versi TLE ekstensi.
- `description`— Penjelasan rinci tentang TLE ekstensi. Deskripsi ini ditampilkan di kolom `comment` pada `pgtle.available_extensions()`.
- `ext`— Isi TLE ekstensi. Nilai ini berisi objek seperti fungsi.
- `requires`— Parameter opsional yang menentukan dependensi untuk ekstensi ini. TLE Ekstensi `pg_tle` secara otomatis ditambahkan sebagai dependensi.

Banyak dari argumen ini sama dengan yang disertakan dalam file kontrol ekstensi untuk menginstal ekstensi Postgre SQL pada sistem file instance SQL Postgre. Untuk informasi selengkapnya, lihat File Ekstensi dalam Kemasan Objek Terkait ke Ekstensi dalam dokumentasi PostgreSQL.

Output

Fungsi ini akan mengembalikan `OK` jika berhasil, dan `NULL` jika terjadi kesalahan.

- `OK`— TLE Ekstensi telah berhasil dipasang di database.
- `NULL`— TLE Ekstensi belum berhasil dipasang di database.

Contoh penggunaan

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

## pgtle.install_update_path

`install_update_path`Fungsi ini menyediakan jalur pembaruan antara dua versi TLE ekstensi yang berbeda. Fungsi ini memungkinkan pengguna TLE ekstensi Anda untuk memperbarui versinya dengan menggunakan `ALTER EXTENSION ... UPDATE` sintaks.

Prototipe fungsi

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

Peran

`pgtle_admin`

Argumen

- `name`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `fromvers`— Versi sumber TLE ekstensi untuk peningkatan.
- `tovers`— Versi tujuan TLE ekstensi untuk peningkatan.
- `ext` – Konten pembaruan. Nilai ini berisi objek seperti fungsi.

Output

Tidak ada.

Contoh penggunaan

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

## pgtle.register_feature

`register_feature`Fungsi menambahkan SQL fitur Postgre internal yang ditentukan ke tabel. `pgtle.feature_info` Postgre SQL hook adalah contoh fitur Postgre internal. SQL Kit pengembangan Ekstensi Bahasa Tepercaya mendukung penggunaan kait PostgreSQL. Saat ini, fungsi ini mendukung fitur berikut.

- `passcheck`— Mendaftarkan hook pemeriksaan kata sandi dengan prosedur atau fungsi Anda yang menyesuaikan perilaku pemeriksaan kata sandi Postgre. SQL

Prototipe fungsi

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama prosedur atau fungsi yang disimpan untuk digunakan dengan fitur tersebut.
- `feature` – Nama fitur `pg_tle` (seperti `passcheck`) untuk mendaftar dengan fungsi.

Output

Tidak ada.

Contoh penggunaan

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

## pgtle.register_feature_if_not_exists

`pgtle.register_feature_if_not_exists`Fungsi menambahkan SQL fitur Postgre yang ditentukan ke `pgtle.feature_info` tabel dan mengidentifikasi TLE ekstensi atau prosedur atau fungsi lain yang menggunakan fitur tersebut. Untuk informasi selengkapnya tentang hook dan Trusted Language Extensions, lihat Menggunakan hook PostgreSQL dengan ekstensi TLE.

Prototipe fungsi

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

Peran

`pgtle_admin`

Argumen

- `proc`— Nama prosedur atau fungsi tersimpan yang berisi logika (kode) untuk digunakan sebagai fitur untuk TLE ekstensi Anda. Misalnya, kode `pw_hook`.
- `feature`— Nama SQL fitur Postgre untuk mendaftar untuk fungsi tersebut. TLE Saat ini, satu-satunya fitur yang tersedia adalah hook `passcheck`. Untuk informasi selengkapnya, lihat Hook pemeriksaan kata sandi (passcheck).

Output

Mengembalikan `true` setelah mendaftarkan fitur untuk ekstensi yang ditentukan. Mengembalikan `false` jika fitur sudah terdaftar.

Contoh penggunaan

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

## pgtle.set_default_version

`set_default_version`Fungsi ini memungkinkan Anda menentukan `default_version` untuk TLE ekstensi Anda. Anda dapat menggunakan fungsi ini untuk menentukan jalur pemutakhiran dan menetapkan versi sebagai default untuk TLE ekstensi Anda. Ketika pengguna database menentukan TLE ekstensi Anda dalam `ALTER EXTENSION ... UPDATE` perintah `CREATE EXTENSION` dan, versi TLE ekstensi Anda dibuat dalam database untuk pengguna tersebut.

Fungsi ini mengembalikan `true` jika berhasil. Jika TLE ekstensi yang ditentukan dalam `name` argumen tidak ada, fungsi mengembalikan kesalahan. Demikian pula, jika TLE ekstensi tidak ada, ia mengembalikan kesalahan. `version`

Prototipe fungsi

```
pgtle.set_default_version(name text, version text)
```

Peran

`pgtle_admin`

Argumen

- `name`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `version`— Versi TLE ekstensi untuk mengatur default.

Output

- `true` – Saat pengaturan versi default berhasil, fungsi mengembalikan `true`.
- `ERROR`— Mengembalikan pesan kesalahan jika TLE ekstensi dengan nama atau versi tertentu tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

## pgtle.uninstall_extension(nama)

`uninstall_extension`Fungsi ini menghapus semua versi TLE ekstensi dari database. Fungsi ini mencegah panggilan future `CREATE EXTENSION` dari menginstal TLE ekstensi. Jika TLE ekstensi tidak ada dalam database, kesalahan muncul.

`uninstall_extension`Fungsi tidak akan menjatuhkan TLE ekstensi yang saat ini aktif dalam database. Untuk menghapus TLE ekstensi yang saat ini aktif, Anda perlu memanggil secara eksplisit `DROP EXTENSION` untuk menghapusnya.

Prototipe fungsi

```
pgtle.uninstall_extension(extname text)
```

Peran

`pgtle_admin`

Argumen

- `extname`— Nama TLE ekstensi yang akan dihapus. Nama ini sama dengan yang digunakan `CREATE EXTENSION` untuk memuat TLE ekstensi untuk digunakan dalam database tertentu.

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

## pgtle.uninstall_extension (nama, versi)

`uninstall_extension(name, version)`Fungsi ini menghapus versi TLE ekstensi yang ditentukan dari database. Fungsi ini mencegah `CREATE EXTENSION` dan `ALTER EXTENSION` dari menginstal atau memperbarui TLE ekstensi ke versi yang ditentukan. Fungsi ini juga menghapus semua jalur pembaruan untuk versi TLE ekstensi yang ditentukan. Fungsi ini tidak akan menghapus instalasi TLE ekstensi jika saat ini aktif dalam database. Anda harus secara eksplisit memanggil

`DROP EXTENSION` untuk menghapus ekstensi. TLE Untuk menghapus semua versi TLE ekstensi, lihat[pgtle.uninstall_extension(nama)](#).

Prototipe fungsi

```
pgtle.uninstall_extension(extname text, version text)
```

Peran

`pgtle_admin`

Argumen

- `extname`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `version`— Versi TLE ekstensi untuk menghapus instalasi dari database.

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

## pgtle.uninstall_extension_if_exists

`uninstall_extension_if_exists`Fungsi ini menghapus semua versi TLE ekstensi dari database yang diberikan. Jika TLE ekstensi tidak ada, fungsi kembali diam-diam (tidak ada pesan kesalahan yang muncul). Jika ekstensi yang ditentukan sedang aktif dalam basis data, fungsi ini tidak akan menghapusnya. Anda harus secara eksplisit memanggil `DROP EXTENSION` untuk menghapus TLE ekstensi sebelum menggunakan fungsi ini untuk menghapus artefaknya.

Prototipe fungsi

```
pgtle.uninstall_extension_if_exists(extname text)
```

Peran

`pgtle_admin`

Argumen

- `extname`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.

Output

Fungsi `uninstall_extension_if_exists` mengembalikan `true` setelah menghapus instalasi ekstensi yang ditentukan. Jika ekstensi yang ditentukan tidak ada, fungsi akan mengembalikan `false`.

- `true`— Kembali `true` setelah mencopot pemasangan TLE ekstensi.
- `false`— Kembali `false` ketika TLE ekstensi tidak ada dalam database.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

## pgtle.uninstall_update_path

`uninstall_update_path`Fungsi menghapus jalur pembaruan tertentu dari TLE ekstensi. Fungsi ini membuat `ALTER EXTENSION ... UPDATE TO` tidak dapat menggunakan ekstensi ini sebagai jalur pembaruan.

Jika TLE ekstensi saat ini digunakan oleh salah satu versi di jalur pembaruan ini, ekstensi tetap ada di database.

Jika jalur pembaruan yang ditentukan tidak ada, fungsi akan memunculkan kesalahan.

Prototipe fungsi

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

Peran

`pgtle_admin`

Argumen

- `extname`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `fromvers`— Versi sumber TLE ekstensi yang digunakan pada jalur pembaruan.

- `tovers`— Versi tujuan TLE ekstensi yang digunakan pada jalur pembaruan.

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

## pgtle.uninstall_update_path_if_exists

`uninstall_update_path_if_exists`Fungsinya mirip `uninstall_update_path` dengan menghapus jalur pembaruan yang ditentukan dari TLE ekstensi. Namun, jika jalur pembaruan tidak ada, fungsi ini tidak memunculkan pesan kesalahan. Sebaliknya, fungsi mengembalikan `false`.

Prototipe fungsi

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

Peran

`pgtle_admin`

Argumen

- `extname`— Nama TLE ekstensi. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `fromvers`— Versi sumber TLE ekstensi yang digunakan pada jalur pembaruan.
- `tovers`— Versi tujuan TLE ekstensi yang digunakan pada jalur pembaruan.

Output

- `true`— Fungsi telah berhasil memperbarui jalur untuk TLE ekstensi.
- `false`— Fungsi tidak dapat memperbarui jalur untuk TLE ekstensi.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

## pgtle.unregister_feature

Fungsi `unregister_feature` menyediakan cara untuk menghapus fungsi yang terdaftar untuk menggunakan fitur `pg_tle`, seperti hook. Untuk informasi tentang pendaftaran fitur, lihat pgtle.register_feature.

Prototipe fungsi

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama fungsi tersimpan untuk mendaftar dengan fitur `pg_tle`.
- `feature` – Nama fitur `pg_tle` untuk mendaftar dengan fungsi. Misalnya, `passcheck` adalah fitur yang dapat didaftarkan untuk digunakan oleh ekstensi bahasa tepercaya yang Anda kembangkan. Untuk informasi selengkapnya, lihat Hook pemeriksaan kata sandi (passcheck).

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

## pgtle.unregister_feature_if_exists

Fungsi `unregister_feature` menyediakan cara untuk menghapus fungsi yang terdaftar untuk menggunakan fitur `pg_tle`, seperti hook. Untuk informasi selengkapnya, lihat Menggunakan hook PostgreSQL dengan ekstensi TLE. Mengembalikan `true` setelah berhasil membatalkan pendaftaran fitur. Mengembalikan `false` jika fitur tidak terdaftar.

Untuk informasi tentang mendaftarkan `pg_tle` fitur untuk TLE ekstensi Anda, lihatpgtle.register_feature.

Prototipe fungsi

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama fungsi tersimpan yang terdaftar untuk menyertakan fitur `pg_tle`.
- `feature` – Nama fitur `pg_tle` yang terdaftar dengan ekstensi bahasa tepercaya.

Output

Mengembalikan `true` atau `false`, sebagai berikut.

- `true` – Fungsi telah berhasil membatalkan pendaftaran fitur dari ekstensi.
- `false`— Fungsi tidak dapat membatalkan pendaftaran fitur dari TLE ekstensi.

Contoh penggunaan

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

## Referensi hook untuk Trusted Language Extensions for PostgreSQL

Trusted Language Extensions for PostgreSQL mendukung hook PostgreSQL. Hook adalah mekanisme panggilan balik internal yang tersedia bagi developer untuk memperluas fungsionalitas inti PostgreSQL. Dengan hook, developer dapat mengimplementasikan fungsi atau prosedurnya sendiri untuk digunakan dalam berbagai operasi basis data, sehingga mengubah perilaku PostgreSQL dalam beberapa cara. Misalnya, Anda dapat menggunakan hook `passcheck` untuk menyesuaikan cara PostgreSQL menangani kata sandi yang diberikan saat membuat atau mengubah kata sandi untuk pengguna (peran).

Lihat dokumentasi berikut untuk mempelajari kait passcheck yang tersedia untuk ekstensi TLE Anda. Untuk mempelajari lebih lanjut tentang kait yang tersedia termasuk hook autentikasi klien, lihat kait Ekstensi Bahasa Tepercaya.

## Hook pemeriksaan kata sandi (passcheck)

Hook `passcheck` digunakan untuk menyesuaikan perilaku PostgreSQL selama proses pemeriksaan kata sandi untuk perintah SQL dan metacommand `psql` berikut.

- `CREATE ROLE` *`username`* `...PASSWORD` – Untuk informasi selengkapnya, lihat CREATE ROLE dalam dokumentasi PostgreSQL.
- `ALTER ROLE` *`username`*`...PASSWORD` – Untuk informasi selengkapnya, lihat ALTER ROLE dalam dokumentasi PostgreSQL.
- `\password` *`username`* – Metacommand `psql` interaktif ini secara aman mengubah kata sandi untuk pengguna yang ditentukan dengan hashing kata sandi sebelum menggunakan sintaks `ALTER ROLE ... PASSWORD` secara transparan. Metacommand adalah pembungkus aman untuk perintah `ALTER ROLE ... PASSWORD`, sehingga hook diterapkan untuk perilaku metacommand `psql`.

Untuk contohnya, lihat Daftar kode hook pemeriksaan kata sandi.

Daftar Isi


- Prototipe fungsi
- Argumen
- Konfigurasi
- Catatan penggunaan


### Prototipe fungsi

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

### Argumen

Fungsi hook `passcheck` memiliki argumen berikut.

- `username` – Nama (sebagai teks) dari peran (nama pengguna) yang mengatur kata sandi.
- `password` – Kata sandi yang di-hash atau teks biasa. Kata sandi yang dimasukkan harus sesuai dengan jenis yang ditentukan dalam `password_type`.

- `password_type` – Menentukan format `pgtle.password_type` kata sandi. Format ini dapat berupa salah satu opsi berikut.
  - `PASSWORD_TYPE_PLAINTEXT` – Kata sandi teks biasa.
  - `PASSWORD_TYPE_MD5`— Kata sandi yang telah di-hash menggunakan algoritma MD5 (message digest 5).
  - `PASSWORD_TYPE_SCRAM_SHA_256` – Kata sandi yang telah di-hash menggunakan algoritma SCRAM-SHA-256.
- `valid_until` – Menentukan waktu ketika kata sandi menjadi tidak valid. Argumen ini opsional. Jika Anda menggunakan argumen ini, tentukan waktu sebagai nilai `timestamptz`.
- `valid_null` – Jika Boolean ini diatur ke `true`, opsi `valid_until` diatur ke `NULL`.

### Konfigurasi

Fungsi `pgtle.enable_password_check` mengontrol apakah hook passcheck aktif. Hook passcheck memiliki tiga pengaturan yang memungkinkan.

- `off` – Menonaktifkan hook pemeriksaan kata sandi `passcheck`. Ini adalah nilai default.
- on – Mengaktifkan hook pemeriksaan kata sandi `passcode` sehingga kata sandi diperiksa di tabel.
- `require` – Memerlukan hook pemeriksaan kata sandi untuk didefinisikan.

### Catatan penggunaan

Untuk mengaktifkan atau menonaktifkan hook `passcheck`, Anda perlu memodifikasi grup parameter DB kustom untuk instans penulis klaster DB Aurora PostgreSQL Anda.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
```

```
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

# Referensi Amazon Aurora PostgreSQL

Dalam topik berikut, Anda dapat menemukan informasi tentang susunan, fungsi, parameter, dan acara tunggu di Amazon Aurora PostgreSQL.

Topik


- Kolasi Aurora PostgreSQL untuk EBCDIC dan migrasi mainframe lainnya
-
- Referensi fungsi Aurora PostgreSQL
- Parameter Amazon Aurora PostgreSQL
- Peristiwa tunggu Amazon Aurora PostgreSQL


## Kolasi Aurora PostgreSQL untuk EBCDIC dan migrasi mainframe lainnya

Migrasi aplikasi mainframe ke platform baru seperti AWS idealnya mempertahankan perilaku aplikasi. Untuk mempertahankan perilaku aplikasi pada platform baru persis seperti pada mainframe mengharuskan data yang dimigrasi dikolasi menggunakan aturan kolasi dan pengurutan yang sama. Misalnya, banyak solusi migrasi Db2 menggeser nilai nol ke u0180 (posisi Unicode 0180), jadi kolasi ini mengurutkan u0180 terlebih dahulu. Ini adalah salah satu contoh bagaimana kolasi dapat bervariasi dari sumber mainframe-nya dan mengapa perlu memilih kolasi yang memetakan lebih baik ke kolasi EBCDIC asli.

Aurora PostgreSQL 14.3 dan versi yang lebih tinggi menyediakan banyak ICU dan EBCDIC untuk mendukung migrasi tersebut untuk menggunakan layanan Modernisasi Mainframe. AWS AWS Untuk mempelajari lebih lanjut tentang layanan ini, lihat Apa itu Modernisasi AWS Mainframe?

Pada tabel berikut, Anda dapat menemukan kolasi yang disediakan Aurora PostgreSQL. Penyusunan ini mengikuti aturan EBCDIC dan memastikan bahwa aplikasi mainframe berfungsi sama AWS seperti yang mereka lakukan di lingkungan mainframe. Nama kolasi mencakup halaman kode yang relevan, (cpnnnn), sehingga Anda dapat memilih kolasi yang sesuai untuk sumber mainframe Anda. Misalnya, gunakan `en-US-cp037-x-icu` untuk mencapai perilaku kolasi data EBCDIC yang berasal dari aplikasi mainframe yang menggunakan kode halaman 037.

| Kolasi EBCDIC | AWS Kolasi Blu Age | AWS Kolasi Fokus Mikro |
|---|---|---|
| da-DK-cp1142-x-icu | DA-DK-CP1142 b-x-icu | DA-DK-CP1142 m-x-icu |

| Kolasi EBCDIC | AWS Kolasi Blu Age | AWS Kolasi Fokus Mikro |
|---|---|---|
| da-DK-cp277-x-icu | DA-DK-CP277 b-x-icu | – |
| de-DE-cp1141-x-icu | De-De-CP1141 b-x-icu | De-De-CP1141 m-x-icu |
| de-DE-cp273-x-icu | De-de-CP273 b-x-icu | – |
| en-GB-cp1146-x-icu | EN-GB-CP1146 b-x-icu | EN-GB-CP1146 m-x-icu |
| en-GB-cp285-x-icu | EN-GB-CP285 b-x-icu | – |
| en-US-cp037-x-icu | id-US-CP037 b-x-icu | – |
| en-US-cp1140-x-icu | en-AS-CP1140 b-x-icu | en-AS-CP1140 m-x-icu |
| es-ES-cp1145-x-icu | ES-ES-CP1145 b-x-icu | ES-ES-CP1145 m-x-icu |
| es-ES-cp284-x-icu | es-ES-CP284 b-x-icu | – |
| fi-FI-cp1143-x-icu | fi-fi-CP1143 b-x-icu | fi-fi-CP1143 m-x-icu |
| fi-FI-cp278-x-icu | fi-fi-CP278 b-x-icu | – |
| fr-FR-cp1147-x-icu | fr-FR-CP1147 b-x-icu | fr-FR-CP1147 m-x-icu |
| fr-FR-cp297-x-icu | FR-FR-CP297 b-x-icu | – |
| it-IT-cp1144-x-icu | IT-IT-CP1144 b-x-icu | IT-IT-CP1144 m-x-icu |
| it-IT-cp280-x-icu | IT-IT-CP280 b-x-icu | – |
| nl-BE-cp1148-x-icu | nl-BE-CP1148 b-x-icu | nl-BE-CP1148 m-x-icu |
| nl-BE-cp500-x-icu | NL-BE-CP500 b-x-icu | – |

Untuk mempelajari lebih lanjut tentang AWS Blu Age, lihat Tutorial: Managed Runtime for AWS Blu Age di Panduan Pengguna Modernisasi AWS Mainframe.

Untuk informasi selengkapnya tentang bekerja dengan AWS Micro Focus, lihat Tutorial: Managed Runtime for Micro Focus di Panduan Pengguna Modernisasi AWS Mainframe.

Untuk mempelajari selengkapnya tentang mengelola kolasi PostgreSQL, lihat Collation Support dalam dokumentasi PostgreSQL.

Kolasi adalah serangkaian aturan yang menentukan cara pengurutan dan perbandingan string karakter yang disimpan di basis data. Kolasi memiliki peran mendasar dalam sistem komputer dan dimasukkan sebagai bagian dari sistem operasi. Kolasi berubah dari waktu ke waktu ketika karakter baru ditambahkan ke bahasa atau ketika aturan urutan berubah.

Pustaka kolasi menentukan aturan dan algoritma khusus untuk kolasi. Pustaka pengumpulan yang paling populer digunakan dalam Postgre SQL adalah GNU C (glibc) dan komponen Internasionalisasi untuk Unicode (). ICU Secara default, Aurora Postgre SQL menggunakan pemeriksaan glibc yang menyertakan urutan pengurutan karakter unicode untuk urutan karakter multi-byte.

Saat Anda membuat , ia memeriksa sistem operasi untuk pemeriksaan yang tersedia. SQLParameter Postgre dari `CREATE DATABASE` perintah `LC_COLLATE` dan `LC_CTYPE` digunakan untuk menentukan pemeriksaan, yang berdiri sebagai pemeriksaan default dalam database itu. Atau, Anda juga dapat menggunakan parameter `LOCALE` di `CREATE DATABASE` untuk menetapkan parameter ini. Parameter ini menentukan kolasi default untuk string karakter dalam basis data dan aturan untuk mengklasifikasikan karakter sebagai huruf, angka, atau simbol. Anda juga dapat memilih kolasi untuk digunakan pada kolom, indeks, atau kueri.

Aurora Postgre SQL bergantung pada pustaka glibc dalam sistem operasi untuk dukungan pemeriksaan. Aurora Postgre SQL RDS diperbarui secara berkala dengan versi terbaru dari sistem operasi. Pembaruan ini terkadang menyertakan versi pustaka glibc yang lebih baru. Jarang sekali versi glibc yang lebih baru mengubah tata urutan atau kolasi beberapa karakter, yang dapat menyebabkan data diurutkan secara berbeda atau menghasilkan entri indeks yang tidak valid. Jika terjadi masalah terkait tata urutan kolasi selama pembaruan, Anda mungkin perlu membuat ulang indeks.

Untuk mengurangi kemungkinan dampak pembaruan glibc, Aurora Postgre SQL RDS untuk Postgre independen. Pustaka ini kompatibel dengan glibc 2.26-59.amzn2, dan menyediakan stabilitas tata urutan untuk mencegah kesalahan hasil kueri.

## Referensi fungsi Aurora PostgreSQL

Berikut ini, Anda dapat menemukan daftar fungsi Aurora PostgreSQL yang tersedia untuk klaster DB Aurora Anda yang menjalankan mesin DB Aurora Edisi Kompatibel PostgreSQL. Fungsi Aurora PostgreSQL ini merupakan tambahan dari fungsi PostgreSQL standar. Untuk informasi selengkapnya tentang fungsi PostgreSQL standar, lihat PostgreSQL–Fungsi dan Operator.

## Gambaran Umum

Anda dapat menggunakan fungsi berikut untuk instans DB Amazon RDS yang menjalankan Aurora PostgreSQL:

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status
- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statements
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

## aurora_db_instance_identifier

Melaporkan nama instans DB yang terhubung dengan Anda.

Sintaksis

```
aurora_db_instance_identifier()
```

Argumen

Tidak ada

Jenis pengembalian

String VARCHAR

Catatan penggunaan

Fungsi ini menampilkan nama instans DB klaster Aurora Edisi Kompatibel PostgreSQL untuk klien basis data atau koneksi aplikasi Anda.

Fungsi ini tersedia dimulai dengan rilis Aurora PostgreSQL versi 13.7, 12.11, 11.16, dan 10.21 serta untuk semua versi lain yang lebih baru.

Contoh

Contoh berikut menampilkan hasil pemanggilan fungsi `aurora_db_instance_identifier`.

```
=> SELECT aurora_db_instance_identifier();
aurora_db_instance_identifier
------------------------------
 test-my-instance-name
```

Anda dapat menggabungkan hasil fungsi ini dengan fungsi `aurora_replica_status` untuk mendapatkan detail tentang instans DB untuk koneksi Anda. aurora_replica_status sendiri tidak memberikan tampilan instans DB mana yang Anda gunakan. Contoh kode berikut ini menunjukkan caranya.

```
=> SELECT *
```

```
    FROM aurora_replica_status() rt,
         aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]----------------------+----------------------
server_id                          | test-my-instance-name
session_id                         | MASTER_SESSION_ID
durable_lsn                        | 88492069
highest_lsn_rcvd                   |
current_read_lsn                   |
cur_replay_latency_in_usec         |
active_txns                        |
is_current                         | t
last_transport_error               | 0
last_error_timestamp               |
last_update_timestamp              | 2022-06-03 11:18:25+00
feedback_xmin                      |
feedback_epoch                     |
replica_lag_in_msec                |
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number         | 0
oldest_read_view_trx_id            |
oldest_read_view_lsn               |
pending_read_ios                   | 819
```

## aurora_ccm_status

Menampilkan status pengelola cache klaster.

Sintaksis

```
aurora_ccm_status()
```

Argumen

Tidak ada.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `buffers_sent_last_minute` – Jumlah buffer yang dikirim ke pembaca yang ditunjuk dalam beberapa menit terakhir.

- `buffers_found_last_minute` – Jumlah buffer yang sering diakses diidentifikasi selama beberapa menit terakhir.
- `buffers_sent_last_scan` – Jumlah buffer yang dikirim ke pembaca yang ditunjuk selama pemindaian lengkap cache buffer.
- `buffers_found_last_scan` – Jumlah buffer yang sering diakses yang dikirim selama pemindaian lengkap cache buffer. Buffer yang sudah di-cache di pembaca yang ditunjuk tidak akan dikirimkan.
- `buffers_sent_current_scan` – Jumlah buffer yang dikirim selama pemindaian saat ini.
- `buffers_found_current_scan` – Jumlah buffer yang sering diakses yang diidentifikasi dalam pemindaian saat ini.
- `current_scan_progress` – Jumlah buffer yang dikunjungi sejauh ini selama pemindaian saat ini.

Catatan penggunaan

Anda dapat menggunakan fungsi ini untuk memeriksa dan memantau fitur manajemen cache klaster (CCM). Fungsi ini hanya berfungsi jika CCM aktif di klaster DB Aurora PostgreSQL Anda. Untuk menggunakan fungsi ini, Anda menghubungkan ke instans Write DB di klaster Aurora PostgreSQL Anda.

Anda mengaktifkan CCM untuk klaster DB Aurora PostgreSQL dengan mengatur `apg_ccm_enabled` ke 1 di grup parameter klaster DB kustom klaster. Untuk mempelajari caranya, lihat Mengonfigurasi manajemen cache klaster.

Manajemen cache klaster aktif pada klaster DB Aurora PostgreSQL ketika klaster memiliki instans Pembaca Aurora yang dikonfigurasi sebagai berikut:

- Instans Pembaca Aurora menggunakan jenis dan ukuran kelas instans DB yang sama dengan instans Penulis klaster.
- Instans Pembaca Aurora dikonfigurasi sebagai Tier-0 untuk klaster. Jika klaster memiliki lebih dari satu Pembaca, ini adalah satu-satunya Pembaca Tier-0.

Mengatur lebih dari satu Pembaca ke Tier-0 menonaktifkan CCM. Ketika CCM dinonaktifkan, memanggil fungsi ini mengembalikan pesan kesalahan berikut:

```
ERROR: Cluster Cache Manager is disabled
```

Anda juga dapat menggunakan ekstensi pg_buffercache PostgreSQL untuk menganalisis cache buffer. Untuk informasi selengkapnya, lihat pg_buffercache di dokumentasi PostgreSQL.

Untuk informasi selengkapnya, lihat Pengantar manajemen cache klaster Aurora PostgreSQL.

Contoh

Contoh berikut menampilkan hasil pemanggilan fungsi `aurora_ccm_status`. Contoh pertama ini menunjukkan statistik CCM.

```
=> SELECT * FROM aurora_ccm_status();
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+------------------------
+-------------------------+---------------------------+----------------------------
+------------------------
                   2242000 |                   2242003 |               17920442 |
         17923410 |                  14098000 |                  14100964 |
   15877443
```

Untuk detail yang lebih lengkap, Anda dapat menggunakan tampilan yang diperluas, seperti yang ditunjukkan berikut:

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]----------------------+---------
buffers_sent_last_minute          | 2242000
buffers_found_last_minute         | 2242003
buffers_sent_last_scan            | 17920442
buffers_found_last_scan           | 17923410
buffers_sent_current_scan         | 14098000
buffers_found_current_scan        | 14100964
current_scan_progress             | 15877443
```

Contoh ini menunjukkan cara memeriksa tingkat hangat dan persentase hangat.

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
```

```
----------------+--------------
 16523 |        100.0
```

## aurora_global_db_instance_status

Menampilkan status semua instans Aurora, termasuk replika dalam klaster DB global Aurora.

Sintaksis

```
aurora_global_db_instance_status()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `server_id` – Pengidentifikasi instans DB.
- `session_id` – Pengidentifikasi unik untuk sesi saat ini. Nilai `MASTER_SESSION_ID` mengidentifikasi instans DB Penulis (primer).
- `aws_region`— Wilayah AWS Di mana instans DB global ini berjalan. Untuk daftar Wilayah, lihat Ketersediaan wilayah.
- `durable_lsn` – Nomor urutan log (LSN) dibuat durabel di penyimpanan. Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNs dipesan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi selanjutnya.
- `highest_lsn_rcvd` – LSN tertinggi yang diterima oleh instans DB dari instans DB penulis.
- `feedback_epoch` – Masa yang digunakan instans DB saat menghasilkan informasi hot standby. Hot standby adalah instans DB yang mendukung koneksi dan kueri saat DB primer berada dalam mode pemulihan atau mode siaga. Informasi hot standby mencakup masa (titik waktu) dan detail lain tentang instans DB yang digunakan sebagai hot standby. Untuk informasi selengkapnya, lihat Hot standby di dokumentasi PostgreSQL.
- `feedback_xmin` – ID transaksi aktif minimum (terlawas) yang digunakan oleh instans DB.
- `oldest_read_view_lsn` – LSN terlawas yang digunakan oleh instans DB untuk membaca dari penyimpanan.

- `visibility_lag_in_msec` – Seberapa jauh instans DB ini tetap tertinggal di belakang instans DB penulis dalam milidetik.

Catatan penggunaan

Fungsi ini menunjukkan statistik replikasi untuk klaster DB Aurora. Untuk setiap instans DB Aurora PostgreSQL di klaster, fungsi menunjukkan barisan data yang menyertakan replika lintas Wilayah dalam konfigurasi basis data global.

Anda dapat menjalankan fungsi ini dari instans apa pun di klaster DB Aurora PostgreSQL atau basis data global Aurora PostgreSQL. Fungsi mengembalikan detail tentang lag untuk semua instans replika.

Untuk mempelajari selengkapnya tentang pemantauan lag menggunakan fungsi ini (`aurora_global_db_instance_status`) atau dengan menggunakan `aurora_global_db_status`, lihat Memantau basis data global berbasis Aurora Postgre SQL.

Untuk informasi selengkapnya tentang basis data global Aurora, lihat Ikhtisar Database Global Amazon Aurora.

Untuk memulai dengan database global Aurora, lihat Memulai dengan Amazon Aurora Global Database atau lihat Amazon Aurora. FAQs

Contoh

Contoh ini menunjukkan statistik instans lintas Wilayah.

```
=> SELECT *
  FROM aurora_global_db_instance_status();
               server_id                |              session_id              |
  aws_region  | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------+--------------------------------------
+--------------+-------------+------------------+----------------+--------------
+----------------------+------------------------
 db-119-001-instance-01                  | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                [NULL]
 db-119-001-instance-02                  | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                     6
```

```
 db-119-001-instance-03                  | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-04                  | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-instance-05                  | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-global-instance-1            | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                    996
 db-119-001-global-instance-1            | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     14
 db-119-001-global-instance-1-eu-west-2a  | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                      0
```

Contoh ini menunjukkan cara memeriksa lag replika global dalam milidetik.

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |               server_id                |
 visibility_lag_in_msec
-------------+--------------+----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                 |
 [NULL]
 Secondary   | eu-central-1 | db-119-001-global-instance-1           |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                 |
     10
 Secondary   | eu-west-1    | db-119-001-instance-03                 |
      9
```

```
 Secondary   | eu-west-1    | db-119-001-instance-04                  |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                  |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1            |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

Contoh ini menunjukkan cara memeriksa min, max dan rata-rata lag per Wilayah AWS dari konfigurasi database global.

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
  FROM aurora_global_db_instance_status()
 WHERE aws_region NOT IN (SELECT   aws_region
                            FROM aurora_global_db_instance_status()
                           WHERE session_id='MASTER_SESSION_ID')
                        GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

## aurora_global_db_status

Menampilkan informasi tentang berbagai aspek lag basis data global Aurora, khususnya, lag penyimpanan Aurora yang mendasarinya (disebut lag daya tahan) dan lag antara sasaran titik pemulihan (RPO).

Sintaksis

```
aurora_global_db_status()
```

Argumen

Tidak ada.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `aws_region`— Di Wilayah AWS mana cluster DB ini berada. Untuk daftar lengkap Wilayah AWS berdasarkan mesin, lihatWilayah dan Zona Ketersediaan.
- `highest_lsn_written` – nomor urutan log (LSN) tertinggi yang saat ini ada di klaster DB ini. Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNs dipesan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi selanjutnya.
- `durability_lag_in_msec` – Perbedaan nilai stempel waktu antara `highest_lsn_written` di klaster DB sekunder dan `highest_lsn_written` klaster DB primer. Nilai -1 mengidentifikasi klaster DB primer dari basis data global Aurora.
- `rpo_lag_in_msec` – Lag sasaran titik pemulihan (RPO). Lag RPO adalah waktu yang dibutuhkan COMMIT transaksi pengguna terbaru untuk disimpan di klaster DB sekunder setelah disimpan di klaster DB primer dari basis data global Aurora. Nilai -1 menunjukkan klaster DB primer (dan dengan demikian, lag tidak relevan).

  Secara sederhana, metrik ini menghitung tujuan titik pemulihan untuk setiap klaster DB Aurora PostgreSQL di basis data global Aurora, yaitu, berapa banyak data yang mungkin hilang jika ada pemadaman. Seperti halnya lag, RPO diukur dalam waktu.
- `last_lag_calculation_time` – Stempel waktu yang menentukan kapan nilai terakhir dihitung untuk `durability_lag_in_msec` dan `rpo_lag_in_msec`. Nilai waktu seperti `1970-01-01 00:00:00+00` berarti ini adalah klaster DB primer.
- `feedback_epoch` – Masa yang digunakan klaster DB sekunder saat menghasilkan informasi hot standby. Hot standby adalah instans DB yang mendukung koneksi dan kueri saat DB primer berada dalam mode pemulihan atau mode siaga. Informasi hot standby mencakup masa (titik waktu) dan detail lain tentang instans DB yang digunakan sebagai hot standby. Untuk informasi selengkapnya, lihat Hot standby di dokumentasi PostgreSQL.

- `feedback_xmin` – ID transaksi aktif minimum (terlawas) yang digunakan oleh klaster DB sekunder.

Catatan penggunaan

Semua versi Aurora PostgreSQL yang tersedia saat ini mendukung fungsi ini. Fungsi ini menunjukkan statistik replikasi untuk basis data global Aurora. Hal ini menampilkan satu baris untuk setiap klaster DB dalam basis data global Aurora PostgreSQL. Anda dapat menjalankan fungsi ini dari instans apa pun di basis data global Aurora PostgreSQL.

Untuk mengevaluasi lag replikasi basis data global Aurora, yang merupakan lag data yang terlihat, lihat aurora_global_db_instance_status.

Untuk mempelajari selengkapnya tentang menggunakan `aurora_global_db_status` dan `aurora_global_db_instance_status` memantau lag basis data global Aurora, lihat Memantau basis data global berbasis Aurora Postgre SQL. Untuk informasi selengkapnya tentang basis data global Aurora, lihat Ikhtisar Database Global Amazon Aurora.

Contoh

Contoh ini menunjukkan cara menampilkan statistik penyimpanan lintas wilayah.

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+----------------------------+----------------+---------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |              0 |              0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |              0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |              0 |         12640
```

## aurora_list_builtins

Daftar semua fungsi default Aurora PostgreSQL yang tersedia, bersama dengan deskripsi singkat dan detail fungsi.

Sintaksis

```
aurora_list_builtins()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_list_builtins`.

```
=> SELECT *
FROM aurora_list_builtins();

               Name                | Result data type |                   Argument data
 types                     | Type | Volatility |  Parallel  | Security |
             Description
-----------------------------------+------------------
+-------------------------------------------------------------+------+------------
+------------+----------
+-------------------------------------------------------------------------
 aurora_version                    | text             |
                          | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type             | SETOF record     | OUT type_id smallint, OUT
 type_name text                 | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event            | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                   |                  |.me text
                           |      |            |            |          |
```

```
aurora_list_builtins              | SETOF record     | OUT "Name" text, OUT "Result
data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
Aurora built-in functions
                                  |                  |.ent data types" text, OUT
"Type" text, OUT "Volatility" .|      |            |            |          |
                                  |                  |.text, OUT "Parallel" text, OUT
"Security" text, OUT "Des.|      |            |            |          |
                                  |                  |.cription" text
                         |      |            |            |          |
.
.
.
aurora_stat_file                  | SETOF record     | OUT filename text, OUT
allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
all files present in Aurora storage
                                  |                  |.bytes bigint
                         |      |            |            |          |
aurora_stat_get_db_commit_latency | bigint           | oid
                          | func | stable     | restricted | invoker  | Per DB commit
latency in microsecs
```

## aurora_replica_status

Menampilkan status semua simpul pembaca Aurora PostgreSQL.

Sintaksis

```
aurora_replica_status()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `server_id` – ID instans DB (pengidentifikasi).
- `session_id` – Pengidentifikasi unik untuk sesi saat ini, dikembalikan untuk instans utama dan instans pembaca sebagai berikut:

  - Untuk contoh instans primer, `session_id` selalu `` `MASTER_SESSION_ID' ``.
  - Untuk instans pembaca, `session_id` selalu merupakan `UUID` (pengidentifikasi unik universal) dari instans pembaca.
- `durable_lsn` – Nomor urutan log (LSN) yang disimpan di penyimpanan.
  - Untuk volume primer, volume primer LSN (VDL) durabel yang saat ini berlaku.
  - Untuk setiap volume sekunder, VDL primer yang sekundernya telah berhasil diterapkan.

> **Note**
>
> Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNs dipesan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi yang terjadi kemudian dalam urutan.

- `highest_lsn_rcvd` – LSN tertinggi (terbaru) yang diterima oleh instans DB dari instans DB penulis.
- `current_read_lsn` – LSN dari snapshot terbaru yang telah diterapkan untuk semua pembaca.
- `cur_replay_latency_in_usec` – Jumlah mikrodetik yang diharapkan untuk memutar ulang log di sekunder.
- `active_txns` – Jumlah transaksi yang aktif saat ini.
- `is_current` – Tidak digunakan.
- `last_transport_error` – Kode kesalahan replikasi terakhir.
- `last_error_timestamp` – Stempel waktu kesalahan replikasi terakhir.
- `last_update_timestamp` – Stempel waktu pembaruan terakhir ke status replika. Dari Aurora PostgreSQL 13.9, nilai `last_update_timestamp` untuk instans DB yang terhubung dengan Anda diatur ke `NULL`.
- `feedback_xmin` – Hot standby feedback_xmin dari replika. ID transaksi aktif minimum (terlawas) yang digunakan oleh instans DB.
- `feedback_epoch` – Masa yang digunakan instans DB saat menghasilkan informasi hot standby.
- `replica_lag_in_msec`— Waktu instance pembaca tertinggal dari instance penulis, dalam milidetik.
- `log_stream_speed_in_kib_per_second` – Throughput log stream dalam kilobyte per detik.
- `log_buffer_sequence_number` – Nomor urutan buffer log.
- `oldest_read_view_trx_id` – Tidak digunakan.

- `oldest_read_view_lsn` – LSN terlawas yang digunakan oleh instans DB untuk membaca dari penyimpanan.
- `pending_read_ios` – Pembacaan halaman luar biasa yang masih tertunda pada replika.
- `read_ios` – Jumlah pembacaan halaman pada replika.
- `iops` – Tidak digunakan.
- `cpu`— Penggunaan CPU dari Aurora Storage Daemon untuk setiap node di cluster. Untuk informasi tentang penggunaan CPU oleh instans, lihatMetrik tingkat instans untuk Amazon Aurora.

Catatan penggunaan

Semua versi Aurora PostgreSQL yang tersedia saat ini mendukung fungsi ini. Fungsi `aurora_replica_status` mengembalikan nilai dari manajer status replika klaster DB Aurora PostgreSQL. Anda dapat menggunakan fungsi ini untuk mendapatkan informasi tentang status replikasi pada klaster DB PostgreSQL Aurora Anda, termasuk metrik untuk semua instans DB di klaster DB Aurora Anda. Misalnya, Anda dapat melakukan hal berikut:

- Mendapatkan informasi tentang jenis instans (penulis, pembaca) di klaster DB Aurora PostgreSQL – Anda dapat memperoleh informasi ini dengan memeriksa nilai-nilai kolom berikut:
  - `server_id` – Berisi nama instans yang Anda tentukan saat membuat instans. Dalam beberapa kasus, seperti untuk instans primer (penulis), nama biasanya dibuat untuk Anda dengan menambahkan -instance-1 ke nama yang Anda buat untuk klaster DB Aurora PostgreSQL Anda.
  - `session_id` – Bidang `session_id` menunjukkan apakah instans adalah pembaca atau penulis. Untuk instans penulis, `session_id` selalu diatur ke `"MASTER_SESSION_ID"`. Untuk instans pembaca, `session_id` diatur ke `UUID` pembaca tertentu.
- Mendiagnosis masalah replikasi umum, seperti lag replika – Lag replica adalah waktu dalam milidetik cache halaman dari instans pembaca berada di belakang instans penulis. Lag ini terjadi karena klaster Aurora menggunakan replikasi asinkron, seperti yang dijelaskan dalam Replikasi dengan Amazon Aurora. Lagi ini ditampilkan di kolom `replica_lag_in_msec` dalam hasil yang dikembalikan oleh fungsi ini. Lag juga dapat terjadi ketika kueri dibatalkan karena konflik dengan pemulihan pada server siaga. Anda dapat memeriksa `pg_stat_database_conflicts()` untuk memverifikasi bahwa konflik seperti itu menyebabkan lag replika (atau tidak). Untuk informasi selengkapnya, lihat The Statistics Collector di Dokumentasi PostgreSQL. Untuk mempelajari lebih lanjut tentang ketersediaan dan replikasi tinggi, lihat Amazon FAQs Aurora.

Amazon CloudWatch menyimpan `replica_lag_in_msec` hasil dari waktu ke waktu, sebagai `AuroraReplicaLag` metrik. Untuk informasi tentang penggunaan CloudWatch metrik untuk Aurora, lihat Memantau Amazon Aurora dengan Amazon CloudWatch

Untuk mempelajari selengkapnya tentang pemecahan masalah replika dan pemulaian ulang pembacaan Aurora, lihat Mengapa replika pembacaan Amazon Aurora saya tertinggal dan memulai ulang? di Pusat AWS Dukungan.

Contoh

Contoh berikut menunjukkan cara mendapatkan status replikasi dari semua instans di klaster DB Aurora PostgreSQL:

```
=> SELECT *
FROM aurora_replica_status();
```

Contoh berikut menunjukkan contoh instans penulis di klaster DB Aurora PostgreSQL `docs-lab-apg-main`:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

Contoh berikut mencantumkan semua instans pembaca dalam sebuah klaster:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
```

```
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

Contoh berikut mencantumkan semua instans, seberapa jauh setiap instans tertinggal dari penulis, dan berapa lama sejak pembaruan terakhir:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
(3 rows)
```

## aurora_stat_activity

Mengembalikan satu baris per proses server, menampilkan informasi yang terkait dengan aktivitas saat ini dari proses itu.

Sintaks

```
aurora_stat_activity();
```

Argumen

Tidak ada

Jenis pengembalian

Mengembalikan satu baris per proses server. Selain `pg_stat_activity` kolom, bidang berikut ditambahkan:

- planid - pengidentifikasi rencana

Catatan penggunaan

Tampilan tambahan untuk `pg_stat_activity` mengembalikan kolom yang sama dengan `plan_id` kolom tambahan yang menunjukkan rencana eksekusi kueri saat ini.

`aurora_compute_plan_id`harus diaktifkan agar tampilan mengembalikan plan_id.

Fungsi ini tersedia dari Aurora PostgreSQL versi 14.10, 15.5, dan untuk semua versi lain yang lebih baru.

Contoh

Contoh kueri di bawah ini menggabungkan beban teratas dengan query_id dan plan_id.

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+-----------------------+-------------
 11    | -5471422286312252535 | -2054628807
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
(3 rows)
```

Jika rencana yang digunakan untuk query_id berubah, plan_id baru akan dilaporkan oleh aurora_stat_activity.

```
count  |  query_id            |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

## aurora_stat_backend_waits

Menampilkan statistik untuk aktivitas menunggu untuk proses backend tertentu.

Sintaksis

```
aurora_stat_backend_waits(pid)
```

Argumen

`pid` – ID untuk proses backend. Anda dapat memperoleh proses IDs dengan menggunakan `pg_stat_activity` tampilan.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `type_id` – Angka yang menunjukkan jenis peristiwa tunggu, seperti 1 untuk kunci ringan (`LWLock`), 3 untuk kunci, atau 6 untuk sesi klien, untuk menyebutkan beberapa contoh. Nilai-nilai ini menjadi bermakna ketika Anda menggabungkan hasil fungsi ini dengan kolom dari fungsi `aurora_stat_wait_type`, seperti yang ditunjukkan dalam Contoh.
- `event_id` – Nomor pengidentifikasi untuk peristiwa tunggu. Gabungkan nilai ini dengan kolom dari `aurora_stat_wait_event` untuk mendapatkan nama peristiwa yang bermakna.
- `waits` – Hitungan jumlah tunggu yang terakumulasi untuk ID proses yang ditentukan.
- `wait_time` – Waktu tunggu dalam milidetik.

Catatan penggunaan

Anda dapat menggunakan fungsi ini untuk menganalisis peristiwa tunggu backend (sesi) tertentu yang terjadi sejak koneksi dibuka. Untuk mendapatkan informasi yang lebih bermakna tentang nama dan jenis peristiwa tunggu, Anda dapat menggabungkan fungsi `aurora_stat_wait_type`

dan `aurora_stat_wait_event` ini, dengan menggunakan JOIN seperti yang ditunjukkan dalam contoh.

Contoh

Contoh ini menunjukkan semua tunggu, jenis, dan nama peristiwa untuk proses backend ID 3027.

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
type_name |       event_name       | waits | wait_time
----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

Contoh ini menunjukkan jenis tunggu saat ini dan kumulatif serta peristiwa tunggu untuk semua sesi aktif (`pg_stat_activity state <> 'idle'`) (tetapi tanpa sesi saat ini yang menjalankan fungsi (`pid <> pg_backend_pid()`).

```
=> SELECT a.pid,
            a.usename,
            a.app_name,
            a.current_wait_type,
            a.current_wait_event,
            a.current_state,
            wt.type_name AS wait_type,
            we.event_name AS wait_event,
            a.waits,
            a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
```

```
                    coalesce(wait_event_type,'CPU') AS current_wait_type,
                    coalesce(wait_event,'CPU') AS current_wait_event,
                    state AS current_state,
                    (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | username  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
-------+----------+----------+-------------------+--------------------+---------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22903 |    760498
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | lock_manager           | 10012 |    223207
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple                  | 20315 |  63081529
 .
 .
 .
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | WALWrite               | 93293 |    237440
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | XactSync               | 13010 |  19525143
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | ProcArrayLock          |     6 |        53
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1913 |     25450
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22874 |    778005
 .
 .
 .
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | ProcArrayLock          |     3 |        71
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | wal_insert             |  1940 |     27741
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | buffer_content         | 22962 |    776352
```

```
 30109 | postgres | pgbench  | IO               | XactSync          | active       |
 LWLock    | lock_manager          |  9879 |    218826
 30109 | postgres | pgbench  | IO               | XactSync          | active       |
 Lock      | tuple                 | 20401 |  63581306
 30109 | postgres | pgbench  | IO               | XactSync          | active       |
 Lock      | transactionid         | 50769 | 211645008
 30109 | postgres | pgbench  | IO               | XactSync          | active       |
 Client    | ClientRead            | 89901 |  44192439
```

Contoh ini menunjukkan jenis tunggu kumulatif saat ini dan tiga teratas (3) serta peristiwa tunggu kumulatif untuk semua sesi aktif (`pg_stat_activity state <> 'idle'`) tidak termasuk sesi saat ini (`pid <>pg_backend_pid()`).

```
=> SELECT top3.*
       FROM (SELECT a.pid,
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
               FROM (SELECT pid,
                            usename,
                            left(application_name,16) AS app_name,
                            coalesce(wait_event_type,'CPU') AS current_wait_type,
                            coalesce(wait_event,'CPU') AS current_wait_event,
                            state AS current_state,
                            (aurora_stat_backend_waits(pid)).*
                       FROM pg_stat_activity
                      WHERE pid <> pg_backend_pid()
                        AND state <> 'idle') a
       NATURAL JOIN aurora_stat_wait_type() wt
       NATURAL JOIN aurora_stat_wait_event() we) top3
 WHERE RANK <=3;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
-------+----------+----------+-------------------+--------------------+---------------
+-----------+-----------------+---------+------------+------
```

```
20567 | postgres | psql     | CPU              | CPU               | active        |
LWLock    | wal_insert      |   25000 |   67512003 |    1
20567 | postgres | psql     | CPU              | CPU               | active        |
IO        | WALWrite        | 3071758 |    1016961 |    2
20567 | postgres | psql     | CPU              | CPU               | active        |
IO        | BufFileWrite    |   20750 |     184559 |    3
27743 | postgres | pgbench  | Lock             | transactionid     | active        |
Lock      | transactionid   |  237350 | 1265580011 |    1
27743 | postgres | pgbench  | Lock             | transactionid     | active        |
Lock      | tuple           |   93641 |  341472318 |    2
27743 | postgres | pgbench  | Lock             | transactionid     | active        |
Client    | ClientRead      |  417556 |  204796837 |    3
.
.
.
27745 | postgres | pgbench  | IO               | XactSync          | active        |
Lock      | transactionid   |  238068 | 1265816822 |    1
27745 | postgres | pgbench  | IO               | XactSync          | active        |
Lock      | tuple           |   93210 |  338312247 |    2
27745 | postgres | pgbench  | IO               | XactSync          | active        |
Client    | ClientRead      |  419157 |  207836533 |    3
27746 | postgres | pgbench  | Lock             | transactionid     | active        |
Lock      | transactionid   |  237621 | 1264528811 |    1
27746 | postgres | pgbench  | Lock             | transactionid     | active        |
Lock      | tuple           |   93563 |  339799310 |    2
27746 | postgres | pgbench  | Lock             | transactionid     | active        |
Client    | ClientRead      |  417304 |  208372727 |    3
```

## aurora_stat_bgwriter

`aurora_stat_bgwriter` adalah tampilan statistik yang menampilkan informasi tentang penulisan cache Optimized Reads.

### Sintaksis

```
aurora_stat_bgwriter()
```

### Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan semua kolom `pg_stat_bgwriter` dan kolom tambahan berikut. Untuk informasi selengkapnya tentang kolom `pg_stat_bgwriter`, lihat `pg_stat_bgwriter`.

Anda dapat mengatur ulang statistik untuk fungsi ini menggunakan `pg_stat_reset_shared("bgwriter")`.

- `orcache_blks_written` – Jumlah total blok data cache pembacaan yang dioptimalkan ditulis.
- `orcache_blk_write_time` – Jika `track_io_timing` diaktifkan, fungsi ini melacak total waktu yang dihabiskan untuk menulis blok file data cache yang dioptimalkan, dalam milidetik. Untuk informasi selengkapnya, lihat track_io_timing.

Catatan penggunaan

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

Contoh

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

## aurora_stat_database

Fungsi ini membawa semua kolom pg_stat_database dan menambahkan kolom baru pada akhirnya.

Sintaksis

```
aurora_stat_database()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan semua kolom `pg_stat_database` dan kolom tambahan berikut. Untuk informasi selengkapnya tentang kolom `pg_stat_database`, lihat `pg_stat_database`.

- `storage_blks_read` – Jumlah total blok bersama yang dibaca dari penyimpanan aurora dalam basis data ini.
- `orcache_blks_hit` – Jumlah total hit cache pembacaan yang dioptimalkan dalam basis data ini.
- `local_blks_read` – Jumlah total blok lokal yang dibaca dalam basis data ini.
- `storage_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu yang dihabiskan untuk membaca blok file data dari penyimpanan aurora dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `local_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu yang dihabiskan untuk membaca blok file data lokal dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `orcache_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu yang dihabiskan untuk membaca blok file data dari cache pembacaan yang dioptimalkan dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.

> **Note**
>
> Nilai `blks_read` adalah jumlah dari `storage_blks_read`, `orcache_blks_hit`, dan `local_blks_read`.
> Nilai `blk_read_time` adalah jumlah dari `storage_blk_read_time`, `orcache_blk_read_time`, dan `local_blk_read_time`.

Catatan penggunaan

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

### Contoh

Contoh berikut menunjukkan bagaimana fungsi ini membawa semua kolom `pg_stat_database` dan menambahkan 6 kolom baru pada akhirnya:

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
datid                    | 14717
datname                  | postgres
numbackends              | 1
xact_commit              | 223
xact_rollback            | 4
blks_read                | 1059
blks_hit                 | 11456
tup_returned             | 27746
tup_fetched              | 5220
tup_inserted             | 165
tup_updated              | 42
tup_deleted              | 91
conflicts                | 0
temp_files               | 0
temp_bytes               | 0
deadlocks                | 0
checksum_failures        |
checksum_last_failure    |
blk_read_time            | 3358.689
blk_write_time           | 0
session_time             | 1076007.997
active_time              | 3684.371
idle_in_transaction_time | 0
sessions                 | 10
sessions_abandoned       | 0
sessions_fatal           | 0
sessions_killed          | 0
stats_reset              | 2023-01-12 20:15:17.370601+00
orcache_blks_hit         | 425
orcache_blk_read_time    | 89.934
storage_blks_read        | 623
storage_blk_read_time    | 3254.914
local_blks_read          | 0
local_blk_read_time      | 0
```

## aurora_stat_dml_activity

Melaporkan aktivitas kumulatif untuk setiap jenis operasi bahasa manipulasi data (DML) pada basis data dalam klaster Aurora PostgreSQL.

Sintaksis

```
aurora_stat_dml_activity(database_oid)
```

Argumen

database_oid

ID Objek (OID) dari basis data di klaster Aurora PostgreSQL.

Jenis pengembalian

Catatan SETOF

Catatan penggunaan

Fungsi `aurora_stat_dml_activity` hanya tersedia dengan Aurora PostgreSQL rilis 3.1 yang kompatibel dengan mesin PostgreSQL 11.6 dan yang lebih baru.

Gunakan fungsi ini pada klaster Aurora PostgreSQL dengan sejumlah besar basis data untuk mengidentifikasi basis data mana yang memiliki aktivitas DML lebih banyak atau lebih lambat, atau keduanya.

Fungsi `aurora_stat_dml_activity` mengembalikan berapa kali operasi dijalankan dan latensi kumulatif dalam mikrodetik untuk operasi SELECT, INSERT, UPDATE, dan DELETE. Laporan ini hanya mencakup operasi DML yang berhasil.

Anda dapat mengatur ulang statistik ini dengan menggunakan fungsi akses statistik PostgreSQL `pg_stat_reset`. Anda dapat memeriksa kapan terakhir kali statistik ini diatur ulang dengan menggunakan fungsi `pg_stat_get_db_stat_reset_time`. Untuk informasi selengkapnya tentang fungsi akses statistik PostgreSQL, lihat The Statistics Collector dalam dokumentasi PostgreSQL.

Contoh

Contoh berikut menunjukkan cara melaporkan statistik aktivitas DML untuk basis data yang terhubung.

```
--Define the oid variable from connected database by using \gset
=> SELECT oid,
          datname
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
--------------+--------------------------+--------------+--------------------------
+--------------+--------------------------+--------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |            1 |                    53027


-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+-------------
select_count             | 178957
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

Contoh berikut menunjukkan statistik aktivitas DML untuk semua basis data di klaster Aurora PostgreSQL. Klaster ini memiliki dua basis data, `postgres` dan `mydb`. Daftar yang dipisahkan koma sesuai dengan bidang `select_count`, `select_latency_microsecs`, `insert_count`, `insert_latency_microsecs`, `update_count`, `update_latency_microsecs`, `delete_count`, dan `delete_latency_microsecs`.

Aurora PostgreSQL membuat dan menggunakan basis data sistem bernama `rdsadmin` untuk mendukung operasi administratif seperti pencadangan, pemulihan, pemeriksaan kondisi, replikasi, dan sebagainya. Operasi DML ini tidak berdampak pada klaster Aurora PostgreSQL Anda.

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                   aurora_stat_dml_activity
-------+---------------
+-------------------------------------------------------------
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
 14007 | postgres       |
 (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

Contoh berikut menunjukkan statistik aktivitas DML untuk semua basis data, diatur dalam kolom untuk keterbacaan yang lebih baik.

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
---------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
            |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
            | 0            | 0                        | 0            | 0
 template1      |              |                          |              |
            |              |                          |              |
```

```
 fault_test     | 66           | 452099                  | 0            | 0
            | 0            | 0                       | 0            | 0
 db_access_test | 1            | 5982                    | 0            | 0
            | 0            | 0                       | 0            | 0
 postgres       | 42035        | 95179203                | 5752         | 2678832898
            | 21157        | 441883182488            | 2            | 1520
 mydb           | 71           | 453514                  | 0            | 0
            | 1            | 190                     | 1            | 152
```

Contoh berikut menunjukkan latensi kumulatif rata-rata (latensi kumulatif dibagi dengan hitungan) untuk setiap operasi DML untuk basis data dengan OID 16401.

```
=> SELECT select_count,
          select_latency_microsecs,
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

## aurora_stat_get_db_commit_latency

Mendapatkan latensi commit kumulatif dalam mikrodetik untuk basis data Aurora PostgreSQL. Latensi commit diukur sebagai waktu antara saat klien mengirimkan permintaan commit dan saat menerima pengakuan commit.

Sintaksis

```
aurora_stat_get_db_commit_latency(database_oid)
```

Argumen

database_oid

ID Objek (OID) dari basis data Aurora PostgreSQL.

Jenis pengembalian

Catatan SETOF

Catatan penggunaan

Amazon CloudWatch menggunakan fungsi ini untuk menghitung latensi komit rata-rata. Untuk informasi selengkapnya tentang CloudWatch metrik Amazon dan cara memecahkan masalah latensi komit tinggi, lihat Melihat metrik di konsol Amazon RDS serta Membuat keputusan yang lebih baik tentang Amazon RDS dengan metrik Amazon. CloudWatch

Anda dapat mengatur ulang statistik ini dengan menggunakan fungsi akses statistik PostgreSQL `pg_stat_reset`. Anda dapat memeriksa kapan terakhir kali statistik ini diatur ulang dengan menggunakan fungsi `pg_stat_get_db_stat_reset_time`. Untuk informasi selengkapnya tentang fungsi akses statistik PostgreSQL, lihat The Statistics Collector dalam dokumentasi PostgreSQL.

Contoh

Contoh berikut mendapatkan latensi commit kumulatif untuk setiap basis data di klaster `pg_database`.

```
=> SELECT oid,
```

```
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |    datname     | aurora_stat_get_db_commit_latency
-------+----------------+-----------------------------------
 14006 | template0      |                                 0
 16384 | rdsadmin       |                         654387789
     1 | template1      |                                 0
 16401 | mydb           |                            229556
 69768 | postgres       |                             22011
```

Contoh berikut mendapatkan latensi commit kumulatif untuk basis data yang terhubung saat ini. Sebelum memanggil fungsi `aurora_stat_get_db_commit_latency`, contoh pertama menggunakan `\gset` guna mendefinisikan variabel untuk argumen `oid` dan menetapkan nilainya dari basis data yang terhubung.

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

Contoh berikut mendapatkan latensi commit kumulatif untuk basis data `mydb` di klaster `pg_database`. Setelah itu, contoh tersebut mengatur ulang statistik ini dengan menggunakan fungsi `pg_stat_reset` dan menampilkan hasilnya. Terakhir, contoh tersebut menggunakan fungsi `pg_stat_get_db_stat_reset_time` untuk memeriksa kapan terakhir kali statistik diatur ulang.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';

  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
```

```
 16427 | mydb      |                              3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------


=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                                 6


=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);

 pg_stat_get_db_stat_reset_time
--------------------------------
 2021-04-29 21:36:15.707399+00
```

## aurora_stat_logical_wal_cache

Menampilkan penggunaan cache log write-ahead (WAL) per slot.

Sintaksis

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- name – Nama slot replikasi.

- `active_pid` – ID proses walsender.
- `cache_hit` – Jumlah total hit cache wal sejak atur ulang terakhir.
- `cache_miss` – Jumlah total hilang cache wal sejak atur ulang terakhir.
- `blks_read` – Jumlah total permintaan baca cache wal.
- `hit_rate` – Laju hit cache WAL (cache_hit/blks_read).
- `last_reset_timestamp` – Terakhir kali penghitung diatur ulang.

Catatan penggunaan

Fungsi ini tersedia untuk versi Aurora PostgreSQL berikut.

- 15.2 dan semua versi yang lebih tinggi
- 14.7 dan versi yang lebih tinggi
- Versi 13.8 dan versi yang lebih tinggi
- Versi 12.12 dan versi yang lebih tinggi
- Versi 11.17 dan versi yang lebih tinggi

Contoh

Contoh berikut menampilkan dua slot replikasi dengan hanya satu yang aktif. Fungsi `aurora_stat_logical_wal_cache`.

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
(2 rows)
```

## aurora_stat_memctx_usage

Melaporkan penggunaan konteks memori untuk setiap proses PostgreSQL.

Sintaksis

```
aurora_stat_memctx_usage()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `pid` – ID dari proses.
- `name` – Nama konteks memori.
- `allocated` – Jumlah byte yang diperoleh dari subsistem memori yang mendasarinya oleh konteks memori.
- `used` – Jumlah byte yang dijalankan untuk klien dari konteks memori.
- `instances` – Hitungan konteks yang ada saat ini dari jenis ini.

Catatan penggunaan

Fungsi ini menampilkan penggunaan konteks memori untuk setiap proses PostgreSQL. Beberapa proses diberi label `anonymous`. Proses tidak diekspos karena mengandung kata kunci yang dibatasi.

Fungsi ini tersedia mulai dengan versi Aurora PostgreSQL berikut:

- Versi 15.3 dan versi 15 yang lebih tinggi
- 14.8 dan versi 14 yang lebih tinggi
- 13.11 dan versi 13 yang lebih tinggi
- 12.15 dan versi 12 yang lebih tinggi
- Versi 11.20 dan versi 11 yang lebih tinggi

Contoh

Contoh berikut menampilkan hasil pemanggilan fungsi `aurora_stat_memctx_usage`.

```
=> SELECT *
     FROM aurora_stat_memctx_usage();
```

```
    pid| name                             | allocated |   used  | instances
-------+----------------------------------+-----------+---------+-----------
123864 | Miscellaneous                    |     19520 |   15064 |         3
123864 | Aurora File Context              |      8192 |     616 |         1
123864 | Aurora WAL Context               |      8192 |     296 |         1
123864 | CacheMemoryContext               |    524288 |  422600 |         1
123864 | Catalog tuple context            |     16384 |   13736 |         1
123864 | ExecutorState                    |     32832 |   28304 |         1
123864 | ExprContext                      |      8192 |    1720 |         1
123864 | GWAL record construction         |      1024 |     832 |         1
123864 | MdSmgr                           |      8192 |     296 |         1
123864 | MessageContext                   |    532480 |  353832 |         1
123864 | PortalHeapMemory                 |      1024 |     488 |         1
123864 | PortalMemory                     |      8192 |     576 |         1
123864 | printtup                         |      8192 |     296 |         1
123864 | RelCache hash table entries      |      8192 |    8152 |         1
123864 | RowDescriptionContext            |      8192 |    1344 |         1
123864 | smgr relation context            |      8192 |     296 |         1
123864 | Table function arguments         |      8192 |     352 |         1
123864 | TopTransactionContext            |      8192 |     632 |         1
123864 | TransactionAbortContext          |     32768 |     296 |         1
123864 | WAL record construction          |     50216 |   43904 |         1
123864 | hash table                       |     65536 |   52744 |         6
123864 | Relation metadata                |    191488 |  124240 |        87
104992 | Miscellaneous                    |      9280 |    7728 |         3
104992 | Aurora File Context              |      8192 |     376 |         1
104992 | Aurora WAL Context               |      8192 |     296 |         1
104992 ||Autovacuum Launcher              |      8192 |     296 |         1
104992 | Autovacuum database list         |     16384 |     744 |         2
104992 | CacheMemoryContext               |    262144 |  140288 |         1
104992 | Catalog tuple context            |      8192 |     296 |         1
104992 | GWAL record construction         |      1024 |     832 |         1
104992 | MdSmgr                           |      8192 |     296 |         1
104992 | PortalMemory                     |      8192 |     296 |         1
104992 | RelCache hash table entries      |      8192 |     296 |         1
104992 | smgr relation context            |      8192 |     296 |         1
104992 | Autovacuum start worker (tmp)    |      8192 |     296 |         1
104992 | TopTransactionContext            |     16384 |     592 |         2
104992 | TransactionAbortContext          |     32768 |     296 |         1
104992 | WAL record construction          |     50216 |   43904 |         1
104992 | hash table                       |     49152 |   34024 |         4
(39 rows)
```

Beberapa kata kunci terbatas akan disembunyikan dan output akan terlihat sebagai berikut:

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                             | allocated |   used  | instances
-------+----------------------------------+-----------+---------+-----------
  5482 | anonymous                        |      8192 |     456 |         1
  5482 | anonymous                        |      8192 |     296 |         1
```

## aurora_stat_optimized_reads_cache

Fungsi ini menunjukkan statistik cache berjenjang.

Sintaksis

```
aurora_stat_optimized_reads_cache()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `total_size` – Total ukuran cache bacaan yang dioptimalkan.
- `used_size` – Ukuran halaman yang digunakan dalam cache bacaan yang dioptimalkan.

Catatan penggunaan

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

Contoh

Contoh berikut menampilkan output pada contoh instans r6gd.8xlarge:

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
                as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

## aurora_stat_plans

Mengembalikan baris untuk setiap rencana eksekusi dilacak.

Sintaks

```
aurora_stat_plans(
    showtext
)
```

Argumen

- showtext - Tampilkan kueri dan rencana teks. Nilai yang valid adalah NULL, benar atau salah. True akan menampilkan kueri dan rencana teks.

Jenis pengembalian

Mengembalikan baris untuk setiap rencana dilacak yang berisi semua kolom dari `aurora_stat_statements` dan kolom tambahan berikut.

- planid - pengidentifikasi rencana
- explain_plan - jelaskan teks rencana
- plan_type:
  - `no plan`- tidak ada rencana yang ditangkap
  - `estimate`- rencana ditangkap dengan perkiraan biaya
  - `actual`- rencana ditangkap dengan EXPLORE ANALYSIS
- plan_captured_time — terakhir kali rencana ditangkap

Catatan penggunaan

`aurora_compute_plan_id`harus diaktifkan dan `pg_stat_statements` harus dalam `shared_preload_libraries` agar rencana dilacak.

Jumlah paket yang tersedia dikendalikan oleh nilai yang ditetapkan dalam `pg_stat_statements.max` parameter. Saat `aurora_compute_plan_id` diaktifkan, Anda dapat melacak paket hingga nilai yang ditentukan ini di`aurora_stat_plans`.

Fungsi ini tersedia dari Aurora PostgreSQL versi 14.10, 15.5, dan untuk semua versi lain yang lebih baru.

Contoh

Dalam contoh di bawah ini, dua rencana yang untuk pengidentifikasi kueri -5471422286312252535 ditangkap dan statistik pernyataan dilacak oleh planid.

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
db1-# from aurora_stat_plans(true)
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                explain_plan
---------+--------------------+-------------+-------------------------------
+-----------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                        +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                       +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                 +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                        +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

## aurora_stat_reset_wal_cache

Mengatur ulang penghitung untuk cache wal logis.

Sintaksis

Cara mengatur ulang slot tertentu

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

Cara mengatur ulang semua slot

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

Argumen

`NULL` atau `slot_name`

Jenis pengembalian

Pesan status, string teks

- Atur ulang penghitung cache wal logis - Pesan sukses. Teks ini dikembalikan ketika fungsi berhasil.
- Slot replikasi tidak ditemukan. Coba lagi. – Pesan kegagalan. Teks ini dikembalikan ketika fungsi tidak berhasil.

Catatan penggunaan

Fungsi ini tersedia untuk versi berikut.

- Aurora PostgreSQL versi 14.5 dan lebih tinggi
- Aurora PostgreSQL versi 13.8 dan lebih tinggi
- Aurora PostgreSQL versi 12.12 dan lebih tinggi
- Aurora SQLversion Postgre 11.17 dan lebih tinggi

Contoh

Contoh berikut menggunakan fungsi `aurora_stat_reset_wal_cache` untuk mengatur ulang slot bernama `test_results`, dan kemudian mencoba untuk mengatur ulang slot yang tidak ada.

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
------------------------------------
 Reset the logical wal cache counter.
(1 row)
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
aurora_stat_reset_wal_cache
---------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

## aurora_stat_resource_usage

Melaporkan pemanfaatan sumber daya real-time yang terdiri dari metrik sumber daya backend dan penggunaan cpu untuk semua proses backend Aurora PostgreSQL.

Sintaksis

```
aurora_stat_resource_usage()
```

Argumen

Tidak ada

Jenis pengembalian

Rekam SETOF dengan kolom:

- pid - Pengidentifikasi proses
- allocated_memory - Total memori yang dialokasikan oleh proses dalam byte
- used_memory - Sebenarnya menggunakan memori dengan proses dalam byte
- cpu_usage_percent - Persentase penggunaan CPU dari proses

Catatan penggunaan

Fungsi ini menampilkan penggunaan sumber daya backend untuk setiap proses backend Aurora PostgreSQL.

Fungsi ini tersedia mulai dengan versi Aurora PostgreSQL berikut:

- Aurora PostgreSQL 17.5 dan versi 17 yang lebih tinggi
- Aurora PostgreSQL 16.9 dan versi 16 yang lebih tinggi
- Aurora PostgreSQL 15.13 dan versi 15 yang lebih tinggi
- Aurora PostgreSQL 14.18 dan versi 14 yang lebih tinggi
- Aurora PostgreSQL 13.21 dan versi 13 yang lebih tinggi

Contoh

Contoh berikut menunjukkan output dari `aurora_stat_resource_usage` fungsi.

```
=> select * from aurora_stat_resource_usage();
 pid  | allocated_memory | used_memory |   cpu_usage_percent
------+------------------+-------------+-----------------------
  666 |          1074032 |      333544 |   0.00729274882897963
  667 |           787312 |      287360 | 0.0029263928146372746
  668 |          3076776 |     1563488 |  0.006013116835953961
  684 |           803744 |      307480 |  0.002226855426881142
 2401 |          1232992 |      943144 |                     0
  647 |             8000 |         944 |   0.48853387812429855
  659 |           319344 |      243000 | 0.0004135602076683591
  663 |           262000 |      185736 |  0.008181301476644002
  664 |             9024 |        1216 |   0.10992313082653653
(9 rows)
```

## aurora_stat_statements

Menampilkan semua kolom `pg_stat_statements` dan menambahkan lebih banyak kolom pada akhirnya.

Sintaksis

```
aurora_stat_statements(showtext boolean)
```

Argumen

showtext boolean

Jenis pengembalian

Catatan SETOF dengan semua kolom `pg_stat_statements` dan kolom tambahan berikut. Untuk informasi selengkapnya tentang kolom `pg_stat_statements`, lihat `pg_stat_statements`.

Anda dapat mengatur ulang statistik untuk fungsi ini menggunakan `pg_stat_statements_reset()`.

- `storage_blks_read` – Jumlah total blok bersama yang dibaca dari penyimpanan aurora oleh pernyataan ini.
- `orcache_blks_hit` – Jumlah total hit cache pembacaan yang dioptimalkan oleh pernyataan ini.
- `storage_blk_read_time`— Jika `track_io_timing` diaktifkan, ia melacak total waktu pernyataan yang dihabiskan untuk membaca blok bersama dari penyimpanan aurora, dalam milidetik, jika tidak nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `local_blk_read_time`— Jika `track_io_timing` diaktifkan, ia melacak total waktu pernyataan yang dihabiskan untuk membaca blok lokal, dalam milidetik, jika tidak nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `orcache_blk_read_time`— Jika `track_io_timing` diaktifkan, ia melacak total waktu pernyataan yang dihabiskan untuk membaca blok bersama dari cache pembacaan yang dioptimalkan, dalam milidetik, jika tidak nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `total_plan_peakmem`— Jumlah total nilai memori puncak selama fase perencanaan untuk semua panggilan ke pernyataan ini. Untuk melihat memori puncak rata-rata selama perencanaan pernyataan, bagi nilai ini dengan jumlah panggilan.
- `min_plan_peakmem`— Nilai memori puncak terkecil yang terlihat selama perencanaan di semua panggilan ke pernyataan ini.
- `max_plan_peakmem`— Nilai memori puncak terbesar selama perencanaan terlihat di semua panggilan ke pernyataan ini.
- `total_exec_peakmem`— Jumlah total nilai memori puncak selama fase eksekusi untuk semua panggilan ke pernyataan ini. Untuk melihat memori puncak rata-rata selama eksekusi untuk pernyataan, bagi nilai ini dengan jumlah panggilan.
- `min_exec_peakmem`—Nilai memori puncak terkecil, dalam byte, terlihat selama eksekusi di semua panggilan ke pernyataan ini.

- `max_exec_peakmem`— Nilai memori puncak terbesar, dalam byte, terlihat selama eksekusi di semua panggilan ke pernyataan ini.

> **Note**
>
> `total_plan_peakmen`,`min_plan_peakmem`, dan hanya `max_plan_peakmem` dipantau saat pengaturan `pg_stat_statements.track_planning` dihidupkan.

Catatan penggunaan

Untuk menggunakan fungsi aurora_stat_statements (), Anda harus menyertakan ekstensi dalam parameter. `pg_stat_statements` `shared_preload_libraries`

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

Kolom yang menunjukkan memori puncak tersedia dari versi berikut:

- 16.3 dan versi yang lebih tinggi
- 15.7 dan versi yang lebih tinggi
- 14.12 dan versi yang lebih tinggi

Contoh

Contoh berikut menunjukkan bagaimana ia membawa semua kolom pg_stat_statement dan menambahkan 11 kolom baru pada akhirnya:

```
=> select * from aurora_stat_statements(true) where query like 'with window_max%';
-[ RECORD 1 ]----------
+-----------------------------------------------------------------------------------------------
userid                 | 16409
dbid                   | 5
toplevel               | t
queryid                | -8347523682669847482
query                  | with window_max as (select custid, max(scratch) over (order by
 scratch rows between $1 preceding
```

```
and $2 following) wmax from ts) select sum(wmax), max(custid) from window_max
plans                   | 0
total_plan_time         | 0
min_plan_time           | 0
max_plan_time           | 0
mean_plan_time          | 0
stddev_plan_time        | 0
calls                   | 4
total_exec_time         | 254.105121
min_exec_time           | 57.503164000000005
max_exec_time           | 68.687418
mean_exec_time          | 63.52628025
stddev_exec_time        | 5.150765359979643
rows                    | 4
shared_blks_hit         | 200192
shared_blks_read        | 0
shared_blks_dirtied     | 0
shared_blks_written     | 0
local_blks_hit          | 0
local_blks_read         | 0
local_blks_dirtied      | 0
local_blks_written      | 0
temp_blks_read          | 0
temp_blks_written       | 0
blk_read_time           | 0
blk_write_time          | 0
temp_blk_read_time      | 0
temp_blk_write_time     | 0
wal_records             | 0
wal_fpi                 | 0
wal_bytes               | 0
jit_functions           | 0
jit_generation_time     | 0
jit_inlining_count      | 0
jit_inlining_time       | 0
jit_optimization_count  | 0
jit_optimization_time   | 0
jit_emission_count      | 0
jit_emission_time       | 0
storage_blks_read       | 0
orcache_blks_hit        | 0
storage_blk_read_time   | 0
local_blk_read_time     | 0
orcache_blk_read_time   | 0
```

```
total_plan_peakmem     | 0
min_plan_peakmem       | 0
max_plan_peakmem       | 0
total_exec_peakmem     | 6356224
min_exec_peakmem       | 1589056
max_exec_peakmem       | 1589056
```

## aurora_stat_system_waits

Melaporkan informasi peristiwa tunggu untuk instans DB Aurora PostgreSQL.

Sintaksis

```
aurora_stat_system_waits()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF

Catatan penggunaan

Fungsi ini mengembalikan jumlah tunggu kumulatif dan waktu tunggu kumulatif untuk setiap peristiwa tunggu yang dihasilkan oleh instans DB yang saat ini terhubung.

Recordset yang dikembalikan mencakup kolom-kolom berikut:

- `type_id` – ID dari jenis peristiwa tunggu.
- `event_id` – ID dari peristiwa tunggu.
- `waits` – Jumlah peristiwa menunggu terjadi.
- `wait_time` – Jumlah total waktu dalam mikrodetik yang dihabiskan menunggu peristiwa ini.

Statistik yang dikembalikan oleh fungsi ini diatur ulang ketika instans DB dimulai ulang.

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_stat_system_waits`.

```
=> SELECT *
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+-------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

Contoh berikut menampilkan cara menggunakan fungsi ini bersama-sama dengan `aurora_stat_wait_event` dan `aurora_stat_wait_type` untuk menghasilkan hasil yang lebih mudah dibaca.

```
=> SELECT type_name,
            event_name,
            waits,
            wait_time
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

 type_name |       event_name       |  waits   |  wait_time
-----------+------------------------+----------+--------------
 LWLock    | XidGenLock             |       11 |        12864
 LWLock    | ProcArrayLock          |      501 |       174473
 LWLock    | buffer_content         |    53171 |     23641847
 LWLock    | rdsutils               |        2 |        12764
 Lock      | tuple                  |    75686 |   2033956052
 Lock      | transactionid          |  1765147 |  47267583409
 Activity  | AutoVacuumMain         |   136868 |  56305604538
 Activity  | BgWriterHibernate      |     7486 |  55266949471
 Activity  | BgWriterMain           |     7487 |   1508909964
.
.
```

```
.
 IO        | SLRURead               |        3 |        11756
 IO        | WALWrite               | 52544463 |    388850428
 IO        | XactSync               |   187073 |    597041642
 IO        | ClogRead               |        2 |        47729
 IO        | OutboundCtrlRead       |        1 |          888
 IO        | OutboundCtrlWrite      |        2 |           64
```

## aurora_stat_wait_event

Daftar semua peristiwa tunggu yang didukung untuk Aurora PostgreSQL. Untuk informasi tentang peristiwa tunggu Aurora PostgreSQL, lihat Peristiwa tunggu Amazon Aurora PostgreSQL.

Sintaksis

```
aurora_stat_wait_event()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- type_id – ID dari jenis peristiwa tunggu.
- event_id – ID dari peristiwa tunggu.
- type_name – Nama jenis tunggu
- event_name – Nama peristiwa tunggu

Catatan penggunaan

Untuk melihat nama acara dengan tipe acara, bukan IDs, gunakan fungsi ini bersama dengan fungsi lain seperti `aurora_stat_wait_type` dan`aurora_stat_system_waits`. Nama peristiwa tunggu yang dikembalikan oleh fungsi ini sama dengan yang dikembalikan oleh fungsi `aurora_wait_report`.

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_stat_wait_event`.

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |               event_name
---------+-----------+-----------------------------------------
       1 |  16777216 | <unassigned:0>
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
.
      11 | 184549377 | LsnAllocate
```

Contoh berikut menggabungkan `aurora_stat_wait_type` dan `aurora_stat_wait_event` untuk mengembalikan nama jenis dan nama peristiwa untuk meningkatkan keterbacaan.

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |               event_name
---------+-----------+---------+-----------+-----------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
```

```
       1 | LWLock     |       1 |  16777218 | OidGenLock
       1 | LWLock     |       1 |  16777219 | XidGenLock
       1 | LWLock     |       1 |  16777220 | ProcArrayLock
.
.
.
       3 | Lock       |       3 |  50331648 | relation
       3 | Lock       |       3 |  50331649 | extend
       3 | Lock       |       3 |  50331650 | page
       3 | Lock       |       3 |  50331651 | tuple
.
.
.
      10 | IO         |      10 | 167772214 | TimelineHistorySync
      10 | IO         |      10 | 167772215 | TimelineHistoryWrite
      10 | IO         |      10 | 167772216 | TwophaseFileRead
      10 | IO         |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN        |      11 | 184549376 | LsnDurable
```

## aurora_stat_wait_type

Daftar semua jenis tunggu yang didukung untuk Aurora PostgreSQL.

Sintaksis

```
aurora_stat_wait_type()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- type_id – ID dari jenis peristiwa tunggu.
- type_name – Nama jenis tunggu.

Catatan penggunaan

Untuk melihat nama acara tunggu dengan tipe acara tunggu alih-alih IDs, gunakan fungsi ini bersama dengan fungsi lain seperti `aurora_stat_wait_event` dan`aurora_stat_system_waits`. Nama peristiwa tunggu yang dikembalikan oleh fungsi ini sama dengan yang dikembalikan oleh fungsi `aurora_wait_report`.

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_stat_wait_type`.

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
      10 | IO
      11 | LSN
```

## aurora_version

Mengembalikan nilai string nomor versi Amazon Aurora Edisi Kompatibel PostgreSQL.

Sintaksis

```
aurora_version()
```

Argumen

Tidak ada

Jenis pengembalian

String CHAR atau VARCHAR

Catatan penggunaan

Fungsi ini menampilkan versi mesin basis data Amazon Aurora Edisi Kompatibel PostgreSQL. Nomor versi dikembalikan sebagai string yang diformat sebagai*major*. *minor*. *patch*. Untuk informasi selengkapnya tentang nomor versi Aurora PostgreSQL, lihat Nomor versi Aurora.

Anda dapat memilih kapan harus menerapkan peningkatan versi minor dengan mengatur jendela pemeliharaan untuk klaster DB Aurora PostgreSQL. Untuk mempelajari caranya, lihat Memelihara klaster DB Amazon Aurora.

Mulai dengan rilis Aurora PostgreSQL versi 13.3, 12.8, 11.13, 10.18, dan untuk semua versi lain yang lebih baru, nomor versi Aurora mengikuti nomor versi PostgreSQL. Untuk informasi selengkapnya tentang semua rilis Aurora PostgreSQL, lihat Pembaruan Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

Contoh

Contoh berikut menunjukkan hasil pemanggilan fungsi `aurora_version` pada klaster DB Aurora PostgreSQL yang menjalankan PostgreSQL 12.7, Aurora PostgreSQL rilis 4.2 dan kemudian menjalankan fungsi yang sama pada klaster yang menjalankan Aurora PostgreSQL versi 13.3.

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
```

Contoh ini menunjukkan cara menggunakan fungsi dengan berbagai opsi untuk mendapatkan detail selengkapnya tentang versi Aurora PostgreSQL. Contoh ini memiliki nomor versi Aurora yang berbeda dari nomor versi PostgreSQL.

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)
```

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

Contoh berikutnya ini menggunakan fungsi dengan opsi yang sama pada contoh sebelumnya. Contoh ini memiliki nomor versi Aurora yang berbeda dari nomor versi PostgreSQL.

```
=> SHOW SERVER_VERSION;
server_version
----------------
 13.3

=> SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
```

```
---------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
------------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

## aurora_volume_logical_start_lsn

Mengembalikan nomor urutan log (LSN) yang digunakan untuk mengidentifikasi awal catatan dalam aliran log write-ahead (WAL) logis dari volume klaster Aurora.

Sintaksis

```
aurora_volume_logical_start_lsn()
```

Argumen

Tidak ada

Jenis pengembalian

`pg_lsn`

Catatan penggunaan

Fungsi ini mengidentifikasi awal catatan dalam aliran WAL logis untuk volume klaster Aurora tertentu. Anda dapat menggunakan fungsi ini saat melakukan peningkatan versi mayor menggunakan replikasi logis dan kloning cepat Aurora untuk menentukan LSN tempat snapshot atau klona basis data diambil. Setelah itu, Anda dapat menggunakan replikasi logis untuk terus mengalirkan data baru yang dicatat setelah LSN dan menyinkronkan perubahan dari penerbit ke pelanggan.

Untuk informasi selengkapnya tentang penggunaan replikasi logis untuk meningkatkan versi mayor, lihat Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL.

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.2 dan versi 15 yang lebih tinggi
- 14.3 dan versi 14 yang lebih tinggi
- 13.6 dan versi 13 yang lebih tinggi
- 12.10 dan versi 12 yang lebih tinggi
- 11.15 dan versi 11 yang lebih tinggi
- Versi 10.20 dan versi 10 yang lebih tinggi

Contoh

Anda dapat memperoleh nomor urutan log (LSN) menggunakan kueri berikut:

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

## aurora_wait_report

Fungsi ini menunjukkan aktivitas peristiwa tunggu selama periode waktu tertentu.

Sintaksis

```
aurora_wait_report([time])
```

Argumen

waktu (opsional)

Waktu dalam hitungan detik. Default-nya 10 detik.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- type_name – Nama jenis tunggu
- event_name – Nama peristiwa tunggu
- wait – Jumlah tunggu
- wait_time - Waktu tunggu dalam milidetik
- ms_per_wait - Rata-rata milidetik berdasarkan jumlah sekali tunggu
- waits_per_xact – Rata-rata tunggu berdasarkan jumlah satu transaksi
- ms_per_xact - Rata-rata milidetik berdasarkan jumlah transaksi

Catatan penggunaan

Fungsi ini tersedia pada rilis Aurora PostgreSQL 1.1 kompatibel dengan PostgreSQL 9.6.6 dan versi yang lebih tinggi.

Untuk menggunakan fungsi ini, Anda harus terlebih dahulu membuat ekstensi `aurora_stat_utils` Aurora PostgreSQL, sebagai berikut:

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

Untuk informasi selengkapnya tentang versi ekstensi Aurora PostgreSQL, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

Fungsi ini menghitung peristiwa tunggu level instans dengan membandingkan dua snapshot data statistik dari fungsi aurora_stat_system_waits() dan PostgreSQL Statistics Views pg_stat_database.

Untuk informasi selengkapnya tentang `aurora_stat_system_waits()` dan `pg_stat_database`, lihat The Statistics Collector di dokumentasi PostgreSQL.

Saat dijalankan, fungsi ini mengambil snapshot awal, menunggu jumlah detik yang ditentukan, dan kemudian mengambil snapshot kedua. Fungsi tersebut membandingkan dua snapshot dan mengembalikan perbedaannya. Perbedaan ini mewakili aktivitas instans untuk interval waktu tersebut.

Pada instans penulis, fungsi ini juga menampilkan jumlah transaksi yang dilakukan dan TPS (transaksi per detik). Fungsi ini mengembalikan informasi pada tingkat instans dan mencakup semua basis data pada instans.

Contoh

Contoh ini menunjukkan cara membuat ekstensi aurora_stat_utils agar dapat menggunakan fungsi aurora_wait_report.

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

Contoh ini menunjukkan cara memeriksa laporan tunggu selama 10 detik.

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
 IO        | DataFileExtend    |     1 |      0.02 |       0.018 |           0.03 |
   0.001
```

Contoh ini menunjukkan cara memeriksa laporan tunggu selama 60 detik.

```
=> SELECT *
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
 type_name |       event_name       |  waits  | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+----------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |      30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
 IO        | RelationMapRead        |      72 |      5.16 |       0.072 |
 0.05 |       0.003
 LWLock    | ProcArrayLock          |       1 |      2.01 |       2.008 |
 0.00 |       0.001
```

```
IO        | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
0.00 |       0.000
IO        | DataFileExtend         |       1 |      0.02 |       0.018 |
0.00 |       0.000
IO        | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

## Parameter Amazon Aurora PostgreSQL

Anda mengelola klaster DB Amazon Aurora Anda dengan cara yang sama seperti Anda mengelola instans DB Amazon RDS lain, dengan menggunakan parameter dalam grup parameter DB. Namun, Amazon Aurora berbeda dari Amazon RDS karena klaster DB Aurora memiliki beberapa instans DB. Beberapa parameter yang Anda gunakan untuk mengelola klaster DB Amazon Aurora Anda berlaku untuk seluruh klaster, sementara parameter lain hanya berlaku untuk instans DB tertentu dalam klaster DB, sebagai berikut:

- Grup parameter klaster DB — Grup parameter klaster DB berisi serangkaian parameter konfigurasi engine yang berlaku di seluruh klaster DB Aurora. Misalnya, manajemen cache klaster adalah fitur dari klaster DB Aurora yang dikendalikan oleh parameter `apg_ccm_enabled` yang merupakan bagian dari grup parameter klaster DB. Grup parameter klaster DB juga berisi pengaturan default untuk grup parameter DB untuk instans DB yang membentuk klaster.
- Grup parameter DB — Grup parameter DB adalah kumpulan nilai konfigurasi mesin yang berlaku untuk instans DB tertentu dari jenis mesin tersebut. Grup parameter DB untuk mesin DB PostgreSQL digunakan oleh instans RDS for PostgreSQL dan klaster DB Aurora PostgreSQL. Pengaturan konfigurasi ini berlaku untuk properti yang dapat beragam di antara instans DB dalam klaster Aurora, misalnya ukuran untuk buffer memori.

Anda mengelola parameter tingkat klaster dalam grup parameter klaster DB. Anda mengelola parameter tingkat instans dalam grup parameter klaster DB. Anda dapat mengelola parameter menggunakan konsol Amazon RDS, API Amazon RDS AWS CLI, atau Amazon RDS. Ada perintah terpisah untuk mengelola parameter tingkat klaster dan parameter tingkat instans.

- Untuk mengelola parameter tingkat cluster dalam grup parameter cluster DB, gunakan perintah - group. modify-db-cluster-parameter AWS CLI
- Untuk mengelola parameter tingkat instance dalam grup parameter DB untuk instans DB di cluster DB, gunakan perintah. modify-db-parameter-group AWS CLI

Untuk mempelajari selengkapnya AWS CLI, lihat Menggunakan AWS CLI dalam Panduan AWS Command Line Interface Pengguna.

Untuk informasi selengkapnya tentang grup parameter, lihat .

## Melihat klaster DB dan parameter DB Aurora PostgreSQL

Anda dapat melihat semua grup parameter default yang tersedia untuk instans DB RDS for PostgreSQL dan untuk klaster DB Aurora PostgreSQL di AWS Management Console. Grup parameter default untuk semua mesin DB dan tipe dan versi cluster DB terdaftar untuk setiap AWS Wilayah. Setiap grup parameter kustom juga akan ditampilkan.

Daripada melihat di AWS Management Console, Anda juga dapat mencantumkan parameter yang terdapat dalam grup parameter cluster DB dan grup parameter DB dengan menggunakan AWS CLI atau Amazon RDS API. Misalnya, untuk membuat daftar parameter dalam grup parameter cluster DB Anda menggunakan describe-db-cluster-parameters AWS CLI perintah sebagai berikut:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

Perintah ini akan menghasilkan deskripsi JSON terperinci untuk setiap parameter. Untuk mengurangi jumlah informasi yang dihasilkan, Anda dapat menentukan apa yang Anda inginkan dengan menggunakan opsi `--query`. Misalnya, Anda bisa mendapatkan nama parameter, deskripsinya, dan nilai yang diizinkan untuk grup parameter klaster DB Aurora PostgreSQL 12 default sebagai berikut:

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Untuk Windows:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Grup parameter klaster DB Aurora mencakup grup parameter instans DB dan nilai default untuk mesin Aurora DB tertentu. Anda bisa mendapatkan daftar parameter DB dari grup parameter default

Aurora PostgreSQL default yang sama dengan menggunakan perintah seperti yang ditunjukkan berikut. describe-db-parameters AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Untuk Windows:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Perintah sebelumnya menghasilkan daftar parameter dari klaster DB atau grup parameter DB dengan deskripsi dan detail lainnya yang ditentukan dalam kueri. Berikut adalah respons contohnya:

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

Berikut ini adalah tabel yang berisi nilai untuk parameter klaster DB default dan parameter DB untuk Aurora PostgreSQL versi 14.

## Parameter tingkat klaster Aurora PostgreSQL

Anda dapat melihat parameter tingkat cluster yang tersedia untuk versi PostgreSQL Aurora tertentu menggunakan konsol AWS Manajemen, CLI, atau Amazon RDS API. AWS Untuk informasi tentang cara melihat parameter dalam grup parameter klaster DB Aurora PostgreSQL di konsol RDS, lihat Melihat nilai parameter untuk grup parameter cluster DB di Amazon Aurora.

Beberapa parameter tingkat klaster tidak tersedia di semua versi dan beberapa sedang dihentikan. Untuk informasi tentang cara melihat parameter versi PostgreSQL Aurora tertentu, lihat Melihat klaster DB dan parameter DB Aurora PostgreSQL.

Misalnya, tabel berikut mencantumkan parameter yang tersedia di grup parameter klaster DB default untuk Aurora PostgreSQL versi 14. Jika Anda membuat klaster DB PostgreSQL Aurora tanpa menentukan grup parameter DB kustom Anda sendiri, klaster DB Anda akan dibuat menggunakan grup parameter klaster DB Aurora default untuk versi yang dipilih, seperti `default.aurora-postgresql14`, `default.aurora-postgresql13`, dan sebagainya.

Untuk daftar parameter instans DB untuk grup parameter klaster DB default yang sama ini, lihat Parameter tingkat instans Aurora PostgreSQL.

| Nama parameter | Deskripsi | Default |
|---|---|---|
| ansi_constraint_trigger_ordering | Mengubah urutan pengaktifan pemicu pembatasan agar kompatibel dengan standar ANSI SQL. | – |
| ansi_force_foreign_key_checks | Memastikan tindakan referensial seperti penghapusan kaskade atau pembaruan kaskade akan selalu terjadi terlepas dari berbagai konteks pemicu yang ada untuk tindakan tersebut. | – |
| ansi_qualified_update_set_target | Mendukung pengualifikasi tabel dan skema di UPDATE ... Pernyataan SET. | – |
| apg_ccm_enabled | Mengaktifkan atau menonaktifkan manajemen cache klaster untuk klaster. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| apg_enable_batch_mode_function_execution | Mengaktifkan fungsi mode batch untuk memproses set baris pada suatu waktu. | – |
| apg_enable_correlated_any_transform | Memungkinkan perencana mentransformasikan Subtautan ANY terkorelasi (subkueri IN/NOT IN) menjadi JOIN jika memungkinkan. | – |
| apg_enable_function_migration | Memungkinkan perencana memigrasikan fungsi skalar yang memenuhi syarat ke klausa FROM. | – |
| apg_enable_not_in_transform | Memungkinkan perencana mentransformasikan subkueri NOT IN menjadi ANTI JOIN jika memungkinkan. | – |
| apg_enable_remove_redundant_inner_joins | Memungkinkan perencana menghapus inner join yang redundan. | – |
| apg_enable_semijoin_push_down | Memungkinkan penggunaan filter semijoin untuk hash join. | – |
| apg_plan_mgmt.capture_plan_baselines | Mengambil mode acuan dasar rencana. manual - mengaktifkan pengambilan rencana untuk pernyataan SQL apa pun, off - menonaktifkan pengambilan rencana, automatic - mengaktifkan pengambilan rencana untuk pernyataan dalam pg_stat_statements yang memenuhi kriteria kelayakan. | off |
| apg_plan_mgmt.max_databases | Menetapkan jumlah maksimum basis data yang dapat mengelola kueri menggunakan apg_plan_mgmt. | 10 |
| apg_plan_mgmt.max_plans | Menetapkan jumlah maksimum rencana yang dapat di-cache oleh apg_plan_mgmt. | 10000 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| apg_plan_mgmt.plan_retention_period | Jumlah hari maksimum sejak rencana last_used sebelum rencana akan dihapus secara otomatis. | 32 |
| apg_plan_mgmt.unapproved_plan_execution_threshold | Perkiraan total biaya rencana yang jika tidak tercapai akan membuat rencana yang Tidak Disetujui dieksekusi. | 0 |
| apg_plan_mgmt.use_plan_baselines | Menggunakan hanya rencana yang disetujui atau tetap untuk pernyataan terkelola. | false |
| application_name | Menetapkan nama aplikasi yang akan dilaporkan dalam statistik dan log. | – |
| array_nulls | Mengaktifkan input elemen NULL dalam array. | – |
| aurora_compute_plan_id | Memantau rencana eksekusi kueri untuk mendeteksi rencana eksekusi yang berkontribusi pada pemuatan basis data saat ini dan untuk melacak statistik kinerja rencana eksekusi dari waktu ke waktu. Untuk informasi selengkapnya, lihat Memantau rencana eksekusi kueri untuk Aurora PostgreSQL. | on |
| authentication_timeout | (s) Menetapkan waktu maksimum yang diizinkan untuk menyelesaikan autentikasi klien. | – |
| auto_explain.log_analyze | Menggunakan EXPLAIN ANALYZE untuk pencatatan log rencana. | – |
| auto_explain.log_buffers | Mencatat log penggunaan buffer. | – |
| auto_explain.log_format | Menggunakan EXPLAIN untuk format yang akan digunakan untuk pencatatan log rencana. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| auto_explain.log_min_duration | Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat rencana dicatat. | – |
| auto_explain.log_nested_statements | Mencatat log pernyataan bersarang. | – |
| auto_explain.log_timing | Mengumpulkan data waktu, bukan hanya hitungan baris. | – |
| auto_explain.log_triggers | Menyertakan statistik pemicu dalam rencana. | – |
| auto_explain.log_verbose | Menggunakan EXPLORE VERBOSE untuk pencatatan log rencana. | – |
| auto_explain.sample_rate | Pecahan kueri untuk diproses. | – |
| autovacuum | Memulai subproses autovacuum. | – |
| autovacuum_analyze_scale_factor | Jumlah penyisipan, pembaruan, atau penghapusan tuple sebelum analisis sebagai pecahan dari reltuple. | 0,05 |
| autovacuum_analyze_threshold | Jumlah minimum penyisipan tuple, pembaruan atau penghapusan sebelum analisis. | – |
| autovacuum_freeze_max_age | Usia saat tabel akan di-autovacuum untuk mencegah wraparound ID transaksi. | – |
| autovacuum_max_workers | Menetapkan jumlah maksimum proses pekerja autovacuum yang berjalan secara bersamaan. | TERBESAR (DBInstanceClassMemory/6437 1566592 ,3) |
| autovacuum_multixact_freeze_max_age | Usia multixact saat tabel perlu di-autovacuum untuk mencegah wraparound multixact. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| autovacuum_naptime | (s) Waktu untuk tidur di antara pelaksanaan autovacuum. | 5 |
| autovacuum_vacuum_cost_delay | (ms) Penundaan biaya vacuum dalam milidetik, untuk autovacuum. | 5 |
| autovacuum_vacuum_cost_limit | Jumlah biaya vacuum yang tersedia sebelum napping, untuk autovacuum. | TERBESAR (log (DBInstanceClassMemory/21474836480) * 600,200) |
| autovacuum_vacuum_insert_scale_factor | Jumlah penyisipan tuple sebelum vacuum sebagai pecahan dari reltuple. | – |
| autovacuum_vacuum_insert_threshold | Jumlah minimum penyisipan tuple sebelum vacuum, atau -1 untuk menonaktifkan penyisipan vacuum. | – |
| autovacuum_vacuum_scale_factor | Jumlah pembaruan atau penghapusan tuple sebelum vacuum sebagai pecahan dari reltuple. | 0.1 |
| autovacuum_vacuum_threshold | Jumlah minimum pembaruan atau penghapusan tuple sebelum vacuum. | – |
| autovacuum_work_mem | (kB) Menetapkan memori maksimum yang akan digunakan oleh setiap proses pekerja autovacuum. | TERBESAR (DBInstanceClassMemory/32768 ,131072) |
| babelfishpg_tds.default_server_name | Nama server Babelfish default | Microsoft SQL Server |
| babelfishpg_tds.listen_addresses | Menetapkan nama host atau alamat IP untuk mendengarkan TDS. | * |
| babelfishpg_tds.port | Menetapkan port TCP TDS yang didengarkan server. | 1433 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| babelfishpg_tds.tds_debug_log_level | Menetapkan tingkat pencatatan log di TDS, 0 menonaktifkan pencatatan log | 1 |
| babelfishpg_tds.tds_default_numeric_precision | Menetapkan presisi default jenis numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya. | 38 |
| babelfishpg_tds.tds_default_numeric_scale | Menetapkan skala default jenis numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya. | 8 |
| babelfishpg_tds.tds_default_packet_size | Integer yang menetapkan ukuran paket default untuk semua klien SQL Server yang terhubung. | 4096 |
| babelfishpg_tds.tds_default_protocol_version | Integer yang menetapkan versi protokol TDS default untuk semua klien yang terhubung. | DEFAULT |
| babelfishpg_tds.tds_ssl_encrypt | Menetapkan opsi Enkripsi SSL | 0 |
| babelfishpg_tds.tds_ssl_max_protocol_version | Menetapkan versi protokol SSL/TLS maksimum yang akan digunakan untuk sesi tds. | TLSv1.2 |
| babelfishpg_tds.tds_ssl_min_protocol_version | Menetapkan versi protokol SSL/TLS minimum yang akan digunakan untuk sesi tds. | TLSv1.2 dari Aurora PostgreSQL versi 16, untuk versi yang lebih tua dari Aurora PostgreSQL versi TLSv1 16 |
| babelfishpg_tsql.default_locale | Lokalitas default yang akan digunakan untuk kolasi yang dibuat oleh CREATE COLLATION. | en-US |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| babelfishpg_tsql.migration_mode | Mendefinisikan apakah lebih dari satu basis data pengguna dapat didukung | multi-db dari Aurora PostgreSQL versi 16, single-db untuk versi yang lebih tua dari Aurora PostgreSQL versi 16 |
| babelfishpg_tsql.server_collation_name | Nama kolasi server default | sql_latin1_general_cp1_ci_as |
| babelfishpg_tsql.version | Menetapkan output dari variabel @@VERSION | default |
| backend_flush_after | (8Kb) Jumlah halaman yang jika terpenuhi akan membuat penulisan yang dilakukan sebelumnya dibuang ke disk. | – |
| backslash_quote | Menetapkan apakah \\ diizinkan dalam string literal. | – |
| backtrace_functions | Mencatat log pelacakan mundur untuk kesalahan dalam fungsi ini. | – |
| bytea_output | Menetapkan format output untuk bytea. | – |
| check_function_bodies | Memeriksa konten fungsi selama CREATE FUNCTION. | – |
| client_connection_check_interval | Menetapkan interval waktu di antara pemeriksaan pemutusan koneksi saat menjalankan kueri. | – |
| client_encoding | Menetapkan pengenkodean set karakter klien. | UTF8 |
| client_min_messages | Menetapkan tingkat pesan yang dikirimkan ke klien. | – |
| compute_query_id | Menghitung pengidentifikasi kueri. | auto |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| config_file | Menetapkan file konfigurasi utama server. | /rdsdbdata/config/postgresql.conf |
| constraint_exclusion | Memungkinkan perencana menggunakan pembatasan untuk mengoptimalkan kueri. | – |
| cpu_index_tuple_cost | Menetapkan perkiraan perencana untuk biaya pemrosesan setiap entri indeks selama pemindaian indeks. | – |
| cpu_operator_cost | Menetapkan perkiraan perencana untuk biaya pemrosesan setiap panggilan operator atau fungsi. | – |
| cpu_tuple_cost | Menetapkan perkiraan perencana untuk biaya pemrosesan setiap tuple (baris). | – |
| cron.database_name | Menetapkan basis data untuk menyimpan tabel metadata pg_cron | postgres |
| cron.log_run | Mencatat log semua pelaksanaan pekerjaan ke tabel job_run_details | on |
| cron.log_statement | Mencatat log semua pernyataan cron sebelum eksekusi. | off |
| cron.max_running_jobs | Jumlah maksimum pekerjaan yang dapat dijalankan secara bersamaan. | 5 |
| cron.use_background_workers | Mengaktifkan pekerja latar belakang untuk pg_cron | on |
| cursor_tuple_fraction | Menetapkan perkiraan perencana untuk pecahan dari baris kursor yang akan diambil. | – |
| data_directory | Menetapkan direktori data server. | /rdsdbdata/db |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| datestyle | Menetapkan format tampilan untuk nilai tanggal dan waktu. | – |
| db_user_namespace | Mengaktifkan nama pengguna per basis data. | – |
| deadlock_timeout | (ms) Menetapkan waktu untuk menunggu kunci sebelum memeriksa deadlock. | – |
| debug_pretty_print | Mengindentasi tampilan penguraian dan hierarki rencana. | – |
| debug_print_parse | Mencatat log hierarki penguraian setiap kueri. | – |
| debug_print_plan | Mencatat log rencana eksekusi setiap kueri. | – |
| debug_print_rewritten | Mencatat log hierarki penguraian yang ditulis ulang masing-masing kueri. | – |
| default_statistics_target | Menetapkan target statistik default. | – |
| default_tablespace | Menetapkan ruang tabel default untuk membuat tabel dan indeks. | – |
| default_toast_compression | Menetapkan metode kompresi default untuk nilai kompresibel. | – |
| default_transaction_deferrable | Menetapkan status default yang dapat ditangguhkan untuk transaksi baru. | – |
| default_transaction_isolation | Menetapkan tingkat isolasi transaksi untuk setiap transaksi baru. | – |
| default_transaction_read_only | Menetapkan status hanya baca default untuk transaksi baru. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| effective_cache_size | (8kB) Menetapkan asumsi perencana tentang ukuran cache disk. | JUMLAH (DBInstanceClassMemory/12038, -50003) |
| effective_io_concurrency | Jumlah permintaan serentak yang dapat ditangani secara efisien oleh subsistem disk. | – |
| enable_async_append | Memungkinkan perencana menggunakan rencana async append. | – |
| enable_bitmapscan | Memungkinkan perencana menggunakan rencana bitmap-scan. | – |
| enable_gathermerge | Memungkinkan perencana menggunakan rencana gather merge. | – |
| enable_hashagg | Memungkinkan perencana menggunakan rencana hashed aggregation. | – |
| enable_hashjoin | Memungkinkan perencana menggunakan rencana hash join. | – |
| enable_incremental_sort | Memungkinkan perencana menggunakan langkah-langkah pengurutan inkremental. | – |
| enable_indexonlyscan | Memungkinkan perencana menggunakan index-only-scan rencana. | – |
| enable_indexscan | Memungkinkan perencana menggunakan rencana index-scan. | – |
| enable_material | Memungkinkan perencana menggunakan materialization. | – |
| aktifkan_memoize | Memungkinkan perencana menggunakan memoization | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| enable_mergejoin | Memungkinkan perencana menggunakan rencana merge join. | – |
| enable_nestloop | Memungkinkan perencana menggunakan rencana nested-loop join. | – |
| enable_parallel_append | Mengaktifkan penggunaan perencana atas rencana parallel append. | – |
| aktifkan_parallel_hash | Mengaktifkan penggunaan perencana atas rencana parallel hash. | – |
| enable_partition_pruning | Mengaktifkan pemangkasan partisi plan-time dan run-time. | – |
| enable_partitionwise_aggregate | Mengaktifkan agregasi dan pengelompokan partitionwise. | – |
| enable_partitionwise_join | Mengaktifkan partitionwise join. | – |
| enable_seqscan | Memungkinkan perencana menggunakan rencana sequential-scan. | – |
| enable_sort | Memungkinkan perencana menggunakan langkah-langkah penyortiran eksplisit. | – |
| enable_tidscan | Memungkinkan perencana menggunakan rencana TID scan. | – |
| escape_string_warning | Memperingatkan tentang escape garis miring dalam string literal biasa. | – |
| exit_on_error | Mengakhiri sesi saat terjadi kesalahan apa pun. | – |
| extra_float_digits | Menetapkan jumlah digit yang ditampilkan untuk nilai floating-point. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| force_parallel_mode | Memaksa penggunaan fasilitas kueri paralel. | – |
| from_collapse_limit | Menetapkan ukuran FROM-list yang jika terlampaui akan membuat subkueri tidak ditutup. | – |
| geqo | Memungkinkan optimisasi kueri genetik. | – |
| geqo_effort | GEQO: upaya digunakan untuk mengatur nilai default untuk parameter GEQO lainnya. | – |
| geqo_generations | GEQO: jumlah iterasi algoritma. | – |
| geqo_pool_size | GEQO: jumlah individu dalam populasi. | – |
| geqo_seed | GEQO: seed untuk pemilihan jalur acak. | – |
| geqo_selection_bias | GEQO: tekanan selektif di dalam populasi. | – |
| geqo_threshold | Menetapkan ambang batas item FROM yang jika terlampaui akan membuat GEQO digunakan. | – |
| gin_fuzzy_search_limit | Menetapkan hasil maksimum yang diizinkan untuk pencarian persis oleh GIN. | – |
| gin_pending_list_limit | (kB) Menetapkan ukuran maksimum daftar tertunda untuk indeks GIN. | – |
| hash_mem_multiplier | Beberapa work_mem yang akan digunakan untuk tabel hash. | – |
| hba_file | Menetapkan file konfigurasi hba server. | /rdsdbdata/config/pg_hba.conf |
| hot_standby_feedback | Memungkinkan umpan balik dari hot standby ke primer yang akan menghindari konflik kueri. | on |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| huge_pages | Mengurangi overhead saat instans DB menangani potongan besar memori yang berdekatan, seperti yang digunakan oleh buffer bersama. Parameter ini diaktifkan secara default untuk semua kelas instans DB selain kelas instans t3.medium, db.t3.large, db.t4g.medium, db.t4g.large. | on |
| ident_file | Menetapkan file konfigurasi ident server. | /rdsdbdata/config/pg_ident.conf |
| idle_in_transaction_session_timeout | (ms) Menetapkan durasi maksimum yang diizinkan untuk setiap transaksi idling. | 86400000 |
| idle_session_timeout | Mengakhiri setiap sesi yang telah idle (yaitu, menunggu kueri klien), tetapi tidak dalam transaksi terbuka, lebih lama dari jumlah waktu yang ditentukan | – |
| intervalstyle | Menetapkan format tampilan untuk nilai interval. | – |
| join_collapse_limit | Menetapkan ukuran FROM-list yang jika terlampaui akan membuat konstruk JOIN tidak diratakan. | – |
| krb_caseins_users | Menetapkan apakah nama pengguna GSSAPI (Generic Security Service API) harus diperlakukan tanpa peka huruf besar/kecil (true) atau tidak. Secara default, parameter ini disetel ke false, sehingga Kerberos mengharapkan nama pengguna yang peka huruf besar/kecil. Untuk informasi selengkapnya, lihat GSSAPI Authentication dalam dokumentasi PostgreSQL. | false |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| lc_messages | Menetapkan dalam bahasa apa pesan akan ditampilkan. | – |
| lc_monetary | Menetapkan lokalitas untuk memformat jumlah uang. | – |
| lc_numeric | Menetapkan lokalitas untuk memformat angka. | – |
| lc_time | Menetapkan lokalitas untuk memformat nilai tanggal dan waktu. | – |
| listen_addresses | Menetapkan nama host atau alamat IP untuk didengarkan. | * |
| lo_compat_privileges | Mengaktifkan mode kompatibilitas mundur untuk pemeriksaan hak akses pada objek besar. | 0 |
| log_autovacuum_min_duration | (ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat tindakan autovacuum dicatat. | 10000 |
| log_connections | Mencatat log setiap koneksi yang berhasil. | – |
| log_destination | Menetapkan tujuan untuk output log server. | stderr |
| log_directory | Menetapkan direktori tujuan untuk file log. | /rdsdbdata/log/error |
| log_disconnections | Mencatat log akhir sebuah sesi, termasuk durasinya. | – |
| log_duration | Mencatat log durasi setiap pernyataan SQL yang diselesaikan. | – |
| log_error_verbosity | Menetapkan verbositas pesan yang dicatat. | – |
| log_executor_stats | Menulis statistik performa pengeksekusi ke log server. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| log_file_mode | Menetapkan izin file untuk file log. | 0644 |
| log_filename | Menetapkan pola nama file untuk file log. | postgresql.log.%Y-%m-%d-%H%M |
| logging_collector | Mulai subproses untuk menangkap output stderr dan/atau csvlog ke dalam file log. | 1 |
| log_hostname | Mencatat log nama host dalam log koneksi. | 0 |
| logical_decoding_work_mem | (kB) Memori sebanyak ini dapat digunakan oleh setiap buffer penyusun ulang internal sebelum dialirkan ke disk. | – |
| log_line_prefix | Mengontrol informasi yang ditambahkan ke awal setiap baris log. | %t:%r:%u@%d:%p]: |
| log_lock_waits | Mencatat log waktu tunggu kunci yang panjang. | – |
| log_min_duration_sample | (ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat sampel pernyataan dicatat. Pengambilan sampel ditentukan oleh log_statement_sample_rate. | – |
| log_min_duration_statement | (ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat pernyataan dicatat. | – |
| log_min_error_statement | Menyebabkan semua pernyataan yang menghasilkan kesalahan pada atau di atas tingkat ini dicatat. | – |
| log_min_messages | Menetapkan tingkat pesan yang dicatat. | – |
| log_parameter_max_length | (B) Saat mencatat pernyataan, membatasi nilai parameter yang dicatat ke N byte pertama. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| log_parameter_max_length_on_error | (B) Saat melaporkan kesalahan, membatasi nilai parameter yang dicatat ke N byte pertama. | – |
| log_parser_stats | Menulis statistik performa pengurai ke log server. | – |
| log_planner_stats | Menulis statistik performa perencana ke log server. | – |
| log_replication_commands | Mencatat log setiap perintah replikasi. | – |
| log_rotation_age | (min) Rotasi file log otomatis akan terjadi setelah N menit. | 60 |
| log_rotation_size | (kB) Rotasi file log otomatis akan terjadi setelah N kilobyte. | 100000 |
| log_statement | Menetapkan jenis pernyataan yang dicatat. | – |
| log_statement_sample_rate | Pecahan dari pernyataan yang melebihi log_min_duration_sample yang akan dicatat. | – |
| log_statement_stats | Menulis statistik performa kumulatif ke log server. | – |
| log_temp_files | (kB) Mencatat log penggunaan file sementara yang lebih besar dari jumlah kilobyte ini. | – |
| log_timezone | Menetapkan zona waktu yang akan digunakan dalam pesan log. | UTC |
| log_transaction_sample_rate | Menetapkan pecahan transaksi yang akan dicatat untuk transaksi baru. | – |
| log_truncate_on_rotation | Memotong file log yang ada dengan nama yang sama selama rotasi log. | 0 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| maintenance_io_concurrency | Varian effective_io_concurrency yang digunakan untuk pekerjaan pemeliharaan. | 1 |
| maintenance_work_mem | (kB) Menetapkan memori maksimum yang akan digunakan untuk operasi pemeliharaan. | TERBESAR (DBInstanceClassMemory/63963136 *1024.65536) |
| max_connections | Menetapkan jumlah maksimum koneksi serentak. | PALING SEDIKIT (DBInstanceClassMemory/9531392 ,5000) |
| max_files_per_process | Menetapkan jumlah maksimum file yang terbuka secara bersamaan untuk setiap proses server. | – |
| max_locks_per_transaction | Menetapkan jumlah maksimum kunci per transaksi. | 64 |
| max_logical_replication_workers | Jumlah maksimum proses pekerja replikasi logis. | – |
| max_parallel_maintenance_workers | Menetapkan jumlah maksimum proses paralel per operasi pemeliharaan. | – |
| max_parallel_workers | Menetapkan jumlah maksimum pekerja paralel yang dapat aktif pada satu waktu. | TERBESAR ($ DBInstance VCPU/2,8) |
| max_parallel_workers_per_gather | Menetapkan jumlah maksimum proses paralel per simpul eksekutor. | – |
| max_pred_locks_per_page | Menetapkan jumlah maksimum tuple predicate-lock per halaman. | – |
| max_pred_locks_per_relation | Menetapkan jumlah maksimum halaman dan tuple predicate-lock per relasi. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| max_pred_locks_per_transaction | Menetapkan jumlah maksimum kunci predikat per transaksi. | – |
| max_prepared_transactions | Menetapkan jumlah maksimum transaksi yang disiapkan secara bersamaan. | 0 |
| max_replication_slots | Menetapkan jumlah maksimum slot replikasi yang dapat didukung oleh server. | 20 |
| max_slot_wal_keep_size | (MB) Slot replikasi akan ditandai sebagai gagal, dan segmen akan dirilis untuk dihapus atau didaur ulang, jika ruang sebanyak ini ditempati oleh WAL pada disk. | – |
| max_stack_depth | (kB) Menetapkan kedalaman tumpukan maksimum, dalam kilobyte. | 6144 |
| max_standby_streaming_delay | (ms) Menetapkan penundaan maksimum sebelum membatalkan kueri saat hot standby sedang memproses data WAL yang dialirkan. | 14000 |
| max_sync_workers_per_subscription | Jumlah maksimum pekerja sinkronisasi per langganan | 2 |
| max_wal_senders | Menetapkan jumlah maksimum proses pengirim WAL yang berjalan secara bersamaan. | 10 |
| max_worker_processes | Menetapkan jumlah maksimum proses pekerja serentak. | TERBESAR ($DBInstance VCPU*2,8) |
| min_dynamic_shared_memory | (MB) Jumlah memori bersama dinamis yang dicadangkan saat pengaktifan. | – |
| min_parallel_index_scan_size | (8kB) Menetapkan jumlah minimum data indeks untuk pemindaian paralel. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| min_parallel_table_scan_size | (8kB) Menetapkan jumlah minimum data tabel untuk pemindaian paralel. | – |
| old_snapshot_threshold | (min) Waktu sebelum snapshot terlalu lama untuk membaca halaman yang diubah setelah snapshot diambil. | – |
| orafce.nls_date_format | Mengemulasi perilaku output tanggal Oracle. | – |
| orafce.timezone | Menentukan zona waktu yang digunakan untuk fungsi sysdate. | – |
| parallel_leader_participation | Mengontrol apakah Gather and Gather Merge juga menjalankan subrencana. | – |
| parallel_setup_cost | Menetapkan perkiraan perencana terkait biaya untuk memulai proses pekerja untuk kueri paralel. | – |
| parallel_tuple_cost | Menetapkan perkiraan perencana untuk biaya pengiriman setiap tuple (baris) dari pekerja ke backend master. | – |
| password_encryption | Mengenkripsi kata sandi. | – |
| pgaudit.log | Menentukan kelas pernyataan mana yang akan dicatat oleh pencatatan log audit sesi. | – |
| pgaudit.log_catalog | Menentukan bahwa pencatatan log sesi harus diaktifkan jika semua relasi dalam pernyataan berada di pg_catalog. | – |
| pgaudit.log_level | Menentukan tingkat log yang akan digunakan untuk entri log. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pgaudit.log_parameter | Menentukan bahwa audit pencatatan log harus mencakup parameter yang diteruskan dengan pernyataan. | – |
| pgaudit.log_relation | Menentukan apakah pencatatan log audit sesi harus membuat entri log terpisah untuk setiap relasi (TABLE, VIEW, dll.) yang direferensikan dalam pernyataan SELECT atau DML. | – |
| pgaudit.log_statement_once | Menentukan apakah pencatatan log akan mencakup teks pernyataan dan parameter dengan entri log pertama untuk kombinasi pernyataan/subpernyataan atau dengan setiap entri. | – |
| pgaudit.role | Menentukan peran master yang akan digunakan untuk pencatatan log audit objek. | – |
| pg_bigm.enable_recheck | Menentukan apakah akan melakukan Pemeriksaan Ulang yang merupakan proses internal pencarian teks lengkap. | on |
| pg_bigm.gin_key_limit | Menentukan jumlah maksimum 2 gram kata kunci pencarian yang akan digunakan untuk pencarian teks lengkap. | 0 |
| pg_bigm.last_update | Melaporkan tanggal pembaruan terakhir modul pg_bigm. | 2013.11.22 |
| pg_bigm.similarity_limit | Menentukan ambang batas minimum yang digunakan oleh pencarian kesamaan. | 0,3 |
| pg_hint_plan.debug_print | Mencatat log hasil dari penguraian petunjuk. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_hint_plan.enable_hint | Memaksa perencana untuk menggunakan rencana yang ditentukan dalam komentar petunjuk sebelum kueri. | – |
| pg_hint_plan.enable_hint_table | Memaksa perencana untuk tidak mendapatkan petunjuk dengan menggunakan pencarian tabel. | – |
| pg_hint_plan.message_level | Tingkat pesan untuk pesan debug. | – |
| pg_hint_plan.parse_messages | Tingkat pesan kesalahan penguraian. | – |
| pglogical.batch_inserts | Melakukan penyisipan batch jika memungkinkan | – |
| pglogical.conflict_log_level | Menetapkan tingkat log yang digunakan untuk mencatat konflik yang diselesaikan. | – |
| pglogical.conflict_resolution | Menetapkan metode yang digunakan untuk penyelesaian konflik untuk konflik yang dapat diselesaikan. | – |
| pglogical.extra_connection_options | Opsi koneksi untuk ditambahkan ke semua koneksi simpul peer | – |
| pglogical.synchronous_commit | nilai komit sinkron spesifik pglogical | – |
| pglogical.use_spi | Menggunakan SPI alih-alih API tingkat rendah untuk menerapkan perubahan | – |
| pgtle.clientauth_databases_to_skip | Daftar database untuk dilewati untuk fitur clientauth. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pgtle.clientauth_db_name | Mengontrol database mana yang digunakan untuk fitur clientauth. | – |
| pgtle.clientauth_num_parallel_workers | Jumlah pekerja latar belakang yang digunakan untuk fitur clientauth. | – |
| pgtle.clientauth_users_to_skip | Daftar pengguna yang akan dilewati untuk fitur clientauth. | – |
| pgtle.enable_clientauth | Mengaktifkan fitur clientauth. | – |
| pgtle.passcheck_db_name | Menetapkan database mana yang digunakan untuk fitur passcheck di seluruh cluster. | – |
| pg_prewarm.autoprewarm | Memulai pekerja autoprewarm. | – |
| pg_prewarm.autoprewarm_interval | Menetapkan interval di antara dump buffer bersama | – |
| pg_similarity.block_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.block_threshold | Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Blok. | – |
| pg_similarity.block_tokenizer | Menetapkan tokenizer untuk fungsi kesamaan Blok. | – |
| pg_similarity.cosine_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.cosine_threshold | Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Kosinus. | – |
| pg_similarity.cosine_tokenizer | Menetapkan pentokenisasi untuk fungsi kesamaan Kosinus. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_similarity.dice_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.dice_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Dice. | – |
| pg_similarity.dice_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Dice. | – |
| pg_similarity.euclidean_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.euclidean_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Euclidean. | – |
| pg_similarity.euclidean_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Euclidean. | – |
| pg_similarity.hamming_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.hamming_threshold | Menetapkan ambang batas yang digunakan oleh metrik kesamaan Blok. | – |
| pg_similarity.jaccard_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.jaccard_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaccard. | – |
| pg_similarity.jaccard_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Jaccard. | – |
| pg_similarity.jaro_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.jaro_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaro. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_similarity.jaro winkler_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.jaro winkler_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jarowinkler. | – |
| pg_similarity.leve nshtein_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.leve nshtein_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Levenshtein. | – |
| pg_similarity.matc hing_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.matc hing_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Pencocokan. | – |
| pg_similarity.matc hing_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Pencocokan. | – |
| pg_similarity.mong eelkan_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.mong eelkan_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Monge-Elkan. | – |
| pg_similarity.mong eelkan_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Monge-Elkan. | – |
| pg_similarity.nw_g ap_penalty | Menetapkan penalti kesenjangan yang digunakan oleh ukuran kesamaan Needleman-Wunsch. | – |
| pg_similarity.nw_i s_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_similarity.nw_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Needleman-Wunsch. | – |
| pg_similarity.overlap_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.overlap_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Tumpang-Tindih. | – |
| pg_similarity.overlap_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Tumpang-Tindih. | – |
| pg_similarity.qgram_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.qgram_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Q-Gram. | – |
| pg_similarity.qgram_tokenizer | Menetapkan pentokenisasi untuk ukuran Q-Gram. | – |
| pg_similarity.swg_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.swg_threshold | Menetapkan ambang batas yang digunakan oleh ukuran Smith-Waterman-Gotoh kesamaan. | – |
| pg_similarity.sw_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.sw_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Smith-Waterman. | – |
| pg_stat_statements.max | Menetapkan jumlah maksimum pernyataan yang dilacak oleh pg_stat_statements. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_stat_statements.save | Save pg_stat_statements statistics across server shutdowns. | – |
| pg_stat_statements.track | Memilih pernyataan mana yang dilacak oleh pg_stat_statements. | – |
| pg_stat_statements.track_planning | Memilih apakah durasi perencanaan dilacak oleh pg_stat_statements. | – |
| pg_stat_statements.track_utility | Memilih apakah perintah utilitas dilacak oleh pg_stat_statements. | – |
| plan_cache_mode | Mengontrol pemilihan perencana untuk rencana kustom atau generik. | – |
| port | Menetapkan port TCP yang didengarkan server. | EndPointPort |
| postgis.gdal_enabled_drivers | Mengaktifkan atau menonaktifkan driver GDAL yang digunakan dengan PostGIS di Postgres 9.3.5 dan versi lebih tinggi. | ENABLE_ALL |
| quote_all_identifiers | Saat membuat fragmen SQL, mengutip semua pengidentifikasi. | – |
| random_page_cost | Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara tidak berurutan. | – |
| rdkit.dice_threshold | Ambang batas bawah kesamaan Dice. Molekul dengan kesamaan lebih rendah dari ambang batas tidak sama berdasarkan operasi #. | – |
| rdkit.do_chiral_sss | Apakah stereokimia harus diperhitungkan dalam pencocokan substruktur. Jika salah, tidak ada informasi stereokimia yang digunakan dalam kecocokan substruktur. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| rdkit.tanimoto_threshold | Ambang batas bawah kesamaan Tanimoto. Molekul dengan kesamaan lebih rendah dari ambang batas tidak sama berdasarkan operasi %. | – |
| rds.accepted_password_auth_method | Memaksa autentikasi untuk koneksi dengan kata sandi yang disimpan secara lokal. | md5+scram |
| rds.adaptive_autovacuum | Parameter RDS untuk mengaktifkan/menonaktifkan autovacuum adaptif. | 1 |
| rds.babelfish_status | Parameter RDS untuk mengaktifkan/menonaktifkan Babelfish for Aurora PostgreSQL. | off |
| rds.enable_plan_management | Mengaktifkan atau menonaktifkan ekstensi apg_plan_mgmt. | 0 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| rds.extensions | Daftar ekstensi yang disediakan oleh RDS | address_standardizer, address_standardizer_data_us, apg_plan_mgmt, aurora_stat_utils, amcheck, autoinc, aws_commons, aws_ml, aws_s3, aws_lambda, bool_plperl, bloom, btree_gin, btree_gist, citext, cube, dblink, dict_int, dict_xsyn, earthdistance, fuzzystrmatch, hll, hstore, hstore_plperl, insert_username, intagg, intarray, ip4r, isn, jsonb_plperl, lo, log_fdw, ltree, moddatetime, old_snapshot, oracle_fdw, orafce, pgaudit, pgcrypto, pglogical, pgrouting, pgrowlocks, pgstattuple, pgtap, pg_bigm, pg_buffercache, pg_cron, pg_freespacemap, pg_hint_plan, pg_partman, pg_prewarm, pg_proctab, pg_repack, pg_simila |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| | | rity, pg_stat_s tatements, pg_trgm, pg_visibility, plcoffee, plls, plperl, plpgsql, plprofiler, pltcl, plv8, postgis, postgis_t iger_geocoder, postgis_raster, postgis_topology, postgres_fdw, prefix, rdkit, rds_tools, refint, sslinfo, tablefunc, tds_fdw, test_parser, tsm_system_rows, tsm_system_time, unaccent, uuid-ossp |
| rds.force_admin_lo gging_level | Melihat pesan log untuk tindakan pengguna admin RDS di basis data pelanggan. | – |
| rds.force_autovacu um_logging_level | Melihat pesan log yang terkait dengan operasi autovacuum. | WARNING |
| rds.force_ssl | Memaksa koneksi SSL. | 0 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| rds.global_db_rpo | (s) Ambang batas sasaran titik pemulihan, dalam hitungan detik, yang memblokir komit pengguna jika ambang batas ini dilanggar.<br><br>**Important**<br>Parameter ini dimaksudkan untuk basis data global berbasis Aurora PostgreSQL. Untuk basis data nonglobal, biarkan pada nilai default. Untuk informasi selengkapnya tentang penggunaan parameter ini, lihat the section called “Mengelola RPOs (Aurora PostgreSQL)”. | – |
| rds.logical_replication | Mengaktifkan pendekodean logis. | 0 |
| rds.logically_replicate_unlogged_tables | Tabel yang tidak dicatat direplikasi secara logis. | 1 |
| rds.log_retention_period | Amazon RDS akan menghapus log PostgreSQL yang lebih lama dari N menit. | 4320 |
| rds.pg_stat_ramdisk_size | Ukuran statistik ramdisk dalam MB. Nilai bukan nol akan mengatur ramdisk. Parameter ini hanya tersedia di Aurora PostgreSQL 14 dan versi lebih rendah. | 0 |
| rds.rds_superuser_reserved_connections | Menetapkan jumlah slot koneksi yang disediakan untuk rds_superusers. Parameter ini hanya tersedia dalam versi 15 dan sebelumnya. Untuk informasi selengkapnya, lihat koneksi cadangan dokumentasi PostgreSQL. | 2 |
| rds.restrict_password_commands | Membatasi perintah terkait kata sandi untuk anggota rds_password | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| rds.superuser_variables | Daftar variabel superuser-only yang pernyataan modifikasi rds_superuser-nya dielevasikan. | session_replication_role |
| recovery_init_sync_method | Menetapkan metode untuk menyinkronkan direktori data sebelum pemulihan crash. | sinkronisasi |
| remove_temp_files_after_crash | Menghapus file sementara setelah backend crash. | 0 |
| restart_after_crash | Menginisialisasi ulang server setelah backend crash. | – |
| row_security | Mengaktifkan keamanan baris. | – |
| search_path | Menetapkan urutan pencarian skema untuk nama yang tidak memenuhi syarat skema. | – |
| seq_page_cost | Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara berurutan. | – |
| session_replication_role | Menetapkan perilaku sesi untuk pemicu dan aturan penulisan ulang. | – |
| shared_buffers | (8kB) Menetapkan jumlah buffer memori bersama yang digunakan oleh server. | JUMLAH (DBInstanceClassMemory/12038, -50003) |
| shared_preload_libraries | Menampilkan daftar pustaka bersama untuk dimuat sebelumnya ke server. | pg_stat_statements |
| ssl | Mengaktifkan koneksi SSL. | 1 |
| ssl_ca_file | Lokasi file otoritas server SSL. | /rdsdbdata/rds-metadata/ca-cert.pem |
| ssl_cert_file | Lokasi file sertifikat server SSL. | /rdsdbdata/rds-metadata/server-cert.pem |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| ssl_ciphers | Menetapkan daftar cipher TLS yang diizinkan untuk digunakan pada koneksi aman. | – |
| ssl_crl_dir | Lokasi direktori daftar pencabutan sertifikat SSL. | /rdsdbdata/rds-metadata/ssl_crl_dir/ |
| ssl_key_file | Lokasi file kunci privat server SSL | /rdsdbdata/rds-metadata/server-key.pem |
| ssl_max_protocol_version | Menetapkan versi protokol SSL/TLS maksimum yang diizinkan | – |
| ssl_min_protocol_version | Menetapkan versi protokol SSL/TLS minimum yang diizinkan | TLSv1.2 |
| standard_conforming_strings | Menyebabkan string ... memperlakukan garis miring secara literal. | – |
| statement_timeout | (ms) Menetapkan durasi maksimum yang diizinkan untuk setiap pernyataan. | – |
| stats_temp_directory | Menulis file statistik sementara ke direktori yang ditentukan. | /rdsdbdata/db/pg_stat_tmp |
| superuser_reserved_connections | Menetapkan jumlah slot koneksi yang disediakan untuk pengguna super. | 3 |
| synchronize_seqscans | Mengaktifkan pemindaian berurutan yang disinkronkan. | – |
| synchronous_commit | Menetapkan tingkat sinkronisasi transaksi saat ini. | on |
| tcp_keepalives_count | Jumlah maksimum pengiriman ulang keepalive TCP. | – |
| tcp_keepalives_idle | (s) Waktu di antara penerbitan keepalive TCP. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| tcp_keepalives_interval | (s) Waktu di antara pengiriman ulang keepalive TCP. | – |
| temp_buffers | (8kB) Menetapkan jumlah maksimum buffer sementara yang digunakan oleh setiap sesi. | – |
| temp_file_limit | Membatasi jumlah total ruang disk dalam kilobyte yang dapat digunakan oleh proses PostgreSQL tertentu untuk file sementara, tidak termasuk ruang yang digunakan untuk tabel sementara eksplisit | -1 |
| temp_tablespaces | Menetapkan ruang tabel yang akan digunakan di tabel sementara dan sort file. | – |
| timezone | Menetapkan zona waktu untuk menampilkan dan menginterpretasikan stempel waktu. | UTC |
| track_activities | Mengumpulkan informasi tentang eksekusi perintah. | – |
| track_activity_query_size | Menetapkan ukuran yang disediakan untuk pg_stat_activity.current_query, dalam byte. | 4096 |
| track_commit_timestamp | Mengumpulkan waktu komit transaksi. | – |
| track_counts | Mengumpulkan statistik tentang aktivitas basis data. | – |
| track_functions | Mengumpulkan statistik tingkat fungsi tentang aktivitas basis data. | pl |
| track_io_timing | Mengumpulkan statistik waktu tentang aktivitas IO basis data. | 1 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| track_wal_io_timing | Mengumpulkan statistik waktu untuk aktivitas I/O WAL. | – |
| transform_null_equals | Memperlakukan expr=NULL sebagai expr IS NULL. | – |
| update_process_title | Memperbarui judul proses untuk menampilkan perintah SQL aktif. | – |
| vacuum_cost_delay | (ms) Penundaan biaya vacuum dalam milidetik. | – |
| vacuum_cost_limit | Jumlah biaya vacuum yang tersedia sebelum napping. | – |
| vacuum_cost_page_dirty | Biaya vacuum untuk halaman yang kotor oleh vacuum. | – |
| vacuum_cost_page_hit | Biaya vacuum untuk halaman yang ditemukan di cache buffer. | – |
| vacuum_cost_page_miss | Biaya vacuum untuk halaman yang tidak ditemukan dalam cache buffer. | 0 |
| vacuum_defer_cleanup_age | Jumlah transaksi yang harus ditangguhkan dengan pembersihan VACUUM dan HOT, jika ada. | – |
| vacuum_failsafe_age | Usia saat VACUUM harus memicu failsafe untuk menghindari gangguan wraparound. | 1200000000 |
| vacuum_freeze_min_age | Usia minimum saat VACUUM harus membekukan baris tabel. | – |
| vacuum_freeze_table_age | Usia saat VACUUM harus memindai seluruh tabel untuk membekukan tuple. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| vacuum_multixact_failsafe_age | Usia multixact saat VACUUM harus memicu failsafe untuk menghindari gangguan wraparound. | 1200000000 |
| vacuum_multixact_freeze_min_age | Usia minimum di mana VACUUM harus membekukan a MultiXactId dalam baris tabel. | – |
| vacuum_multixact_freeze_table_age | Usia multixact saat VACUUM harus memindai seluruh tabel untuk membekukan tuple. | – |
| wal_buffers | (8kB) Menetapkan jumlah buffer halaman disk dalam memori bersama untuk WAL. | – |
| wal_receiver_create_temp_slot | Menetapkan apakah penerima WAL harus membuat slot replikasi sementara jika tidak ada slot permanen yang dikonfigurasi. | 0 |
| wal_receiver_status_interval | (s) Menetapkan interval maksimum di antara laporan status penerima WAL ke primer. | – |
| wal_receiver_timeout | (ms) Menetapkan waktu tunggu maksimum untuk menerima data dari primer. | 30000 |
| wal_sender_timeout | (ms) Menetapkan waktu maksimum untuk menunggu replikasi WAL. | – |
| work_mem | (kB) Menetapkan memori maksimum yang akan digunakan untuk workspace kueri. | – |
| xmlbinary | Menetapkan cara nilai biner akan dienkode dalam XML. | – |
| xmloption | Menetapkan apakah data XML dalam operasi penguraian implisit dan serialisasi dianggap sebagai dokumen atau fragmen konten. | – |

## Parameter tingkat instans Aurora PostgreSQL

Anda dapat melihat parameter tingkat instans yang tersedia untuk versi PostgreSQL Aurora tertentu menggunakan konsol AWS Manajemen, CLI, atau Amazon RDS API. AWS Untuk informasi tentang cara melihat parameter dalam grup parameter DB Aurora PostgreSQL di konsol RDS, lihat Melihat nilai parameter untuk grup parameter DB di Aurora.

Beberapa parameter tingkat instans tidak tersedia di semua versi dan beberapa sedang dihentikan. Untuk informasi tentang cara melihat parameter versi PostgreSQL Aurora tertentu, lihat Melihat klaster DB dan parameter DB Aurora PostgreSQL.

Misalnya, tabel berikut mencantumkan parameter yang berlaku untuk instans DB tertentu di klaster DB Aurora PostgreSQL. Daftar ini dihasilkan dengan menjalankan describe-db-parameters AWS CLI perintah dengan `default.aurora-postgresql14` `--db-parameter-group-name` nilai.

Untuk daftar parameter klaster DB untuk grup parameter DB default yang sama ini, lihatParameter tingkat klaster Aurora PostgreSQL.

| Nama parameter | Deskripsi | Default |
|---|---|---|
| apg_enable_batch_mode_function_execution | Mengaktifkan fungsi mode batch untuk memproses set baris pada suatu waktu. | – |
| apg_enable_correlated_any_transform | Memungkinkan perencana mentransformasikan Subtautan ANY terkorelasi (subkueri IN/NOT IN) menjadi JOIN jika memungkinkan. | – |
| apg_enable_function_migration | Memungkinkan perencana memigrasikan fungsi skalar yang memenuhi syarat ke klausa FROM. | – |
| apg_enable_not_in_transform | Memungkinkan perencana mentransformasikan subkueri NOT IN menjadi ANTI JOIN jika memungkinkan. | – |
| apg_enable_remove_redundant_inner_joins | Memungkinkan perencana menghapus inner join yang redundan. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| apg_enable_semijoin_push_down | Memungkinkan penggunaan filter semijoin untuk hash join. | – |
| apg_plan_mgmt.capture_plan_baselines | Mengambil mode acuan dasar rencana. manual - mengaktifkan pengambilan rencana untuk pernyataan SQL apa pun, off - menonaktifkan pengambilan rencana, automatic - mengaktifkan pengambilan rencana untuk pernyataan dalam pg_stat_statements yang memenuhi kriteria kelayakan. | off |
| apg_plan_mgmt.max_databases | Menetapkan jumlah maksimum basis data yang dapat mengelola kueri menggunakan apg_plan_mgmt. | 10 |
| apg_plan_mgmt.max_plans | Menetapkan jumlah maksimum rencana yang dapat di-cache oleh apg_plan_mgmt. | 10000 |
| apg_plan_mgmt.plan_retention_period | Jumlah hari maksimum sejak rencana last_used sebelum rencana akan dihapus secara otomatis. | 32 |
| apg_plan_mgmt.unapproved_plan_execution_threshold | Perkiraan total biaya rencana yang jika tidak tercapai akan membuat rencana yang Tidak Disetujui dieksekusi. | 0 |
| apg_plan_mgmt.use_plan_baselines | Menggunakan hanya rencana yang disetujui atau tetap untuk pernyataan terkelola. | false |
| application_name | Menetapkan nama aplikasi yang akan dilaporkan dalam statistik dan log. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| aurora_compute_plan_id | Memantau rencana eksekusi kueri untuk mendeteksi rencana eksekusi yang berkontribusi pada pemuatan basis data saat ini dan untuk melacak statistik kinerja rencana eksekusi dari waktu ke waktu. Untuk informasi selengkapnya, lihat Memantau rencana eksekusi kueri untuk Aurora PostgreSQL. | on |
| aurora_temp_space_size | (MB) Menetapkan ukuran ruang yang dialokasikan untuk objek sementara yang diaktifkan Bacaan yang Dioptimalkan pada klaster Aurora I/O-Optimized dengan kelas instance yang didukung. | DBInstanceClassMemory/524288 |
| authentication_timeout | (s) Menetapkan waktu maksimum yang diizinkan untuk menyelesaikan autentikasi klien. | – |
| auto_explain.log_analyze | Menggunakan EXPLAIN ANALYZE untuk pencatatan log rencana. | – |
| auto_explain.log_buffers | Mencatat log penggunaan buffer. | – |
| auto_explain.log_format | Menggunakan EXPLAIN untuk format yang akan digunakan untuk pencatatan log rencana. | – |
| auto_explain.log_min_duration | Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat rencana dicatat. | – |
| auto_explain.log_nested_statements | Mencatat log pernyataan bersarang. | – |
| auto_explain.log_timing | Mengumpulkan data waktu, bukan hanya hitungan baris. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| auto_explain.log_triggers | Menyertakan statistik pemicu dalam rencana. | – |
| auto_explain.log_verbose | Menggunakan EXPLORE VERBOSE untuk pencatatan log rencana. | – |
| auto_explain.sample_rate | Pecahan kueri untuk diproses. | – |
| babelfishpg_tds.listen_addresses | Menetapkan nama host atau alamat IP untuk mendengarkan TDS. | * |
| babelfishpg_tds.tds_debug_log_level | Menetapkan tingkat pencatatan log di TDS, 0 menonaktifkan pencatatan log | 1 |
| backend_flush_after | (8Kb) Jumlah halaman yang jika terpenuhi akan membuat penulisan yang dilakukan sebelumnya dibuang ke disk. | – |
| bytea_output | Menetapkan format output untuk bytea. | – |
| check_function_bodies | Memeriksa konten fungsi selama CREATE FUNCTION. | – |
| client_connection_check_interval | Menetapkan interval waktu di antara pemeriksaan pemutusan koneksi saat menjalankan kueri. | – |
| client_min_messages | Menetapkan tingkat pesan yang dikirimkan ke klien. | – |
| config_file | Menetapkan file konfigurasi utama server. | /rdsdbdata/config/postgresql.conf |
| constraint_exclusion | Memungkinkan perencana menggunakan pembatasan untuk mengoptimalkan kueri. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| cpu_index_tuple_cost | Menetapkan perkiraan perencana untuk biaya pemrosesan setiap entri indeks selama pemindaian indeks. | – |
| cpu_operator_cost | Menetapkan perkiraan perencana untuk biaya pemrosesan setiap panggilan operator atau fungsi. | – |
| cpu_tuple_cost | Menetapkan perkiraan perencana untuk biaya pemrosesan setiap tuple (baris). | – |
| cron.database_name | Menetapkan basis data untuk menyimpan tabel metadata pg_cron | postgres |
| cron.log_run | Mencatat log semua pelaksanaan pekerjaan ke tabel job_run_details | on |
| cron.log_statement | Mencatat log semua pernyataan cron sebelum eksekusi. | off |
| cron.max_running_jobs | Jumlah maksimum pekerjaan yang dapat dijalankan secara bersamaan. | 5 |
| cron.use_background_workers | Mengaktifkan pekerja latar belakang untuk pg_cron | on |
| cursor_tuple_fraction | Menetapkan perkiraan perencana untuk pecahan dari baris kursor yang akan diambil. | – |
| db_user_namespace | Mengaktifkan nama pengguna per basis data. | – |
| deadlock_timeout | (ms) Menetapkan waktu untuk menunggu kunci sebelum memeriksa deadlock. | – |
| debug_pretty_print | Mengindentasi tampilan penguraian dan hierarki rencana. | – |
| debug_print_parse | Mencatat log hierarki penguraian setiap kueri. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| debug_print_plan | Mencatat log rencana eksekusi setiap kueri. | – |
| debug_print_rewritten | Mencatat log hierarki penguraian yang ditulis ulang masing-masing kueri. | – |
| default_statistics_target | Menetapkan target statistik default. | – |
| default_transaction_deferrable | Menetapkan status default yang dapat ditangguhkan untuk transaksi baru. | – |
| default_transaction_isolation | Menetapkan tingkat isolasi transaksi untuk setiap transaksi baru. | – |
| default_transaction_read_only | Menetapkan status hanya baca default untuk transaksi baru. | – |
| effective_cache_size | (8kB) Menetapkan asumsi perencana tentang ukuran cache disk. | JUMLAH (DBInstanceClassMemory/12038, -50003 |
| effective_io_concurrency | Jumlah permintaan serentak yang dapat ditangani secara efisien oleh subsistem disk. | – |
| enable_async_append | Memungkinkan perencana menggunakan rencana async append. | – |
| enable_bitmapscan | Memungkinkan perencana menggunakan rencana bitmap-scan. | – |
| enable_gathermerge | Memungkinkan perencana menggunakan rencana gather merge. | – |
| enable_hashagg | Memungkinkan perencana menggunakan rencana hashed aggregation. | – |
| enable_hashjoin | Memungkinkan perencana menggunakan rencana hash join. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| enable_incremental_sort | Memungkinkan perencana menggunakan langkah-langkah pengurutan inkremental. | – |
| enable_indexonlyscan | Memungkinkan perencana menggunakan index-only-scan rencana. | – |
| enable_indexscan | Memungkinkan perencana menggunakan rencana index-scan. | – |
| enable_material | Memungkinkan perencana menggunakan materialization. | – |
| aktifkan_memoize | Memungkinkan perencana menggunakan memoization | – |
| enable_mergejoin | Memungkinkan perencana menggunakan rencana merge join. | – |
| enable_nestloop | Memungkinkan perencana menggunakan rencana nested-loop join. | – |
| enable_parallel_append | Mengaktifkan penggunaan perencana atas rencana parallel append. | – |
| aktifkan_parallel_hash | Mengaktifkan penggunaan perencana atas rencana parallel hash. | – |
| enable_partition_pruning | Mengaktifkan pemangkasan partisi plan-time dan run-time. | – |
| enable_partitionwise_aggregate | Mengaktifkan agregasi dan pengelompokan partitionwise. | – |
| enable_partitionwise_join | Mengaktifkan partitionwise join. | – |
| enable_seqscan | Memungkinkan perencana menggunakan rencana sequential-scan. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| enable_sort | Memungkinkan perencana menggunakan langkah-langkah penyortiran eksplisit. | – |
| enable_tidscan | Memungkinkan perencana menggunakan rencana TID scan. | – |
| escape_string_warning | Memperingatkan tentang escape garis miring dalam string literal biasa. | – |
| exit_on_error | Mengakhiri sesi saat terjadi kesalahan apa pun. | – |
| force_parallel_mode | Memaksa penggunaan fasilitas kueri paralel. | – |
| from_collapse_limit | Menetapkan ukuran FROM-list yang jika terlampaui akan membuat subkueri tidak ditutup. | – |
| geqo | Memungkinkan optimisasi kueri genetik. | – |
| geqo_effort | GEQO: upaya digunakan untuk mengatur nilai default untuk parameter GEQO lainnya. | – |
| geqo_generations | GEQO: jumlah iterasi algoritma. | – |
| geqo_pool_size | GEQO: jumlah individu dalam populasi. | – |
| geqo_seed | GEQO: seed untuk pemilihan jalur acak. | – |
| geqo_selection_bias | GEQO: tekanan selektif di dalam populasi. | – |
| geqo_threshold | Menetapkan ambang batas item FROM yang jika terlampaui akan membuat GEQO digunakan. | – |
| gin_fuzzy_search_limit | Menetapkan hasil maksimum yang diizinkan untuk pencarian persis oleh GIN. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| gin_pending_list_limit | (kB) Menetapkan ukuran maksimum daftar tertunda untuk indeks GIN. | – |
| hash_mem_multiplier | Beberapa work_mem yang akan digunakan untuk tabel hash. | – |
| hba_file | Menetapkan file konfigurasi hba server. | /rdsdbdata/config/pg_hba.conf |
| hot_standby_feedback | Memungkinkan umpan balik dari hot standby ke primer yang akan menghindari konflik kueri. | on |
| ident_file | Menetapkan file konfigurasi ident server. | /rdsdbdata/config/pg_ident.conf |
| idle_in_transaction_session_timeout | (ms) Menetapkan durasi maksimum yang diizinkan untuk setiap transaksi idling. | 86400000 |
| idle_session_timeout | Mengakhiri setiap sesi yang telah idle (yaitu, menunggu kueri klien), tetapi tidak dalam transaksi terbuka, lebih lama dari jumlah waktu yang ditentukan | – |
| join_collapse_limit | Menetapkan ukuran FROM-list yang jika terlampaui akan membuat konstruk JOIN tidak diratakan. | – |
| lc_messages | Menetapkan dalam bahasa apa pesan akan ditampilkan. | – |
| listen_addresses | Menetapkan nama host atau alamat IP untuk didengarkan. | * |
| lo_compat_privileges | Mengaktifkan mode kompatibilitas mundur untuk pemeriksaan hak akses pada objek besar. | 0 |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| log_connections | Mencatat log setiap koneksi yang berhasil. | – |
| log_destination | Menetapkan tujuan untuk output log server. | stderr |
| log_directory | Menetapkan direktori tujuan untuk file log. | /rdsdbdata/log/error |
| log_disconnections | Mencatat log akhir sebuah sesi, termasuk durasinya. | – |
| log_duration | Mencatat log durasi setiap pernyataan SQL yang diselesaikan. | – |
| log_error_verbosity | Menetapkan verbositas pesan yang dicatat. | – |
| log_executor_stats | Menulis statistik performa pengeksekusi ke log server. | – |
| log_file_mode | Menetapkan izin file untuk file log. | 0644 |
| log_filename | Menetapkan pola nama file untuk file log. | postgresql.log.%Y-%m-%d-%H%M |
| logging_collector | Mulai subproses untuk menangkap output stderr dan/atau csvlog ke dalam file log. | 1 |
| log_hostname | Mencatat log nama host dalam log koneksi. | 0 |
| logical_decoding_work_mem | (kB) Memori sebanyak ini dapat digunakan oleh setiap buffer penyusun ulang internal sebelum dialirkan ke disk. | – |
| log_line_prefix | Mengontrol informasi yang ditambahkan ke awal setiap baris log. | %t:%r:%u@%d:%p]: |
| log_lock_waits | Mencatat log waktu tunggu kunci yang panjang. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| log_min_duration_sample | (ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat sampel pernyataan dicatat. Pengambilan sampel ditentukan oleh log_statement_sample_rate. | – |
| log_min_duration_statement | (ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat pernyataan dicatat. | – |
| log_min_error_statement | Menyebabkan semua pernyataan yang menghasilkan kesalahan pada atau di atas tingkat ini dicatat. | – |
| log_min_messages | Menetapkan tingkat pesan yang dicatat. | – |
| log_parameter_max_length | (B) Saat mencatat pernyataan, membatasi nilai parameter yang dicatat ke N byte pertama. | – |
| log_parameter_max_length_on_error | (B) Saat melaporkan kesalahan, membatasi nilai parameter yang dicatat ke N byte pertama. | – |
| log_parser_stats | Menulis statistik performa pengurai ke log server. | – |
| log_planner_stats | Menulis statistik performa perencana ke log server. | – |
| log_replication_commands | Mencatat log setiap perintah replikasi. | – |
| log_rotation_age | (min) Rotasi file log otomatis akan terjadi setelah N menit. | 60 |
| log_rotation_size | (kB) Rotasi file log otomatis akan terjadi setelah N kilobyte. | 100000 |
| log_statement | Menetapkan jenis pernyataan yang dicatat. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| log_statement_sample_rate | Pecahan dari pernyataan yang melebihi log_min_duration_sample yang akan dicatat. | – |
| log_statement_stats | Menulis statistik performa kumulatif ke log server. | – |
| log_temp_files | (kB) Mencatat log penggunaan file sementara yang lebih besar dari jumlah kilobyte ini. | – |
| log_timezone | Menetapkan zona waktu yang akan digunakan dalam pesan log. | UTC |
| log_truncate_on_rotation | Memotong file log yang ada dengan nama yang sama selama rotasi log. | 0 |
| maintenance_io_concurrency | Varian effective_io_concurrency yang digunakan untuk pekerjaan pemeliharaan. | 1 |
| maintenance_work_mem | (kB) Menetapkan memori maksimum yang akan digunakan untuk operasi pemeliharaan. | TERBESAR (DBInstanceClassMemory/63963136 *1024,65536 |
| max_connections | Menetapkan jumlah maksimum koneksi serentak. | PALING SEDIKIT (DBInstanceClassMemory/9531392 ,5000 |
| max_files_per_process | Menetapkan jumlah maksimum file yang terbuka secara bersamaan untuk setiap proses server. | – |
| max_locks_per_transaction | Menetapkan jumlah maksimum kunci per transaksi. | 64 |
| max_parallel_maintenance_workers | Menetapkan jumlah maksimum proses paralel per operasi pemeliharaan. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| max_parallel_workers | Menetapkan jumlah maksimum pekerja paralel yang dapat aktif pada satu waktu. | TERBESAR ($ DBInstance VCPU/2,8 |
| max_parallel_workers_per_gather | Menetapkan jumlah maksimum proses paralel per simpul eksekutor. | – |
| max_pred_locks_per_page | Menetapkan jumlah maksimum tuple predicate-lock per halaman. | – |
| max_pred_locks_per_relation | Menetapkan jumlah maksimum halaman dan tuple predicate-lock per relasi. | – |
| max_pred_locks_per_transaction | Menetapkan jumlah maksimum kunci predikat per transaksi. | – |
| max_slot_wal_keep_size | (MB) Slot replikasi akan ditandai sebagai gagal, dan segmen akan dirilis untuk dihapus atau didaur ulang, jika ruang sebanyak ini ditempati oleh WAL pada disk. | – |
| max_stack_depth | (kB) Menetapkan kedalaman tumpukan maksimum, dalam kilobyte. | 6144 |
| max_standby_streaming_delay | (ms) Menetapkan penundaan maksimum sebelum membatalkan kueri saat hot standby sedang memproses data WAL yang dialirkan. | 14000 |
| max_worker_processes | Menetapkan jumlah maksimum proses pekerja serentak. | TERBESAR ($ DBInstance VCPU*2,8 |
| min_dynamic_shared_memory | (MB) Jumlah memori bersama dinamis yang dicadangkan saat pengaktifan. | – |
| min_parallel_index_scan_size | (8kB) Menetapkan jumlah minimum data indeks untuk pemindaian paralel. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| min_parallel_table_scan_size | (8kB) Menetapkan jumlah minimum data tabel untuk pemindaian paralel. | – |
| old_snapshot_threshold | (min) Waktu sebelum snapshot terlalu lama untuk membaca halaman yang diubah setelah snapshot diambil. | – |
| parallel_leader_participation | Mengontrol apakah Gather and Gather Merge juga menjalankan subrencana. | – |
| parallel_setup_cost | Menetapkan perkiraan perencana terkait biaya untuk memulai proses pekerja untuk kueri paralel. | – |
| parallel_tuple_cost | Menetapkan perkiraan perencana untuk biaya pengiriman setiap tuple (baris) dari pekerja ke backend master. | – |
| pgaudit.log | Menentukan kelas pernyataan mana yang akan dicatat oleh pencatatan log audit sesi. | – |
| pgaudit.log_catalog | Menentukan bahwa pencatatan log sesi harus diaktifkan jika semua relasi dalam pernyataan berada di pg_catalog. | – |
| pgaudit.log_level | Menentukan tingkat log yang akan digunakan untuk entri log. | – |
| pgaudit.log_parameter | Menentukan bahwa audit pencatatan log harus mencakup parameter yang diteruskan dengan pernyataan. | – |
| pgaudit.log_relation | Menentukan apakah pencatatan log audit sesi harus membuat entri log terpisah untuk setiap relasi (TABLE, VIEW, dll.) yang direferensikan dalam pernyataan SELECT atau DML. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pgaudit.log_statement_once | Menentukan apakah pencatatan log akan mencakup teks pernyataan dan parameter dengan entri log pertama untuk kombinasi pernyataan/subpernyataan atau dengan setiap entri. | – |
| pgaudit.role | Menentukan peran master yang akan digunakan untuk pencatatan log audit objek. | – |
| pg_bigm.enable_recheck | Menentukan apakah akan melakukan Pemeriksaan Ulang yang merupakan proses internal pencarian teks lengkap. | on |
| pg_bigm.gin_key_limit | Menentukan jumlah maksimum 2 gram kata kunci pencarian yang akan digunakan untuk pencarian teks lengkap. | 0 |
| pg_bigm.last_update | Melaporkan tanggal pembaruan terakhir modul pg_bigm. | 2013.11.22 |
| pg_bigm.similarity_limit | Menentukan ambang batas minimum yang digunakan oleh pencarian kesamaan. | 0,3 |
| pg_hint_plan.debug_print | Mencatat log hasil dari penguraian petunjuk. | – |
| pg_hint_plan.enable_hint | Memaksa perencana untuk menggunakan rencana yang ditentukan dalam komentar petunjuk sebelum kueri. | – |
| pg_hint_plan.enable_hint_table | Memaksa perencana untuk tidak mendapatkan petunjuk dengan menggunakan pencarian tabel. | – |
| pg_hint_plan.message_level | Tingkat pesan untuk pesan debug. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_hint_plan.parse_messages | Tingkat pesan kesalahan penguraian. | – |
| pglogical.batch_inserts | Melakukan penyisipan batch jika memungkinkan | – |
| pglogical.conflict_log_level | Menetapkan tingkat log yang digunakan untuk mencatat konflik yang diselesaikan. | – |
| pglogical.conflict_resolution | Menetapkan metode yang digunakan untuk penyelesaian konflik untuk konflik yang dapat diselesaikan. | – |
| pglogical.extra_connection_options | Opsi koneksi untuk ditambahkan ke semua koneksi simpul peer | – |
| pglogical.synchronous_commit | nilai komit sinkron spesifik pglogical | – |
| pglogical.use_spi | Menggunakan SPI alih-alih API tingkat rendah untuk menerapkan perubahan | – |
| pg_similarity.block_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.block_threshold | Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Blok. | – |
| pg_similarity.block_tokenizer | Menetapkan tokenizer untuk fungsi kesamaan Blok. | – |
| pg_similarity.cosine_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.cosine_threshold | Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Kosinus. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_similarity.cosine_tokenizer | Menetapkan pentokenisasi untuk fungsi kesamaan Kosinus. | – |
| pg_similarity.dice_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.dice_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Dice. | – |
| pg_similarity.dice_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Dice. | – |
| pg_similarity.euclidean_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.euclidean_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Euclidean. | – |
| pg_similarity.euclidean_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Euclidean. | – |
| pg_similarity.hamming_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.hamming_threshold | Menetapkan ambang batas yang digunakan oleh metrik kesamaan Blok. | – |
| pg_similarity.jaccard_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.jaccard_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaccard. | – |
| pg_similarity.jaccard_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Jaccard. | – |
| pg_similarity.jaro_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_similarity.jaro_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaro. | – |
| pg_similarity.jarowinkler_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.jarowinkler_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jarowinkler. | – |
| pg_similarity.levenshtein_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.levenshtein_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Levenshtein. | – |
| pg_similarity.matching_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.matching_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Pencocokan. | – |
| pg_similarity.matching_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Pencocokan. | – |
| pg_similarity.mongeelkan_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.mongeelkan_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Monge-Elkan. | – |
| pg_similarity.mongeelkan_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Monge-Elkan. | – |
| pg_similarity.nw_gap_penalty | Menetapkan penalti kesenjangan yang digunakan oleh ukuran kesamaan Needleman-Wunsch. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_similarity.nw_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.nw_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Needleman-Wunsch. | – |
| pg_similarity.overlap_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.overlap_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Tumpang-Tindih. | – |
| pg_similarity.overlap_tokenizer | Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Tumpang-Tindih. | – |
| pg_similarity.qgram_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.qgram_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Q-Gram. | – |
| pg_similarity.qgram_tokenizer | Menetapkan pentokenisasi untuk ukuran Q-Gram. | – |
| pg_similarity.swg_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.swg_threshold | Menetapkan ambang batas yang digunakan oleh ukuran Smith-Waterman-Gotoh kesamaan. | – |
| pg_similarity.sw_is_normalized | Menetapkan apakah nilai hasil dinormalisasi atau tidak. | – |
| pg_similarity.sw_threshold | Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Smith-Waterman. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| pg_stat_statements.max | Menetapkan jumlah maksimum pernyataan yang dilacak oleh pg_stat_statements. | – |
| pg_stat_statements.save | Save pg_stat_statements statistics across server shutdowns. | – |
| pg_stat_statements.track | Memilih pernyataan mana yang dilacak oleh pg_stat_statements. | – |
| pg_stat_statements.track_planning | Memilih apakah durasi perencanaan dilacak oleh pg_stat_statements. | – |
| pg_stat_statements.track_utility | Memilih apakah perintah utilitas dilacak oleh pg_stat_statements. | – |
| postgis.gdal_enabled_drivers | Mengaktifkan atau menonaktifkan driver GDAL yang digunakan dengan PostGIS di Postgres 9.3.5 dan versi lebih tinggi. | ENABLE_ALL |
| quote_all_identifiers | Saat membuat fragmen SQL, mengutip semua pengidentifikasi. | – |
| random_page_cost | Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara tidak berurutan. | – |
| rds.enable_memory_management | Meningkatkan kemampuan manajemen memori di Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5, dan versi yang lebih tinggi yang mencegah masalah stabilitas dan restart database yang disebabkan oleh memori bebas yang tidak mencukupi. Untuk informasi selengkapnya, lihat Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL. | True |
| rds.force_admin_logging_level | Melihat pesan log untuk tindakan pengguna admin RDS di basis data pelanggan. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| rds.log_retention_period | Amazon RDS akan menghapus log PostgreSQL yang lebih lama dari N menit. | 4320 |
| rds.memory_allocation_guard | Meningkatkan kemampuan manajemen memori di Aurora PostgreSQL 11.21, 12.16, 13.12, 14.9, 15.4, dan versi lama yang mencegah masalah stabilitas dan restart database yang disebabkan oleh memori bebas yang tidak mencukupi. Untuk informasi selengkapnya, lihat Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL. | False |
| rds.pg_stat_ramdisk_size | Ukuran statistik ramdisk dalam MB. Nilai bukan nol akan mengatur ramdisk. | 0 |
| rds.rds_superuser_reserved_connections | Menetapkan jumlah slot koneksi yang disediakan untuk rds_superusers. Parameter ini hanya tersedia dalam versi 15 dan sebelumnya. Untuk informasi selengkapnya, lihat koneksi cadangan dokumentasi PostgreSQL. | 2 |
| rds.superuser_variables | Daftar variabel superuser-only yang pernyataan modifikasi rds_superuser-nya dielevasikan. | session_replication_role |
| remove_temp_files_after_crash | Menghapus file sementara setelah backend crash. | 0 |
| restart_after_crash | Menginisialisasi ulang server setelah backend crash. | – |
| row_security | Mengaktifkan keamanan baris. | – |
| search_path | Menetapkan urutan pencarian skema untuk nama yang tidak memenuhi syarat skema. | – |
| seq_page_cost | Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara berurutan. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| session_replication_role | Menetapkan perilaku sesi untuk pemicu dan aturan penulisan ulang. | – |
| shared_buffers | (8kB) Menetapkan jumlah buffer memori bersama yang digunakan oleh server. | JUMLAH (DBInstanceClassMemory/12038, -50003) |
| shared_preload_libraries | Menampilkan daftar pustaka bersama untuk dimuat sebelumnya ke server. | pg_stat_statements |
| ssl_ca_file | Lokasi file otoritas server SSL. | /rdsdbdata/rds-metadata/ca-cert.pem |
| ssl_cert_file | Lokasi file sertifikat server SSL. | /rdsdbdata/rds-metadata/server-cert.pem |
| ssl_crl_dir | Lokasi direktori daftar pencabutan sertifikat SSL. | /rdsdbdata/rds-metadata/ssl_crl_dir/ |
| ssl_key_file | Lokasi file kunci privat server SSL | /rdsdbdata/rds-metadata/server-key.pem |
| standard_conforming_strings | Menyebabkan string ... memperlakukan garis miring secara literal. | – |
| statement_timeout | (ms) Menetapkan durasi maksimum yang diizinkan untuk setiap pernyataan. | – |
| stats_temp_directory | Menulis file statistik sementara ke direktori yang ditentukan. | /rdsdbdata/db/pg_stat_tmp |
| superuser_reserved_connections | Menetapkan jumlah slot koneksi yang disediakan untuk pengguna super. | 3 |
| synchronize_seqscans | Mengaktifkan pemindaian berurutan yang disinkronkan. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| tcp_keepalives_count | Jumlah maksimum pengiriman ulang keepalive TCP. | – |
| tcp_keepalives_idle | (s) Waktu di antara penerbitan keepalive TCP. | – |
| tcp_keepalives_interval | (s) Waktu di antara pengiriman ulang keepalive TCP. | – |
| temp_buffers | (8kB) Menetapkan jumlah maksimum buffer sementara yang digunakan oleh setiap sesi. | – |
| temp_file_limit | Membatasi jumlah total ruang disk dalam kilobyte yang dapat digunakan oleh proses PostgreSQL tertentu untuk file sementara, tidak termasuk ruang yang digunakan untuk tabel sementara eksplisit | -1 |
| temp_tablespaces | Menetapkan ruang tabel yang akan digunakan di tabel sementara dan sort file. | – |
| track_activities | Mengumpulkan informasi tentang eksekusi perintah. | – |
| track_activity_query_size | Menetapkan ukuran yang disediakan untuk pg_stat_activity.current_query, dalam byte. | 4096 |
| track_counts | Mengumpulkan statistik tentang aktivitas basis data. | – |
| track_functions | Mengumpulkan statistik tingkat fungsi tentang aktivitas basis data. | pl |
| track_io_timing | Mengumpulkan statistik waktu tentang aktivitas IO basis data. | 1 |
| transform_–_equals | Memperlakukan expr=– sebagai expr IS –. | – |

| Nama parameter | Deskripsi | Default |
| --- | --- | --- |
| update_process_title | Memperbarui judul proses untuk menampilkan perintah SQL aktif. | – |
| wal_receiver_status_interval | (s) Menetapkan interval maksimum di antara laporan status penerima WAL ke primer. | – |
| work_mem | (kB) Menetapkan memori maksimum yang akan digunakan untuk workspace kueri. | – |
| xmlbinary | Menetapkan cara nilai biner akan dienkode dalam XML. | – |
| xmloption | Menetapkan apakah data XML dalam operasi penguraian implisit dan serialisasi dianggap sebagai dokumen atau fragmen konten. | – |

## Peristiwa tunggu Amazon Aurora PostgreSQL

Berikut ini adalah peristiwa tunggu yang umum terjadi di Aurora PostgreSQL. Untuk mempelajari selengkapnya tentang peristiwa tunggu dan menyetel klaster DB PostgreSQL, lihat Menyetel dengan peristiwa tunggu di Aurora PostgreSQL.

Aktivitas: ArchiverMain

Proses pengarsip menunggu aktivitas.

Aktivitas: AutoVacuumMain

Proses peluncur autovacuum menunggu aktivitas.

Aktivitas: BgWriterHibernate

Proses penulis latar belakang berhibernasi sambil menunggu aktivitas.

Aktivitas: BgWriterMain

Proses penulis latar belakang menunggu aktivitas.

Aktivitas: CheckpointerMain

Proses checkpointer menunggu aktivitas.

Aktivitas: LogicalApplyMain

Proses penerapan replikasi logis menunggu aktivitas.

Aktivitas: LogicalLauncherMain

Proses peluncur replikasi logis menunggu aktivitas.

Aktivitas: PgStatMain

Proses pengumpul statistik menunggu aktivitas.

Aktivitas: RecoveryWalAll

Sebuah proses menunggu write-ahead log (WAL) dari aliran saat pemulihan.

Aktivitas: RecoveryWalStream

Proses pengaktifan menunggu write-ahead log (WAL) untuk tiba selama pemulihan aliran.

Aktivitas: SysLoggerMain

Proses syslogger menunggu aktivitas.

Aktivitas: WalReceiverMain

Proses penerima write-ahead log (WAL) menunggu aktivitas.

Aktivitas: WalSenderMain

Proses pengirim write-ahead log (WAL) menunggu aktivitas.

Aktivitas: WalWriterMain

Proses penulis write-ahead log (WAL) menunggu aktivitas.

BufferPin:BufferPin

Sebuah proses menunggu untuk mendapatkan pin eksklusif pada buffer.

Klien: GSSOpen Server

Sebuah proses menunggu untuk membaca data dari klien sambil membuat sesi Generic Security Service Application Program Interface (GSSAPI).

Klien: ClientRead

Sebuah proses backend menunggu untuk menerima data dari klien PostgreSQL. Untuk informasi selengkapnya, lihat Klien: ClientRead.

Klien: ClientWrite

Sebuah proses backend menunggu untuk mengirim lebih banyak data ke klien PostgreSQL. Untuk informasi selengkapnya, lihat Klien: ClientWrite.

Klien:Lib PQWal ReceiverConnect

Sebuah proses menunggu di penerima write-ahead log (WAL) untuk membuat koneksi ke server jarak jauh.

Klien:Lib PQWal ReceiverReceive

Sebuah proses menunggu di penerima write-ahead log (WAL) untuk menerima data dari server jarak jauh.

Klien: SSLOpen Server

Sebuah proses menunggu Lapisan Soket Aman (SSL) saat mencoba koneksi.

Klien: WalReceiverWaitStart

Sebuah proses menunggu proses pengaktifan untuk mengirim data awal untuk replikasi aliran.

Klien: WalSenderWaitFor WAL

Sebuah proses menunggu write-ahead log (WAL) untuk dibuang dalam proses pengirim WAL.

Klien: WalSenderWriteData

Sebuah proses menunggu aktivitas saat memproses balasan dari penerima write-ahead log (WAL) dalam proses pengirim WAL.

CPU

Sebuah proses backend aktif atau menunggu CPU. Untuk informasi selengkapnya, lihat CPU.

Extension:extension

Sebuah proses backend menunggu kondisi yang ditentukan oleh ekstensi atau modul.

IO: AuroraEnhancedLogical WALRead

Proses backend mengambil catatan log dari volume change data capture (CDC).

IO: AuroraOptimizedReadsCacheRead

Sebuah proses menunggu pembacaan dari cache bertingkat Optimized Reads karena halaman tidak tersedia dalam memori bersama.

IO: AuroraOptimizedReads CacheSegmentMemotong

Sebuah proses menunggu file segmen cache bertingkat Optimized Reads untuk dipotong.

IO: AuroraOptimizedReadsCacheWrite

Proses penulis latar belakang menunggu untuk menulis di cache bertingkat Optimized Reads.

IO: AuroraStorageLogAllocate

Sebuah sesi mengalokasikan metadata dan mempersiapkan penulisan log transaksi.

IO: BufFileRead

Ketika operasi membutuhkan lebih banyak memori daripada jumlah yang ditentukan oleh parameter memori kerja, mesin akan membuat file sementara pada disk. Peristiwa tunggu ini terjadi ketika operasi membaca dari file sementara. Untuk informasi selengkapnya, lihat IO:BufFileRead dan IO:BufFileWrite.

IO: BufFileWrite

Ketika operasi membutuhkan lebih banyak memori daripada jumlah yang ditentukan oleh parameter memori kerja, mesin akan membuat file sementara pada disk. Peristiwa tunggu ini terjadi ketika operasi menulis ke file sementara. Untuk informasi selengkapnya, lihat IO:BufFileRead dan IO:BufFileWrite.

IO: ControlFileRead

Sebuah proses menunggu pembacaan dari file `pg_control`.

IO: ControlFileSync

Sebuah proses menunggu file `pg_control` untuk mencapai penyimpanan durabel.

IO: ControlFileSyncUpdate

Sebuah proses menunggu pembaruan pada file `pg_control` untuk mencapai penyimpanan durabel.

IO: ControlFileWrite

Sebuah proses menunggu penulisan pada file `pg_control`.

IO: ControlFileWriteUpdate

Sebuah proses menunggu penulisan untuk memperbarui file `pg_control`.

IO: CopyFileRead

Sebuah proses menunggu pembacaan selama operasi penyalinan file.

IO: CopyFileWrite

Sebuah proses menunggu penulisan selama operasi penyalinan file.

IO: DataFileExtend

Sebuah proses menunggu file data relasi untuk diperluas.

IO: DataFileFlush

Sebuah proses menunggu file data relasi untuk mencapai penyimpanan durabel.

IO: DataFileImmediateSync

Sebuah proses menunggu sinkronisasi langsung file data relasi ke penyimpanan durabel.

IO: DataFilePrefetch

Sebuah proses menunggu prefetch asinkron dari file data relasi.

IO: DataFileSync

Sebuah proses menunggu perubahan pada file data relasi untuk mencapai penyimpanan durabel.

IO: DataFileRead

Sebuah proses backend mencoba menemukan halaman di buffer bersama, tidak menemukannya, sehingga membacanya dari penyimpanan. Untuk informasi selengkapnya, lihat IO: DataFileRead.

IO: DataFileTruncate

Sebuah proses menunggu file data relasi untuk dipotong.

IO: DataFileWrite

Sebuah proses menunggu penulisan pada file data relasi.

IO: DSMFill ZeroWrite

Sebuah proses menunggu untuk menulis nol byte pada backing file memori bersama dinamis.

IO: LockFileAddToDataDirRead

Sebuah proses menunggu pembacaan sambil menambahkan baris pada file kunci direktori data.

IO: LockFileAddToDataDirSync

Sebuah proses menunggu data untuk mencapai penyimpanan durabel sambil menambahkan baris pada file kunci direktori data.

IO: LockFileAddToDataDirWrite

Sebuah proses menunggu penulisan sambil menambahkan baris pada file kunci direktori data.

IO: LockFileCreateRead

Sebuah proses menunggu untuk membaca saat membuat file kunci direktori data.

IO: LockFileCreateSync

Sebuah proses menunggu data untuk mencapai penyimpanan durabel sambil membuat file kunci direktori data.

IO: LockFileCreateWrite

Sebuah proses menunggu penulisan saat membuat file kunci direktori data.

IO: LockFileReCheckDataDirRead

Sebuah proses menunggu pembacaan selama pemeriksaan ulang file kunci direktori data.

IO: LogicalRewriteCheckpointSync

Sebuah proses menunggu pemetaan penulisan ulang logis untuk mencapai penyimpanan durabel selama checkpoint.

IO: LogicalRewriteMappingSync

Sebuah proses menunggu pemetaan data untuk mencapai penyimpanan durabel selama penulisan ulang logis.

IO: LogicalRewriteMappingWrite

Sebuah proses menunggu penulisan data pemetaan selama penulisan ulang logis.

IO: LogicalRewriteSync

Sebuah proses menunggu pemetaan penulisan ulang logis untuk mencapai penyimpanan durabel.

IO: LogicalRewriteTruncate

Sebuah proses menunggu pemotongan data pemetaan selama penulisan ulang logis.

IO: LogicalRewriteWrite

Sebuah proses menunggu penulisan pemetaan penulisan ulang logis.

IO: RelationMapRead

Sebuah proses menunggu pembacaan file peta relasi.

IO: RelationMapSync

Sebuah proses menunggu file peta relasi untuk mencapai penyimpanan durabel.

IO: RelationMapWrite

Sebuah proses menunggu penulisan pada file peta relasi.

IO: ReorderBufferRead

Sebuah proses menunggu pembacaan selama manajemen buffer penyusunan ulang.

IO: ReorderBufferWrite

Sebuah proses menunggu penulisan selama manajemen buffer penyusunan ulang.

IO: ReorderLogicalMappingRead

Sebuah proses menunggu pembacaan pemetaan logis selama manajemen buffer penyusunan ulang.

IO: ReplicationSlotRead

Sebuah proses menunggu pembacaan dari file kontrol slot replikasi.

IO: ReplicationSlotRestoreSync

Sebuah proses menunggu file kontrol slot replikasi untuk mencapai penyimpanan durabel sambil memulihkannya ke memori.

IO: ReplicationSlotSync

Sebuah proses menunggu file kontrol slot replikasi untuk mencapai penyimpanan durabel.

IO: ReplicationSlotWrite

Sebuah proses menunggu penulisan pada file kontrol slot replikasi.

IO: SLRUFlush Sinkronisasi

Sebuah proses menunggu data simple least-recently used (SLRU) untuk mencapai penyimpanan durabel selama checkpoint atau penonaktifan basis data.

IO: SLRURead

Sebuah proses menunggu pembacaan halaman simple least-recently used (SLRU).

IO: SLRUSync

Sebuah proses menunggu data simple least-recently used (SLRU) untuk mencapai penyimpanan durabel setelah penulisan halaman.

IO: SLRUWrite

Sebuah proses menunggu penulisan halaman simple least-recently used (SLRU).

IO: SnapbuildRead

Sebuah proses menunggu pembacaan snapshot katalog historis terserialisasi.

IO: SnapbuildSync

Sebuah proses menunggu snapshot katalog historis terserialisasi untuk mencapai penyimpanan durabel.

IO: SnapbuildWrite

Sebuah proses menunggu penulisan snapshot katalog historis terserialisasi.

IO: TimelineHistoryFileSync

Sebuah proses menunggu file riwayat lini masa yang diterima melalui replikasi aliran untuk mencapai penyimpanan durabel.

IO: TimelineHistoryFileWrite

Sebuah proses menunggu penulisan file riwayat lini masa yang diterima melalui replikasi aliran.

IO: TimelineHistoryRead

Sebuah proses menunggu pembacaan file riwayat lini masa.

IO: TimelineHistorySync

Sebuah proses menunggu file riwayat lini masa yang baru dibuat untuk mencapai penyimpanan durabel.

IO: TimelineHistoryWrite

Sebuah proses menunggu penulisan file riwayat lini masa yang baru dibuat.

IO: TwophaseFileRead

Sebuah proses menunggu pembacaan file status dua fase.

IO: TwophaseFileSync

Sebuah proses menunggu file status dua fase untuk mencapai penyimpanan durabel.

IO: TwophaseFileWrite

Sebuah proses menunggu penulisan file status dua fase.

IO: WALBootstrap Sinkronisasi

Sebuah proses menunggu write-ahead log (WAL) untuk mencapai penyimpanan durabel selama bootstrapping.

IO: WALBootstrap Menulis

Sebuah proses menunggu penulisan halaman write-ahead log (WAL) selama bootstrapping.

IO: WALCopy Baca

Sebuah proses menunggu pembacaan saat membuat segmen write-ahead log (WAL) baru dengan menyalin yang sudah ada.

IO: WALCopy Sinkronisasi

Sebuah proses menunggu segmen write-ahead log (WAL) baru yang dibuat dengan menyalin yang sudah ada untuk mencapai penyimpanan durabel.

IO: WALCopy Menulis

Sebuah proses menunggu penulisan saat membuat segmen write-ahead log (WAL) baru dengan menyalin yang sudah ada.

IO: WALInit Sinkronisasi

Sebuah proses menunggu file write-ahead log (WAL) yang baru diinisialisasi untuk mencapai penyimpanan durabel.

IO: WALInit Menulis

Sebuah proses menunggu penulisan saat menginisialisasi file write-ahead log (WAL).

IO: WALRead

Sebuah proses menunggu pembacaan dari file write-ahead log (WAL).

IO: WALSender TimelineHistoryRead

Sebuah proses menunggu pembacaan dari file riwayat lini masa selama perintah lini masa pengirim WAL.

IO: WALSync

Sebuah proses menunggu file write-ahead log (WAL) untuk mencapai penyimpanan durabel.

IO: WALSync MethodAssign

Sebuah proses menunggu data untuk mencapai penyimpanan durabel sambil menetapkan metode sinkronisasi write-ahead log (WAL) baru.

IO: WALWrite

Sebuah proses menunggu penulisan pada file write-ahead log (WAL).

IO: XactSync

Sebuah proses backend menunggu subsistem penyimpanan Aurora untuk mengonfirmasi komit transaksi reguler, atau komit atau rollback transaksi yang disiapkan. Untuk informasi selengkapnya, lihat IO: XactSync.

IPC: AuroraLogicalSchemaUpdate

Dua proses backend mencoba memasukkan entri yang sama ke dalam cache skema. Satu proses akan berlanjut sementara yang lain menunggu sampai selesai.

IPC: AuroraOptimizedReadsCacheWriteStop

Sebuah proses sedang menunggu penulis latar belakang untuk berhenti menulis ke cache berjenjang Optimized Reads.

IPC: BackupWaitWalArchive

Sebuah proses menunggu file write-ahead log (WAL) yang diperlukan agar cadangan berhasil diarsipkan.

IPC: BgWorkerShutdown

Sebuah proses menunggu pekerja latar belakang untuk dinonaktifkan.

IPC: BgWorkerStartup

Sebuah proses menunggu pekerja latar belakang untuk dimulai.

IPC: BtreePage

Sebuah proses menunggu nomor halaman yang diperlukan untuk melanjutkan pemindaian B-tree paralel menjadi tersedia.

IPC: CheckpointDone

Sebuah proses menunggu checkpoint selesai.

IPC: CheckpointStart

Sebuah proses menunggu checkpoint untuk dimulai.

IPC: ClogGroupUpdate

Sebuah proses menunggu pemimpin grup untuk memperbarui status transaksi di akhir transaksi.

IPC: DamRecordTxAck

Sebuah proses backend telah menghasilkan peristiwa aliran aktivitas basis data dan menunggu peristiwa menjadi durabel. Untuk informasi selengkapnya, lihat IPC:DamRecordTxAck.

IPC: ExecuteGather

Sebuah proses menunggu aktivitas dari proses turunan saat menjalankan simpul rencana Gather.

IPC: Hash/Batch/Allocating

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash.

IPC: Hash/Batch/Electing

Sebuah proses memilih peserta hash paralel untuk mengalokasikan tabel hash.

IPC: Hash/Batch/Loading

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan pemuatan tabel hash.

IPC: Hash/Build/Allocating

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash awal.

IPC: Hash/Build/Electing

Sebuah proses memilih peserta hash paralel untuk mengalokasikan tabel hash awal.

IPC: Hash/Build/HashingInner

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan hashing relasi dalam.

IPC: Hash/Build/HashingOuter

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi relasi luar.

IPC: Hash/GrowBatches/Allocating

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan lebih banyak batch.

IPC: Hash/GrowBatches/Deciding

Sebuah proses memilih peserta hash paralel untuk memutuskan pertumbuhan batch mendatang.

IPC: Hash/GrowBatches/Electing

Sebuah proses memilih peserta hash paralel untuk mengalokasikan lebih banyak batch.

IPC: Hash/GrowBatches/Finishing

Sebuah proses menunggu peserta hash paralel terpilih untuk memutuskan pertumbuhan batch mendatang.

IPC: Hash/GrowBatches/Repartitioning

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi ulang.

IPC: Hash/GrowBuckets/Allocating

Sebuah proses menunggu peserta hash paralel terpilih untuk menyelesaikan alokasi lebih banyak bucket.

IPC: Hash/GrowBuckets/Electing

Sebuah proses memilih peserta hash paralel untuk mengalokasikan lebih banyak bucket.

IPC: Hash/GrowBuckets/Reinserting

Sebuah proses menunggu peserta hash paralel lainnya selesai menyisipkan tuple ke dalam bucket baru.

IPC: HashBatchAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash.

IPC: HashBatchElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan tabel hash.

IPC: HashBatchLoad

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan pemuatan tabel hash.

IPC: HashBuildAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash awal.

IPC: HashBuildElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan tabel hash awal.

IPC: HashBuildHashInner

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan hashing relasi dalam.

IPC: 'HashBuildHashOuter

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi relasi luar.

IPC: HashGrowBatchesAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan lebih banyak batch.

IPC: 'HashGrowBatchesDecide

Sebuah proses menunggu untuk memilih peserta hash paralel untuk memutuskan pertumbuhan batch mendatang.

IPC: HashGrowBatchesElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan lebih banyak batch.

IPC: HashGrowBatchesFinish

Sebuah proses menunggu peserta hash paralel terpilih untuk memutuskan pertumbuhan batch mendatang.

IPC: HashGrowBatchesRepartition

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi ulang.

IPC: HashGrowBucketsAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk menyelesaikan alokasi lebih banyak bucket.

IPC: HashGrowBucketsElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan lebih banyak bucket.

IPC: HashGrowBucketsReinsert

Sebuah proses menunggu peserta hash paralel lainnya selesai menyisipkan tuple ke dalam bucket baru.

IPC: LogicalSyncData

Sebuah proses menunggu server jarak jauh replikasi logis untuk mengirim data untuk sinkronisasi tabel awal.

IPC: LogicalSyncStateChange

Sebuah proses menunggu server jarak jauh replikasi logis untuk mengubah status.

IPC: MessageQueueInternal

Sebuah proses menunggu proses lain untuk dilampirkan ke antrean pesan bersama.

IPC: MessageQueuePutMessage

Sebuah proses menunggu untuk menulis pesan protokol ke antrean pesan bersama.

IPC: MessageQueueReceive

Sebuah proses menunggu untuk menerima byte dari antrean pesan bersama.

IPC: MessageQueueSend

Sebuah proses menunggu untuk mengirim byte ke antrean pesan bersama.

IPC: ParallelBitmapScan

Sebuah proses menunggu pemindaian bitmap paralel untuk diinisialisasi.

IPC: ParallelCreateIndexScan

Sebuah proses menunggu pekerja CREATE INDEX paralel untuk menyelesaikan pemindaian heap.

IPC: ParallelFinish

Sebuah proses menunggu pekerja paralel untuk menyelesaikan penghitungan.

IPC: ProcArrayGroupUpdate

Sebuah proses sedang menunggu pemimpin grup untuk menghapus ID transaksi pada akhir transaksi.

IPC: ProcSignalBarrier

Sebuah proses menunggu peristiwa penghalang untuk diproses oleh semua backend.

IPC:Promote

Sebuah proses menunggu promosi siaga.

IPC: RecoveryConflictSnapshot

Sebuah proses menunggu penyelesaian pertentangan pemulihan untuk pembersihan vacuum.

IPC: RecoveryConflictTablespace

Sebuah proses menunggu penyelesaian pertentangan pemulihan untuk menghapus ruang tabel.

IPC: RecoveryPause

Sebuah proses menunggu pemulihan untuk dilanjutkan.

IPC: ReplicationOriginDrop

Sebuah proses menunggu asal replikasi menjadi tidak aktif agar dapat dihapus.

IPC: ReplicationSlotDrop

Sebuah proses menunggu slot replikasi menjadi tidak aktif agar dapat dihapus.

IPC: SafeSnapshot

Sebuah proses menunggu untuk mendapatkan snapshot yang valid untuk transaksi READ ONLY DEFERRABLE.

IPC: SyncRep

Sebuah proses menunggu konfirmasi dari server jarak jauh selama replikasi sinkron.

IPC: XactGroupUpdate

Sebuah proses sedang menunggu pemimpin grup untuk memperbarui status transaksi pada akhir transaksi.

Lock:advisory

Sebuah proses backend meminta kunci advisory dan menunggunya. Untuk informasi selengkapnya, lihat Lock:advisory.

Lock:extend

Sebuah proses backend menunggu kunci dirilis agar dapat memperluas relasi. Kunci ini diperlukan karena hanya satu proses backend yang dapat memperluas relasi pada satu waktu. Untuk informasi selengkapnya, lihat Lock:extend.

Lock:frozenid

Sebuah proses menunggu untuk memperbarui `pg_database.datfrozenxid` dan `pg_database.datminmxid`.

Lock:object

Sebuah proses menunggu untuk mendapatkan kunci pada objek basis data nonrelasi.

Lock:page

Sebuah proses menunggu untuk mendapatkan kunci pada halaman relasi.

Lock:Relation

Sebuah proses backend menunggu untuk mendapatkan kunci pada relasi yang dikunci oleh transaksi lain. Untuk informasi selengkapnya, lihat Lock:Relation.

Lock:spectoken

Sebuah proses menunggu untuk mendapatkan kunci penyisipan spekulatif.

Lock:speculative token

Sebuah proses menunggu untuk mendapatkan kunci penyisipan spekulatif.

Lock:transactionid

Sebuah transaksi menunggu kunci tingkat baris. Untuk informasi selengkapnya, lihat Lock:transactionid.

Lock:tuple

Sebuah proses backend menunggu untuk mendapatkan kunci pada tuple sementara proses backend lain memegang kunci yang bertentangan pada tuple yang sama. Untuk informasi selengkapnya, lihat Lock:tuple.

Lock:userlock

Sebuah proses menunggu untuk mendapatkan kunci pengguna.

Lock:virtualxid

Sebuah proses menunggu untuk mendapatkan kunci ID transaksi virtual.

LWLock:AddinShmemInit

Sebuah proses menunggu untuk mengelola alokasi ruang ekstensi dalam memori bersama.

LWLock:AddinShmemInitLock

Sebuah proses menunggu untuk mengelola alokasi ruang dalam memori bersama.

LWLock:asinkron

Sebuah proses menunggu I/O pada buffer asinkron (notify).

LWLock:AsyncCtlLock

Sebuah proses menunggu untuk membaca atau memperbarui status notifikasi bersama.

LWLock:AsyncQueueLock

Sebuah proses menunggu untuk membaca atau memperbarui pesan notifikasi.

LWLock:AuroraOptimizedReadsCacheMapping

Sebuah proses menunggu untuk mengaitkan blok data dengan halaman di cache bertingkat Optimized Reads.

LWLock:AutoFile

Sebuah proses menunggu untuk memperbarui file `postgresql.auto.conf`.

LWLock:AutoFileLock

Sebuah proses menunggu untuk memperbarui file `postgresql.auto.conf`.

LWLock:Autovacuum

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja autovacuum saat ini.

LWLock:AutovacuumLock

Sebuah pekerja atau peluncur autovacuum menunggu untuk memperbarui atau membaca status pekerja autovacuum saat ini.

LWLock:AutovacuumSchedule

Sebuah proses menunggu untuk memastikan bahwa tabel yang dipilih untuk autovacuum masih memerlukan vacuuming.

LWLock:AutovacuumScheduleLock

Sebuah proses menunggu untuk memastikan bahwa tabel yang telah dipilihnya untuk vacuum masih memerlukan vacuuming.

LWLock:BackendRandomLock

Sebuah proses menunggu untuk menghasilkan angka acak.

LWLock:BackgroundWorker

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja latar belakang.

LWLock:BackgroundWorkerLock

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja latar belakang.

LWLock:BtreeVacuum

Sebuah proses menunggu untuk membaca atau memperbarui informasi terkait vacuum untuk indeks B-tree.

LWLock:BtreeVacuumLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi terkait vacuum untuk indeks B-tree.

LWLock:buffer_content

Sebuah proses backend menunggu untuk memperoleh kunci ringan pada konten buffer memori bersama. Untuk informasi selengkapnya, lihat LWLock:buffer_content (BufferContent).

LWLock:buffer_pemetaan

Sebuah proses backend menunggu untuk mengaitkan blok data dengan buffer di kumpulan buffer bersama. Untuk informasi selengkapnya, lihat LWLock:buffer_mapping.

LWLock:Bufferio

Sebuah proses backend ingin membacakan halaman ke memori bersama. Proses ini menunggu proses lain untuk menyelesaikan I/O-nya untuk halaman. Untuk informasi selengkapnya, lihat LWLock:BufferIO (IPC: Bufferio).

LWLock:Pos pemeriksaan

Sebuah proses menunggu untuk memulai checkpoint.

LWLock:CheckpointLock

Sebuah proses menunggu untuk melakukan checkpoint.

LWLock:CheckpointerComm

Sebuah proses menunggu untuk mengelola permintaan `fsync`.

LWLock:CheckpointerCommLock

Sebuah proses menunggu untuk mengelola permintaan `fsync`.

LWLock:menyumbat

Sebuah proses menunggu I/O pada buffer clog (status transaksi).

LWLock:CLogControlLock

Sebuah proses menunggu untuk membaca atau memperbarui status transaksi.

LWLock:CLogTruncationLock

Sebuah proses menunggu untuk menjalankan `txid_status` atau memperbarui ID transaksi paling lama yang tersedia untuknya.

LWLock:commit_timestamp

Sebuah proses menunggu I/O pada buffer stempel waktu komit.

LWLock:CommitTs

Sebuah proses menunggu untuk membaca atau memperbarui nilai terakhir yang ditetapkan untuk stempel waktu komit transaksi.

LWLock:CommitTsBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk stempel waktu komit.

LWLock:CommitTsControlLock

Sebuah proses menunggu untuk membaca atau memperbarui stempel waktu komit transaksi.

LWLock:CommitTsLock

Sebuah proses menunggu untuk membaca atau memperbarui nilai terakhir yang ditetapkan untuk stempel waktu transaksi.

LWLock: CommitTs SLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk stempel waktu komit.

LWLock:ControlFile

Sebuah proses menunggu untuk membaca atau memperbarui file `pg_control` atau membuat file write-ahead log (WAL) baru.

LWLock:ControlFileLock

Sebuah proses menunggu untuk membaca atau memperbarui file kontrol atau pembuatan file write-ahead log (WAL) baru.

LWLock:DynamicSharedMemoryControl

Sebuah proses menunggu untuk membaca atau memperbarui informasi alokasi memori bersama dinamis.

LWLock:DynamicSharedMemoryControlLock

Sebuah proses menunggu untuk membaca atau memperbarui status memori bersama dinamis.

LWLock:lock_manajer

Sebuah proses backend menunggu untuk menambahkan atau memeriksa kunci untuk proses backend. Atau, proses backend ini menunggu untuk bergabung atau keluar dari grup penguncian yang digunakan oleh kueri paralel. Untuk informasi selengkapnya, lihat LWLock:lock_manager.

LWLock:LockFastPath

Sebuah proses menunggu untuk membaca atau memperbarui informasi kunci jalur cepat proses.

LWLock:LogicalRepWorker

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja replikasi logis.

LWLock:LogicalRepWorkerLock

Sebuah proses menunggu tindakan pada pekerja replikasi logis untuk diselesaikan.

LWLock:LogicalSchemaCache

Sebuah proses memodifikasi cache skema.

LWLock:multixact_anggota

Sebuah proses menunggu I/O pada buffer multixact_member.

LWLock:multixact_offset

Sebuah proses menunggu I/O pada buffer offset multixact.

LWLock:MultiXactGen

Sebuah proses menunggu untuk membaca atau memperbarui status multixact bersama.

LWLock:MultiXactGenLock

Sebuah proses menunggu untuk membaca atau memperbarui status multixact bersama.

LWLock:MultiXactMemberBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk anggota multixact. Untuk informasi selengkapnya, lihat LWLock:MultiXact.

LWLock:MultiXactMemberControlLock

Sebuah proses menunggu untuk membaca atau memperbarui pemetaan anggota multixact.

LWLock: MultiXactMember SLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk anggota multixact. Untuk informasi selengkapnya, lihat LWLock:MultiXact.

LWLock:MultiXactOffsetBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk offset multixact. Untuk informasi selengkapnya, lihat LWLock:MultiXact.

LWLock:MultiXactOffsetControlLock

Sebuah proses menunggu untuk membaca atau memperbarui pemetaan offset multixact.

LWLock: MultiXactOffset SLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk offset multixact. Untuk informasi selengkapnya, lihat LWLock:MultiXact.

LWLock:MultiXactTruncation

Sebuah proses menunggu untuk membaca atau memotong informasi multixact.

LWLock:MultiXactTruncationLock

Sebuah proses menunggu untuk membaca atau memotong informasi multixact.

LWLock:NotifyBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk pesan NOTIFY.

LWLock:NotifyQueue

Sebuah proses menunggu untuk membaca atau memperbarui pesan NOTIFY.

LWLock:NotifyQueueTail

Sebuah proses menunggu untuk memperbarui batas penyimpanan pesan NOTIFY.

LWLock:NotifyQueueTailLock

Sebuah proses menunggu untuk memperbarui batas penyimpanan pesan notifikasi.

LWLock:NotifySLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk pesan NOTIFY.

LWLock:OidGen

Sebuah proses menunggu untuk mengalokasikan ID objek (OID) baru.

LWLock:OidGenLock

Sebuah proses menunggu untuk mengalokasikan atau menetapkan objek ID (OID).

LWLock:oldserxid

Sebuah proses menunggu I/O pada buffer oldserxid.

LWLock:OldSerXidLock

Sebuah proses menunggu untuk membaca atau mencatat transaksi serialisasi yang bertentangan.

LWLock:OldSnapshotTimeMap

Sebuah proses menunggu untuk membaca atau memperbarui informasi kontrol snapshot lama.

LWLock:OldSnapshotTimeMapLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi kontrol snapshot lama.

LWLock:parallel_append

Sebuah proses menunggu untuk memilih subrencana berikutnya selama eksekusi rencana append paralel.

LWLock:parallel_hash_join

Sebuah proses menunggu untuk mengalokasikan atau menukar potongan memori atau memperbarui penghitung selama eksekusi rencana hash paralel.

LWLock:parallel_query_dsa

Sebuah proses menunggu kunci pada alokasi memori bersama dinamis untuk kueri paralel.

LWLock:ParallelAppend

Sebuah proses menunggu untuk memilih subrencana berikutnya selama eksekusi rencana append paralel.

LWLock:ParallelHashJoin

Sebuah proses menunggu untuk menyinkronkan pekerja selama eksekusi rencana untuk hash join paralel.

Lwlock: DSA ParallelQuery

Sebuah proses menunggu alokasi memori bersama dinamis untuk kueri paralel.

Lwlock: DSA PerSession

Sebuah proses menunggu alokasi memori bersama dinamis untuk kueri paralel.

Lwlock: PerSessionRecordType

Sebuah proses menunggu untuk mengakses informasi kueri paralel tentang tipe komposit.

Lwlock: PerSessionRecordTypmod

Sebuah proses menunggu untuk mengakses informasi kueri paralel tentang pengubah tipe yang mengidentifikasi jenis catatan anonim.

Lwlock: PerXactPredicateList

Sebuah proses menunggu untuk mengakses daftar kunci predikat yang dipegang oleh transaksi yang dapat diserialisasi saat ini selama kueri paralel.

Lwlock:predicate_lock_manager

Sebuah proses menunggu untuk menambahkan atau memeriksa informasi kunci predikat.

Lwlock: PredicateLockManager

Sebuah proses menunggu untuk mengakses informasi kunci predikat yang digunakan oleh transaksi yang dapat diserialisasi.

Lwlock:proc

Sebuah proses menunggu untuk membaca atau memperbarui informasi kunci jalur cepat.

LWLock:ProcArray

Sebuah proses menunggu untuk mengakses struktur data per proses bersama (biasanya, untuk mendapatkan snapshot atau melaporkan ID transaksi sesi).

LWLock:ProcArrayLock

Sebuah proses menunggu untuk mendapatkan snapshot atau penghapusan ID transaksi di akhir transaksi.

LWLock:RelationMapping

Sebuah proses menunggu untuk membaca atau memperbarui file `pg_filenode.map` (digunakan untuk melacak penetapan simpul file untuk katalog sistem tertentu).

LWLock:RelationMappingLock

Sebuah proses sedang menunggu untuk memperbarui file peta relasi yang digunakan untuk menyimpan catalog-to-file-node pemetaan.

LWLock:RelCacheInit

Sebuah proses menunggu untuk membaca atau memperbarui file `pg_internal.init` (file inisialisasi cache relasi).

LWLock:RelCacheInitLock

Sebuah proses menunggu untuk membaca atau menulis file inisialisasi cache relasi.

LWLock:replication_origin

Sebuah proses menunggu untuk membaca atau memperbarui progres replikasi.

LWLock:replication_slot_io

Sebuah proses menunggu I/O pada slot replikasi.

LWLock:ReplicationOrigin

Sebuah proses menunggu untuk membuat, menghapus, atau menggunakan asal replikasi.

LWLock:ReplicationOriginLock

Sebuah proses menunggu untuk mengatur, menghapus, atau menggunakan asal replikasi.

LWLock:ReplicationOriginState

Sebuah proses menunggu untuk membaca atau memperbarui progres satu asal replikasi.

LWLock:ReplicationSlotAllocation

Sebuah proses menunggu untuk mengalokasikan atau membebaskan slot replikasi.

LWLock:ReplicationSlotAllocationLock

Sebuah proses menunggu untuk mengalokasikan atau membebaskan slot replikasi.

LWLock:ReplicationSlotControl

Sebuah proses menunggu untuk membaca atau memperbarui status slot replikasi.

LWLock:ReplicationSlotControlLock

Sebuah proses menunggu untuk membaca atau memperbarui status slot replikasi.

LWLock: ReplicationSlot IO

Sebuah proses menunggu I/O pada slot replikasi.

LWLock:SerialBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk pertentangan transaksi yang dapat diserialisasi.

LWLock:SerializableFinishedList

Sebuah proses menunggu untuk mengakses daftar transaksi yang dapat diserialisasi yang selesai.

LWLock:SerializableFinishedListLock

Sebuah proses menunggu untuk mengakses daftar transaksi yang dapat diserialisasi yang selesai.

LWLock:SerializablePredicateList

Sebuah proses menunggu untuk mengakses daftar kunci predikat yang dipegang oleh transaksi yang dapat diserialisasi.

LWLock:SerializablePredicateLockListLock

Sebuah proses menunggu untuk melakukan operasi pada daftar kunci yang dipegang oleh transaksi yang dapat diserialisasi.

LWLock:SerializableXactHash

Sebuah proses menunggu untuk membaca atau memperbarui informasi tentang transaksi serialisasi.

LWLock:SerializableXactHashLock

Sebuah proses menunggu untuk mengambil atau menyimpan informasi tentang transaksi yang dapat diserialisasi.

LWLock:SerialSLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk pertentangan transaksi yang dapat diserialisasi.

LWLock:SharedTidBitmap

Sebuah proses menunggu untuk mengakses bitmap pengidentifikasi tuple bersama (TID) selama pemindaian indeks bitmap paralel.

LWLock:SharedTupleStore

Sebuah proses menunggu untuk mengakses penyimpanan tuple bersama selama kueri paralel.

LWLock:ShmemIndex

Sebuah proses menunggu untuk menemukan atau mengalokasikan ruang dalam memori bersama.

LWLock:ShmemIndexLock

Sebuah proses menunggu untuk menemukan atau mengalokasikan ruang dalam memori bersama.

LWLockBacaSInval:

Sebuah proses menunggu untuk mengambil pesan dari antrean invalidasi katalog bersama.

LWLock:SInvalReadLock

Sebuah proses menunggu untuk mengambil atau menghapus pesan dari antrean invalidasi bersama.

LWLockTulisSInval:

Sebuah proses menunggu untuk menambahkan pesan ke antrean invalidasi katalog bersama.

LWLock:SInvalWriteLock

Sebuah proses menunggu untuk menambahkan pesan dalam antrean invalidasi bersama.

LWLock:subtrans

Sebuah proses menunggu I/O pada buffer subtransaksi.

LWLock:SubtransBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk subtransaksi.

LWLock:SubtransControlLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi subtransaksi.

LWLock:Subtransslru

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk subtransaksi.

LWLock:SyncRep

Sebuah proses menunggu untuk membaca atau memperbarui informasi tentang status replikasi sinkron.

LWLock:SyncRepLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi tentang replika sinkron.

LWLock:SyncScan

Sebuah proses menunggu untuk memilih lokasi awal pemindaian tabel yang disinkronkan.

LWLock:SyncScanLock

Sebuah proses menunggu untuk mendapatkan lokasi awal pemindaian pada tabel untuk pemindaian yang disinkronkan.

LWLock:TablespaceCreate

Sebuah proses menunggu untuk membuat atau menghapus ruang tabel.

LWLock:TablespaceCreateLock

Sebuah proses menunggu untuk membuat atau menghapus ruang tabel.

LWLock:tbm

Sebuah proses menunggu kunci iterator bersama pada tree bitmap (TBM).

LWLock:TwoPhaseState

Sebuah proses menunggu untuk membaca atau memperbarui status transaksi yang disiapkan.

LWLock:TwoPhaseStateLock

Sebuah proses menunggu untuk membaca atau memperbarui status transaksi yang disiapkan.

LWLock:wal_insert

Sebuah proses menunggu untuk menyisipkan write-ahead log (WAL) ke dalam buffer memori.

LWLock: WALBuf Pemetaan

Sebuah proses menunggu untuk mengganti halaman dalam buffer write-ahead log (WAL).

LWLock:WALBufMappingLock

Sebuah proses menunggu untuk mengganti halaman dalam buffer write-ahead log (WAL).

LWLock:WALInsert

Sebuah proses menunggu untuk menyisipkan data write-ahead log (WAL) ke dalam buffer memori.

LWLock:WALWrite

Sebuah proses menunggu buffer write-ahead log (WAL) untuk ditulis ke disk.

LWLock: WALWrite Kunci

Sebuah proses menunggu buffer write-ahead log (WAL) untuk ditulis ke disk.

LWLock:WrapLimitsVacuum

Sebuah proses menunggu untuk memperbarui batas ID transaksi dan konsumsi multixact.

LWLock:WrapLimitsVacuumLock

Sebuah proses menunggu untuk memperbarui batas ID transaksi dan konsumsi multixact.

LWLock:XactBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk status transaksi.

LWLock:XactSLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk status transaksi.

LWLock:XactTruncation

Sebuah proses menunggu untuk menjalankan pg_xact_status atau memperbarui ID transaksi paling lama yang tersedia untuknya.

LWLock:XidGen

Sebuah proses menunggu untuk mengalokasikan ID transaksi baru.

LWLock:XidGenLock

Sebuah proses menunggu untuk mengalokasikan atau menetapkan ID transaksi.

Batas waktu: BaseBackupThrottle

Sebuah proses menunggu selama pencadangan dasar saat aktivitas throttling.

Batas waktu: PgSleep

Sebuah proses backend telah memanggil fungsi pg_sleep dan menunggu batas waktu tidur berakhir. Untuk informasi selengkapnya, lihat Timeout:PgSleep.

Batas waktu: RecoveryApplyDelay

Sebuah proses menunggu untuk menerapkan write-ahead log (WAL) selama pemulihan karena pengaturan penundaan.

Batas waktu: RecoveryRetrieveRetryInterval

Sebuah proses menunggu selama pemulihan ketika data write-ahead log (WAL) tidak tersedia dari sumber apa pun (pg_wal, arsip, atau aliran).

Batas waktu: VacuumDelay

Sebuah proses menunggu di titik penundaan vacuum berbasis biaya.

Untuk daftar lengkap peristiwa tunggu PostgreSQL, lihat The Statistics Collector > Wait Event tables dalam dokumentasi PostgreSQL.

# Pembaruan mesin database untuk Amazon Aurora Postgre SQL

Berikut ini, Anda dapat menemukan informasi tentang rilis dan pembaruan versi SQL mesin Amazon Aurora Postgre. Anda juga dapat menemukan informasi tentang cara meningkatkan mesin Aurora SQL Postgre Anda. Untuk informasi selengkapnya tentang rilis Aurora secara umum, lihat Versi Amazon Aurora.

> **Tip**
>
> Anda dapat meminimalkan waktu henti yang diperlukan untuk meningkatkan klaster DB dengan menggunakan deployment blue/green. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green untuk pembaruan basis data.

Topik


- Mengidentifikasi versi Amazon Aurora Postgre SQL
- Amazon Aurora Postgre SQL rilis dan versi mesin
- Versi ekstensi untuk Amazon Aurora Postgre SQL
- Meningkatkan klaster DB Amazon Aurora PostgreSQL
- Menggunakan rilis dukungan jangka panjang (LTS) Aurora PostgreSQL


## Mengidentifikasi versi Amazon Aurora Postgre SQL

Amazon Aurora menyertakan fitur tertentu yang bersifat umum untuk Aurora dan tersedia untuk semua klaster DB Aurora. Aurora mencakup fitur lain yang khusus untuk mesin basis data tertentu yang didukung Aurora. Fitur-fitur ini hanya tersedia untuk cluster Aurora DB yang menggunakan mesin database itu, seperti Aurora Postgre. SQL

Basis data Aurora biasanya memiliki dua nomor versi, nomor versi mesin basis data dan nomor versi Aurora. Jika rilis Aurora Postgre SQL memiliki nomor versi Aurora, itu disertakan setelah nomor versi mesin dalam daftar. Amazon Aurora Postgre SQL rilis dan versi mesin

Topik


- Nomor versi Aurora
- Nomor versi SQL mesin Postgre

## Nomor versi Aurora

Nomor versi Aurora menggunakan file. *major minor*. *patch*skema penamaan. Versi patch Aurora mencakup perbaikan bug penting yang ditambahkan ke versi minor setelah dirilis. Untuk mempelajari selengkapnya tentang rilis Amazon Aurora mayor, minor, dan patch, lihat Versi mayor Amazon Aurora, Versi minor Amazon Aurora, dan Versi patch Amazon Aurora.

Anda dapat mengetahui nomor versi Aurora dari instans Aurora Postgre SQL DB Anda dengan kueri berikut: SQL

```
postgres=> SELECT aurora_version();
```

Dimulai dengan rilis Postgre SQL versi 13.3, 12.8, 11.13, 10.18, dan untuk semua versi lain yang lebih baru, nomor versi Aurora sejajar lebih dekat dengan versi mesin Postgre. SQL Misalnya, menanyakan cluster Aurora SQL Postgre 13.3 DB mengembalikan yang berikut:

```
aurora_version
----------------
 13.3.1
(1 row)
```

Rilis sebelumnya, seperti Aurora Postgre SQL 10.14 DB cluster, mengembalikan nomor versi yang mirip dengan berikut ini:

```
aurora_version
----------------
 2.7.3
(1 row)
```

## Nomor versi SQL mesin Postgre

Dimulai dengan Postgre SQL 10, versi mesin SQL database Postgre menggunakan file. *major minor*skema penomoran untuk semua rilis. Beberapa contoh termasuk Postgre SQL 10.18, Postgre SQL 12.7, dan Postgre 13.3. SQL

Rilis sebelum Postgre SQL 10 menggunakan a. *major major*. *minor*skema penomoran di mana dua digit pertama membentuk nomor versi utama dan digit ketiga menunjukkan versi minor. Misalnya, Postgre SQL 9.6 adalah versi utama, dengan versi minor 9.6.21 atau 9.6.22 ditunjukkan oleh digit ketiga.

> **Note**
>
> SQLMesin Postgre versi 9.6 tidak lagi didukung. Untuk meningkatkan, lihat Meningkatkan klaster DB Amazon Aurora PostgreSQL. Untuk kebijakan dan lini masa rilis versi, lihatBerapa lama versi mayor Amazon Aurora tetap tersedia.

Anda dapat mengetahui nomor versi mesin SQL database Postgre dengan kueri berikut: SQL

```
postgres=> SELECT version();
```

Untuk cluster Aurora Postgre SQL 13.3 DB, hasilnya adalah sebagai berikut:

```
version
-----------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

## Amazon Aurora Postgre SQL rilis dan versi mesin

Amazon Aurora Postgre SQL -Rilis Edisi Kompatibel diperbarui secara berkala. Pembaruan diterapkan ke cluster Aurora Postgre SQL DB selama jendela pemeliharaan sistem. Kapan pembaruan diterapkan tergantung pada jenis pembaruan, pengaturan jendela Wilayah AWS, dan pemeliharaan untuk cluster DB. Banyak rilis yang terdaftar termasuk nomor versi Postgre dan nomor SQL versi Amazon Aurora. Namun, dimulai dengan rilis Postgre SQL versi 13.3, 12.8, 11.13, 10.18, dan untuk semua versi lain yang lebih baru, nomor versi Aurora tidak digunakan. Untuk mengidentifikasi nomor versi database Aurora Postgre SQL Anda, lihat. Mengidentifikasi versi Amazon Aurora Postgre SQL

Lihat informasi tentang ekstensi dan modul di Versi ekstensi untuk Amazon Aurora Postgre SQL.

> **Note**
>
> Untuk informasi tentang kebijakan dan lini masa rilis versi Amazon Aurora, lihat Berapa lama versi mayor Amazon Aurora tetap tersedia.

Untuk informasi tentang dukungan untuk Amazon Aurora, lihat Amazon. RDS FAQs

Untuk menentukan versi mesin Aurora Postgre SQL DB mana yang tersedia di Wilayah AWS, gunakan perintah. describe-db-engine-versions AWS CLI Sebagai contoh:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

Untuk daftar Wilayah AWS, lihatKetersediaan Wilayah Aurora PostgreSQL.

Untuk detail tentang SQL versi Postgre yang tersedia di Aurora Postgre, lihat Catatan Rilis untuk Aurora PostgreSQL. SQL

## Versi ekstensi untuk Amazon Aurora Postgre SQL

Anda dapat menginstal dan mengkonfigurasi berbagai SQL ekstensi Postgre untuk digunakan dengan cluster Aurora Postgre DB. SQL Misalnya, Anda dapat menggunakan SQL `pg_partman` ekstensi Postgre untuk mengotomatiskan pembuatan dan pemeliharaan partisi tabel. Untuk mempelajari lebih lanjut tentang ini dan ekstensi lain yang tersedia untuk Aurora SQL Postgre, lihat. Menangani ekstensi dan pembungkus data asing

Untuk detail tentang ekstensi Postgre yang didukung di Aurora Postgre, lihat Versi SQL ekstensi untuk Amazon Aurora Postgre SQL di Catatan Rilis untuk Aurora SQL Postgre. SQL

## Meningkatkan klaster DB Amazon Aurora PostgreSQL

Amazon Aurora membuat mesin basis data PostgreSQL versi baru tersedia di Wilayah AWS hanya setelah pengujian ekstensif. Anda dapat meningkatkan klaster DB Aurora PostgreSQL ke versi baru jika tersedia di Wilayah Anda.

Bergantung pada versi Aurora PostgreSQL yang sedang dijalankan oleh klaster DB Anda, tingkatkan ke rilis baru berupa peningkatan minor atau peningkatan mayor. Misalnya, tingkatkan klaster DB Aurora PostgreSQL 11.15 ke Aurora PostgreSQL 13.6 berupa peningkatan versi mayor. Peningkatan klaster DB Aurora PostgreSQL 13.3 ke Aurora PostgreSQL 13.7 berupa peningkatan versi minor. Pada topik berikut, Anda dapat menemukan informasi tentang cara melakukan kedua jenis peningkatan.

Daftar Isi


- Ikhtisar proses peningkatan PostgreSQL Aurora
- Mendapatkan daftar versi yang tersedia di Wilayah AWS

- Melakukan upgrade versi utama
  - Menguji peningkatan klaster DB produksi ke versi mayor baru
  - Rekomendasi pasca-peningkatan
  - Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru
    - Peningkatan mayor untuk basis data global
- Melakukan upgrade versi minor
  - Sebelum melakukan upgrade versi minor
  - Cara melakukan peningkatan versi minor dan menerapkan patch
  - Peningkatan rilis minor dan patching nol-waktu henti
  - Keterbatasan penambalan zero-downtime
  - Meningkatkan mesin Aurora PostgreSQL ke versi minor baru
- Meningkatkan ekstensi PostgreSQL
- Teknik peningkatan blue/green alternatif


## Ikhtisar proses peningkatan PostgreSQL Aurora

Berikut adalah perbedaan antara peningkatan versi mayor dan minor:

Peningkatan dan patch versi minor

Peningkatan dan patch versi minor hanya mencakup perubahan yang kompatibel mundur dengan aplikasi yang ada. Peningkatan dan patch versi minor akan tersedia bagi Anda hanya setelah Aurora PostgreSQL menguji dan menyetujuinya.

Aurora dapat menerapkan upgrade versi minor untuk Anda secara otomatis. Saat Anda membuat cluster Aurora PostgreSQL DB baru, opsi Aktifkan peningkatan versi minor diaktifkan secara default. Kecuali Anda mematikan opsi ini secara manual, Aurora secara berkala menerapkan peningkatan versi minor otomatis selama jendela pemeliharaan terjadwal Anda. Untuk informasi selengkapnya tentang opsi peningkatan versi minor otomatis (AmVU) dan cara mengubah klaster DB Aurora untuk menggunakannya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Jika upgrade versi minor otomatis tidak diaktifkan untuk klaster DB PostgreSQL Aurora Anda, PostgreSQL Aurora Anda tidak secara otomatis ditingkatkan ke versi minor baru. Sebaliknya, jika versi minor baru dirilis di Wilayah AWS dan klaster DB Aurora PostgreSQL Anda menjalankan

versi minor yang lebih lama, Aurora akan meminta Anda untuk melakukan peningkatan. Ini dilakukan dengan menambahkan rekomendasi ke tugas pemeliharaan untuk klaster Anda.

Patch tidak dianggap sebagai peningkatan, dan tidak diterapkan secara otomatis. Aurora PostgreSQL meminta Anda untuk menerapkan patch apa pun dengan menambahkan rekomendasi ke tugas pemeliharaan untuk klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Cara melakukan peningkatan versi minor dan menerapkan patch.

> **Note**
>
> Patch yang menyelesaikan masalah keamanan atau masalah penting lainnya juga ditambahkan sebagai tugas pemeliharaan. Namun, patch ini diperlukan. Pastikan untuk menerapkan patch keamanan ke klaster DB Aurora PostgreSQL Anda jika tersedia dalam tugas pemeliharaan Anda yang tertunda.

Proses peningkatan melibatkan kemungkinan pemadaman singkat karena setiap instans dalam klaster ditingkatkan ke versi baru. Namun, setelah Aurora PostgreSQL versi 14.3.3, 13.7.3, 12.11.3, 11.16.3, 10.21.3 dan rilis lain yang lebih tinggi dari versi minor ini dan versi mayor yang lebih baru, proses peningkatan menggunakan fitur patching nol-waktu henti (ZDP). Fitur ini meminimalkan pemadaman, dan dalam banyak kasus sepenuhnya menghilangkannya. Untuk informasi selengkapnya, lihat Peningkatan rilis minor dan patching nol-waktu henti. Untuk informasi selengkapnya tentang fitur dan batasan ZDP yang didukung, lihat. Keterbatasan penambalan zero-downtime

Peningkatan versi mayor

Tidak seperti peningkatan dan patch versi minor, Aurora PostgreSQL tidak memiliki opsi peningkatan versi mayor utama otomatis. Versi PostgreSQL mayor baru mungkin berisi perubahan basis data yang tidak kompatibel mundur dengan aplikasi yang ada. Fungsionalitas baru dapat menyebabkan aplikasi yang ada tidak berfungsi dengan benar.

Untuk mencegah masalah apa pun, sebaiknya Anda mengikuti proses yang diuraikan dalam Menguji peningkatan klaster DB produksi ke versi mayor baru sebelum meningkatkan instans DB di klaster DB Aurora PostgreSQL Anda. Pertama pastikan aplikasi Anda dapat berjalan di versi baru dengan mengikuti prosedur tersebut. Kemudian, Anda dapat meningkatkan klaster DB Aurora PostgreSQL ke versi baru secara manual.

Proses upgrade melibatkan kemungkinan pemadaman singkat ketika semua instance di cluster ditingkatkan ke versi baru. Proses perencanaan awal juga membutuhkan waktu. Sebaiknya

Anda selalu melakukan tugas peningkatan selama periode pemeliharaan klaster atau saat pengoperasiannya minimum. Untuk informasi selengkapnya, lihat Melakukan upgrade versi utama.

> **Note**
>
> Peningkatan versi minor dan peningkatan versi mayor mungkin melibatkan pemadaman singkat. Oleh karena itu, sebaiknya Anda melakukan atau menjadwalkan peningkatan selama periode pemeliharaan atau selama periode penggunaan rendah lainnya.

Klaster DB Aurora PostgreSQL terkadang memerlukan pembaruan sistem operasi. Pembaruan ini mungkin menyertakan pustaka glibc versi yang lebih baru. Selama pembaruan tersebut, sebaiknya Anda mengikuti pedoman seperti yang dijelaskan dalam .

## Mendapatkan daftar versi yang tersedia di Wilayah AWS

Anda bisa mendapatkan daftar semua versi mesin yang tersedia sebagai target peningkatan untuk cluster Aurora PostgreSQL DB Anda dengan menanyakan Anda menggunakan perintah, sebagai berikut. Wilayah AWS describe-db-engine-versions AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

Misalnya, untuk mengidentifikasi target pemutakhiran yang valid untuk klaster Aurora PostgreSQL versi 12.10 DB, jalankan perintah berikut: AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

Dalam tabel berikut, Anda dapat menemukan target upgrade versi mayor dan minor untuk versi Aurora PostgreSQL DB yang berbeda. Untuk menjaga kompatibilitas, tidak semua versi ditawarkan sebagai target peningkatan. Aurora PostgreSQL memperkenalkan fitur baru dan perbaikan bug dengan setiap rilis versi minor triwulanan. Untuk informasi tentang rilis minor Aurora PostgreSQL, lihat Catatan Rilis untuk Aurora PostgreSQL.

| Versi sumber saat ini | Target peningkatan |
|---|---|
| 17.5 | Tidak ada |
| 17.4 | 17.5, 17.4 |
| 16.9 | 17.5 |
| 16.8 | 17.5, 17.4<br>16.9 |
| 16.6 | 17.5, 17.4<br>16.9, 16,8 |
| 16.4 | 17.5, 17.4<br>16.9, 16.8, 16.6 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 16.3 | 17.5, 17.4<br>16.9, 16.8, 16.6, 16.4 |
| 16.2 | 17.5, 17.4<br>16.9 , 16.8 16.6, 16.4, 16.3 |
| 16.1 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2 |
| 15.13 | 17.5<br>16.9 |
| 15.12 | 17.5, 17.4<br>16.9, 16,8<br>15.13 |
| 15.10 | 17.5, 17.4<br>16.9, 16.8, 16.6<br>15.13, 15.12 |
| 15.8 | 17.5, 17.4<br>16.9, 16.8, 16.6, 16.4<br>15.13, 15.12, 15.10 |
| 15.7 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3<br>15.13, 15.12, 15.10, 15.8 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 15.6 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2<br>15.13 , 15.12, 15.10, 15.8, 15.7 |
| 15.5 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6 |
| 15.4 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5 |
| 15.3 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 |
| 15.2 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3 |
| 14.18 | 17.5<br>16.9<br>15.13 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 14.17 | 17.5, 17.4<br>16.9, 16,8<br>15.13, 15.12<br>14.18 |
| 14.15 | 17.5, 17.4<br>16.9, 16.8, 16.6<br>15.13, 15.12, 15.10<br>14.18, 14.17 |
| 14.13 | 17.5, 17.4<br>16.9, 16.8, 16.6, 16.4<br>15.13, 15.12, 15.10, 15.8<br>14.18, 14.17, 14.15 |
| 14.12 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3<br>15.13 , 15.12, 15.10, 15.8, 15.7<br>14.18, 14.17, 14.15, 14.13 |
| 14.11 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2<br>15.13 , 15.12, 15.10, 15.8, 15.6<br>14.18 , 14.17, 14.15, 14.13, 14.12 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 14.10 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11 |
| 14.9 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10 |
| 14.8 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 |
| 14.7 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8 |
| 14.6 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 14.5 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 |
| 14.4 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5 |
| 14.3 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5, 14.4 |
| 13.21 | 17.5<br><br>16.9<br><br>15.13<br><br>14.18 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 13.20 | 17.5, 17.4<br>16.9, 16,8<br>15.13, 15.12<br>14.18, 14.17<br>13.21 |
| 13.18 | 17.5, 17.4<br>16.9, 16.8, 16.6<br>15.13, 15.12, 15.10<br>14.18, 14.17, 14.15<br>13.21, 13.20 |
| 13.16 | 17.5, 17.4<br>16.9, 16.8, 16.6, 16.4<br>15.13, 15.12, 15.10, 15.8<br>14.18, 14.17, 14.15, 14.13<br>13.21, 13.20, 13.18 |
| 13.15 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3<br>15.13 , 15.12, 15.10, 15.8, 15.7<br>14.18 , 14.17, 14.15, 14.13, 14.12<br>13.21, 13.20, 13.18, 13.16 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 13.14 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6<br>14.18 , 14.17, 14.15, 14.13, 14.11<br>13.21 , 13.20, 13.18, 13.16, 13.15 |
| 13.13 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14 |
| 13.12 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13 |
| 13.11 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 13.10 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11 |
| 13.9 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10 |
| 13.8 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 13.7 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5, 14.4, 14.3<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8 |
| 12.22 | 17.5, 17.4<br><br>16.9, 16.8, 16.6<br><br>15.13, 15.12, 15.10<br><br>14.18, 14.17, 14.15<br><br>13.21, 13.20, 13.18 |
| 12.20 | 17.5, 17.4<br><br>16.9, 16.8, 16.6, 16.4<br><br>15.13, 15.12, 15.10, 15.8<br><br>14.15, 14.13<br><br>13.21, 13.20, 13.18, 13.16<br><br>12.22 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 12.19 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3<br>15.13 , 15.12, 15.10, 15.8, 15.7<br>14.18 , 14.17, 14.15, 14.13, 14.12<br>13.21 , 13.20, 13.18, 13.16, 13.15<br>12.22, 12.20 |
| 12.18 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14<br>12.22, 12.20, 12.19 |
| 12.17 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13<br>12.22, 12.20, 12.19, 12.18 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 12.16 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12<br>12.22 , 12.20, 12.19, 12.18, 12.17 |
| 12.15 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11<br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16 |
| 12.14 | 17.5, 17.4<br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.5, 15.4, 15.3, 15.2<br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7<br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10<br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16, 12.15 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 12.13 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9<br><br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14 |
| 12.12 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8<br><br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 12.11 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.5, 14.4, 14.3<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8, 13.7<br><br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12 |
| 12.9 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7<br><br>13.21 , 13.20, 13.18, 13.16, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8, 13.7<br><br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12, 12.11 |
| 11.21 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12<br><br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16 |

| Versi sumber saat ini | Target peningkatan |
| --- | --- |
| 11.9 | 17.5, 17.4<br><br>16.9 , 16.8, 16.6, 16.4, 16.3, 16.2, 16.1<br><br>15.13 , 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2<br><br>14.18 , 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6<br><br>13.21 , 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9<br><br>12.22 , 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12, 12.11, 12.09<br><br>11.21 |

Untuk versi apa pun yang Anda pertimbangkan, selalu periksa ketersediaan kelas instans DB klaster Anda. Misalnya, `db.r4` tidak didukung untuk Aurora PostgreSQL 13. Jika klaster DB Aurora PostgreSQL Anda saat ini menggunakan kelas instans db.r4, Anda harus migrasi ke db.r5 sebelum mencoba peningkatan. Untuk informasi selengkapnya tentang kelas instans DB, termasuk kelas yang berbasis Graviton2 dan kelas yang berbasis Intel, lihat Kelas instans Amazon Aurora DB.

## Melakukan upgrade versi utama

Peningkatan versi mayor mungkin berisi perubahan basis data yang tidak kompatibel mundur dengan basis data versi sebelumnya. Fungsionalitas baru dalam versi baru dapat menyebabkan aplikasi yang ada tidak berfungsi dengan benar. Untuk menghindari masalah, Amazon Aurora tidak menerapkan peningkatan versi mayor secara otomatis. Sebaliknya, sebaiknya Anda merencanakan peningkatan versi utama secara cermat dengan mengikuti langkah-langkah berikut:

1. Pilih versi mayor yang diinginkan dari daftar target yang tersedia dari yang tercantum untuk versi Anda dalam tabel. Anda bisa mendapatkan daftar versi yang tepat yang tersedia di versi Anda saat ini dengan menggunakan AWS CLI. Wilayah AWS Untuk detailnya, lihat Mendapatkan daftar versi yang tersedia di Wilayah AWS.

2. Verifikasi apakah aplikasi Anda berfungsi seperti yang diharapkan saat deployment uji coba versi baru. Untuk informasi tentang proses lengkapnya, lihat Menguji peningkatan klaster DB produksi ke versi mayor baru.
3. Setelah memverifikasi bahwa aplikasi Anda berfungsi seperti yang diharapkan saat deployment uji coba, Anda dapat meningkatkan klaster. Untuk detailnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.

> **Note**
>
> Anda dapat melakukan peningkatan versi mayor dari versi berbasis Babelfish for Aurora PostgreSQL 13 mulai dari versi 13.6 ke versi berbasis Aurora PostgreSQL 14 mulai dari versi 14.6. Babelfish for Aurora PostgreSQL 13.4 dan 13.5 tidak mendukung peningkatan versi mayor.

Anda bisa mendapatkan daftar versi mesin yang tersedia sebagai target peningkatan versi utama untuk cluster DB PostgreSQL Aurora Anda dengan menanyakan Anda menggunakan perintah, sebagai berikut. Wilayah AWS describe-db-engine-versions AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

Dalam beberapa kasus, versi yang ingin ditingkatkan bukanlah target untuk versi Anda saat ini. Dalam kasus seperti itu, gunakan informasi dalam versions table untuk melakukan peningkatan versi minor hingga klaster Anda memiliki versi dengan target yang Anda pilih di baris targetnya.

Menguji peningkatan klaster DB produksi ke versi mayor baru

Setiap versi mayor baru mencakup penyempurnaan pada pengoptimal kueri yang dirancang untuk meningkatkan performa. Namun, beban kerja Anda mungkin mencakup kueri yang menyebabkan rencana berperforma lebih buruk di versi baru. Itulah mengapa kami menyarankan Anda menguji dan meninjau performa sebelum melakukan peningkatan dalam produksi. Anda dapat mengelola stabilitas rencana kueri di seluruh versi dengan menggunakan ekstensi Manajemen Rencana Kueri (QPM), seperti yang dijelaskan dalam Memastikan stabilitas rencana setelah peningkatan versi mayor.

Sebelum meningkatkan klaster DB Aurora PostgreSQL produksi ke versi mayor baru, sebaiknya Anda menguji peningkatan untuk memverifikasi bahwa semua aplikasi Anda berfungsi dengan benar:

1. Siapkan grup parameter yang kompatibel dengan versi.

   Jika menggunakan instans DB kustom atau grup parameter klaster DB, Anda dapat memilih dari dua opsi:

   a. Tentukan instans DB default, grup parameter klaster DB, atau keduanya untuk versi mesin DB baru.

   b. Buat grup parameter kustom untuk versi mesin DB baru.

   Jika Anda menautkan instans DB baru atau grup parameter klaster DB sebagai bagian dari permintaan peningkatan, pastikan untuk mem-boot ulang basis data setelah proses peningkatan selesai untuk menerapkan parameter. Jika instans DB perlu di-reboot untuk menerapkan perubahan grup parameter, status grup parameter instans akan menampilkan `pending-reboot`. Anda dapat melihat status grup parameter suatu instans di konsol atau menggunakan perintah CLI, seperti describe-db-instances atau describe-db-clusters.
2. Periksa database yang tidak valid dan lepaskan semua yang ada.

   `datconnlimit`Kolom dalam `pg_database` katalog mencakup nilai `-2` untuk menandai sebagai tidak valid setiap database yang terputus selama operasi. `DROP DATABASE` Gunakan kueri berikut untuk memeriksa apakah ada database yang tidak valid.

```
SELECT
    datname
```

```
FROM
    pg_database
WHERE
    datconnlimit = - 2;
```

Jika query mengembalikan nama database, database ini tidak valid. Gunakan `DROP DATABASE invalid_db_name` pernyataan untuk menjatuhkan database yang tidak valid. Anda dapat menggunakan pernyataan dinamis berikut untuk menghapus semua database yang tidak valid.

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

3. Periksa penggunaan yang tidak didukung:

   - Lakukan atau putar kembali semua transaksi terbuka yang disiapkan sebelum mencoba peningkatan. Anda dapat menggunakan kueri berikut untuk memverifikasi bahwa tidak ada transaksi terbuka yang disiapkan di instans Anda.

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

   - Hapus semua penggunaan jenis data reg* sebelum mencoba peningkatan. Kecuali untuk `regtype` dan `regclass`, Anda tidak dapat meningkatkan jenis data reg*. Utilitas pg_upgrade (yang digunakan oleh Amazon Aurora untuk melakukan peningkatan) tidak dapat mempertahankan jenis data ini. Untuk mempelajari selengkapnya tentang utilitas ini, lihat pg_upgrade dalam dokumentasi PostgreSQL.

     Untuk memverifikasi bahwa jenis data reg* yang tidak didukung tidak digunakan, gunakan kueri berikut untuk setiap basis data.

     ```
     SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
      pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid
           AND NOT a.attisdropped
           AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                              'pg_catalog.regprocedure'::pg_catalog.regtype,
                              'pg_catalog.regoper'::pg_catalog.regtype,
                              'pg_catalog.regoperator'::pg_catalog.regtype,
     ```

```
                        'pg_catalog.regconfig'::pg_catalog.regtype,
                        'pg_catalog.regdictionary'::pg_catalog.regtype)
      AND c.relnamespace = n.oid
      AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

- Jika Anda meningkatkan Aurora PostgreSQL versi 10.18 atau klaster DB yang lebih tinggi yang memiliki `pgRouting` ekstensi terinstal, lepaskan ekstensi sebelum ditingkatkan ke versi 12.4 atau lebih tinggi.

  Jika Anda meningkatkan Aurora PostgreSQL versi 10.x yang memiliki ekstensi `pg_repack` versi 1.4.3 terinstal, lepaskan ekstensi sebelum ditingkatkan ke versi yang lebih tinggi.

4. Periksa basis data template1 dan template0.

   Agar peningkatan berhasil, basis data template 1 dan template 0 harus ada dan harus tercantum sebagai template. Untuk melakukannya, gunakan perintah berikut:

```
SELECT datname, datistemplate FROM pg_database;

datname    | datistemplate
-----------+---------------
template0  | t
rdsadmin   | f
template1  | t
postgres   | f
```

   Dalam output perintah, nilai `datistemplate` untuk basis data template1 dan template0 seharusnya `t`.

5. Hapus slot replikasi logis.

   Proses peningkatan tidak dapat dilanjutkan jika klaster DB Aurora PostgreSQL menggunakan slot replikasi logis apa pun. Slot replikasi logis biasanya digunakan untuk tugas migrasi data jangka pendek, seperti migrasi data menggunakan AWS DMS atau untuk mereplikasi tabel dari database ke danau data, alat BI, atau target lainnya. Sebelum meningkatkan, pastikan Anda mengetahui tujuan dari setiap slot replikasi logis yang ada, dan konfirmasikan bahwa menghapusnya tidak akan jadi masalah. Anda dapat memeriksa slot replikasi logis menggunakan kueri berikut:

```
SELECT * FROM pg_replication_slots;
```

Jika slot replikasi logis masih digunakan, Anda tidak boleh menghapusnya, dan Anda tidak dapat melanjutkan dengan peningkatan. Namun, jika slot replikasi logis tidak diperlukan, Anda dapat menghapusnya menggunakan SQL berikut:

```
SELECT pg_drop_replication_slot(slot_name);
```

Skenario replikasi logis yang menggunakan ekstensi `pglogical` juga harus memiliki slot yang dijatuhkan dari simpul penerbit untuk peningkatan versi utama yang berhasil pada simpul tersebut. Namun, Anda dapat memulai ulang proses replikasi dari simpul pelanggan setelah peningkatan. Untuk informasi selengkapnya, lihat Membangun kembali replikasi logis setelah peningkatan besar.

6. Lakukan pencadangan.

   Proses peningkatan membuat snapshot klaster DB dari klaster DB Anda selama peningkatan. Jika Anda juga ingin melakukan pencadangan manual sebelum proses peningkatan, lihat Membuat snapshot klaster DB untuk informasi selengkapnya.

7. Tingkatkan ekstensi tertentu ke versi terbaru yang tersedia sebelum melakukan peningkatan versi utama. Ekstensi yang akan diperbarui meliputi:

   - `pgRouting`
   - `postgis_raster`
   - `postgis_tiger_geocoder`
   - `postgis_topology`
   - `address_standardizer`
   - `address_standardizer_data_us`

   Jalankan perintah berikut untuk setiap ekstensi yang baru diinstal.

   ```
   ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
   ```

   Untuk informasi selengkapnya, lihat Meningkatkan ekstensi PostgreSQL. Untuk mempelajari selengkapnya tentang cara meningkatkan PostGIS, lihat Langkah 6: Tingkatkan GIS ekstensi Post.

8. Jika Anda meningkatkan ke versi 11.x, buang ekstensi yang tidak didukung sebelum melakukan peningkatan versi utama. Ekstensi yang perlu dibuang meliputi:

   - `chkpass`

- `tsearch2`

9. Buang jenis data unknown, bergantung pada versi target Anda.

   PostgreSQL versi 10 tidak mendukung jenis data `unknown`. Jika basis data versi 9.6 menggunakan jenis data `unknown`, tingkatkan ke versi 10 akan menampilkan pesan kesalahan seperti berikut.

   ```
   Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
   The instance could not be upgraded because the 'unknown' data type is used in user
    tables.
   Please remove all usages of the 'unknown' data type and try again."
   ```

   Untuk menemukan jenis data `unknown` di basis data Anda sehingga Anda dapat menghapus kolom tersebut atau mengubahnya ke jenis data yang didukung, gunakan kode SQL berikut untuk setiap basis data.

   ```
   SELECT n.nspname, c.relname, a.attname
       FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
       pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
       a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
       c.relkind IN ('r','m','c') AND
       c.relnamespace = n.oid AND
       n.nspname !~ '^pg_temp_' AND
       n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
    'information_schema');
   ```

10. Lakukan peningkatan dry-run.

    Sebaiknya Anda menguji peningkatan versi utama pada duplikat basis data produksi sebelum mencoba peningkatan pada basis data produksi Anda. Anda dapat memantau rencana eksekusi pada instans pengujian duplikat untuk setiap kemungkinan regresi rencana eksekusi dan untuk mengevaluasi performanya. Untuk membuat instans pengujian duplikat, Anda dapat memulihkan basis data Anda dari snapshot terakhir atau mengkloning basis data Anda. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot atau Mengkloning volume untuk klaster DB Amazon Aurora.

    Untuk informasi selengkapnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.

11. Tingkatkan instans produksi Anda.

Jika peningkatan versi utama operasi dry-run berhasil, Anda seharusnya dapat meningkatkan basis data produksi dengan percaya diri. Untuk informasi selengkapnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.

> **Note**
>
> Selama proses peningkatan, Aurora PostgreSQL mengambil snapshot klaster DB jika periode retensi cadangan klaster lebih besar dari 0. Anda tidak dapat melakukan point-in-time pemulihan klaster Anda selama proses ini. Kemudian, Anda dapat melakukan point-in-time pemulihan ke waktu sebelum pemutakhiran dimulai dan setelah snapshot otomatis instans Anda selesai. Namun, Anda tidak dapat melakukan point-in-time pemulihan ke versi minor sebelumnya.

Untuk informasi tentang peningkatan yang sedang berlangsung, Anda dapat menggunakan Amazon RDS untuk melihat dua log yang dibuat oleh utilitas pg_upgrade. Log tersebut adalah `pg_upgrade_internal.log` dan `pg_upgrade_server.log`. Amazon Aurora menambahkan stempel waktu ke nama file untuk log ini. Anda dapat melihat log ini sebagaimana Anda dapat melihat log lainnya. Untuk informasi selengkapnya, lihat Memantau log Amazon Aurora Amazon.

12 Meningkatkan ekstensi PostgreSQL. Proses peningkatan PostgreSQL tidak meningkatkan ekstensi PostgreSQL apa pun. Untuk informasi selengkapnya, lihat Meningkatkan ekstensi PostgreSQL.

Rekomendasi pasca-peningkatan

Setelah Anda menyelesaikan peningkatan versi utama, sebaiknya ikuti langkah berikut:

- Jalankan operasi `ANALYZE` untuk menyegarkan tabel `pg_statistic`. Anda harus melakukannya untuk setiap basis data pada semua instans DB PostgreSQL Anda. Statistik pengoptimal tidak ditransfer selama peningkatan versi utama, jadi Anda perlu membuat ulang semua statistik untuk menghindari masalah performa. Jalankan perintah tanpa parameter apa pun untuk menghasilkan statistik untuk semua tabel reguler dalam basis data saat ini, sebagai berikut:

```
ANALYZE VERBOSE;
```

Bendera VERBOSE bersifat opsional, tetapi kemajuannya akan ditampilkan jika digunakan. Untuk informasi selengkapnya, lihat ANALYZE di dokumentasi PostgreSQL.

> **Note**
>
> Jalankan ANALYZE pada sistem Anda setelah peningkatan untuk menghindari masalah performa.

- Jika Anda melakukan peningkatan ke PostgreSQL versi 10, jalankan REINDEX pada setiap indeks hash yang dimiliki. Indeks hash diubah di versi 10 dan harus dibuat ulang. Untuk menemukan indeks hash yang tidak valid, jalankan SQL berikut untuk setiap basis data yang berisi indeks hash.

```
SELECT idx.indrelid::regclass AS table_name,
   idx.indexrelid::regclass AS index_name
FROM pg_catalog.pg_index idx
   JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
   JOIN pg_catalog.pg_am am ON am.oid = cls.relam
WHERE am.amname = 'hash'
AND NOT idx.indisvalid;
```

- Sebaiknya Anda menguji aplikasi di basis data yang ditingkatkan dengan beban kerja serupa untuk memverifikasi bahwa semuanya berfungsi sesuai harapan. Setelah peningkatan diverifikasi, Anda dapat menghapus instans pengujian ini.

### Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru

Saat Anda memulai proses peningkatan ke versi mayor baru, Aurora PostgreSQL mengambil snapshot dari klaster DB Aurora sebelum membuat perubahan apa pun pada klaster Anda. Snapshot ini dibuat hanya untuk peningkatan versi mayor, bukan peningkatan versi minor. Saat proses peningkatan selesai, Anda dapat menemukan snapshot ini di antara snapshot manual yang tercantum di bagian Snapshot di konsol RDS. Nama snapshot mencakup `preupgrade` sebagai awalan, nama klaster DB Aurora PostgreSQL Anda, versi sumber, versi target, dan tanggal serta stempel waktu, seperti yang ditunjukkan pada contoh berikut.

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

Setelah peningkatan selesai, Anda dapat menggunakan snapshot yang dibuat dan disimpan Aurora dalam daftar snapshot manual untuk memulihkan klaster DB ke versi sebelumnya, jika perlu.

> **Tip**
>
> Secara umum, snapshot menyediakan banyak cara untuk memulihkan klaster DB Aurora ke berbagai titik waktu. Untuk mempelajari lebih lanjut, lihat Memulihkan dari snapshot klaster DB dan Memulihkan klaster DB ke waktu tertentu. Namun, Aurora PostgreSQL tidak mendukung penggunaan snapshot untuk memulihkan ke versi minor sebelumnya.

Selama proses peningkatan versi mayor, Aurora mengalokasikan volume dan mengkloning klaster DB Aurora PostgreSQL sumber. Jika peningkatan gagal karena alasan apa pun, Aurora PostgreSQL menggunakan klon ini untuk memutar kembali peningkatan. Setelah lebih dari 15 klon dari volume sumber dialokasikan, klon berikutnya menjadi salinan lengkap dan memakan waktu lebih lama. Hal ini juga dapat menyebabkan proses peningkatan memakan waktu lebih lama. Jika Aurora PostgreSQL membatalkan peningkatan, perhatikan hal berikut:

- Anda dapat melihat entri dan metrik penagihan untuk volume awal dan volume kloning yang dialokasikan selama peningkatan. Aurora PostgreSQL membersihkan volume ekstra setelah periode retensi cadangan klaster melampaui waktu peningkatan .
- Salinan snapshot lintas-wilayah berikutnya dari klaster ini akan menjadi salinan lengkap, bukan salinan tambahan.

Untuk meningkatkan instans DB yang membentuk klaster Anda dengan aman, Aurora PostgreSQL menggunakan utilitas pg_upgrade. Setelah peningkatan penulis selesai, setiap instans pembaca mengalami pemadaman singkat saat ditingkatkan ke versi mayor baru. Untuk mempelajari selengkapnya tentang utilitas PostgreSQL ini, lihat pg_upgrade di dokumentasi PostgreSQL.

Anda dapat memutakhirkan cluster Aurora PostgreSQL DB Anda ke versi baru dengan menggunakan, API, atau RDS. AWS Management Console AWS CLI

Konsol

Untuk meningkatkan versi mesin klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin Anda tingkatkan.
3. Pilih Ubah. Halaman Ubah klaster DB akan muncul.

4. Untuk Versi mesin, pilih versi baru.
5. Pilih Lanjutkan dan periksa ringkasan perubahan.
6. Untuk segera menerapkan perubahan, pilih Terapkan segera. Dalam beberapa kasus, pemadaman dapat terjadi jika opsi ini dipilih. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Ubah Klaster untuk menyimpan perubahan.

   Atau pilih Kembali untuk mengedit perubahan atau Batal untuk membatalkan perubahan.

AWS CLI

Untuk memutakhirkan versi mesin cluster DB, gunakan modify-db-cluster AWS CLI perintah. Tentukan parameter berikut:

- `--db-cluster-identifier` – Nama klaster DB.
- `--engine-version` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan AWS CLI describe-db-engine-versionsperintah.
- `--allow-major-version-upgrade` – Tanda yang diperlukan saat parameter `--engine-version` merupakan versi utama yang berbeda dengan versi utama saat ini dari klaster DB.
- `--no-apply-immediately` – Terapkan perubahan selama periode pemeliharaan berikutnya. Untuk segera menerapkan perubahan, gunakan `--apply-immediately`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
```

```
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

API RDS

Untuk meng-upgrade versi mesin dari cluster DB, gunakan DBCluster operasi Modify. Tentukan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB, misalnya *`mydbcluster`*.
- `EngineVersion` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan operasi Deskripsikan DBEngine Versi.
- `AllowMajorVersionUpgrade` – Tanda yang diperlukan saat parameter `EngineVersion` merupakan versi utama yang berbeda dengan versi utama saat ini dari klaster DB.
- `ApplyImmediately` – Apakah akan menerapkan perubahan secara langsung atau selama periode pemeliharaan berikutnya. Untuk segera menerapkan perubahan, tetapkan nilai ke `true`. Untuk menerapkan perubahan selama periode pemeliharaan berikutnya, tetapkan nilai ke `false`.

Peningkatan mayor untuk basis data global

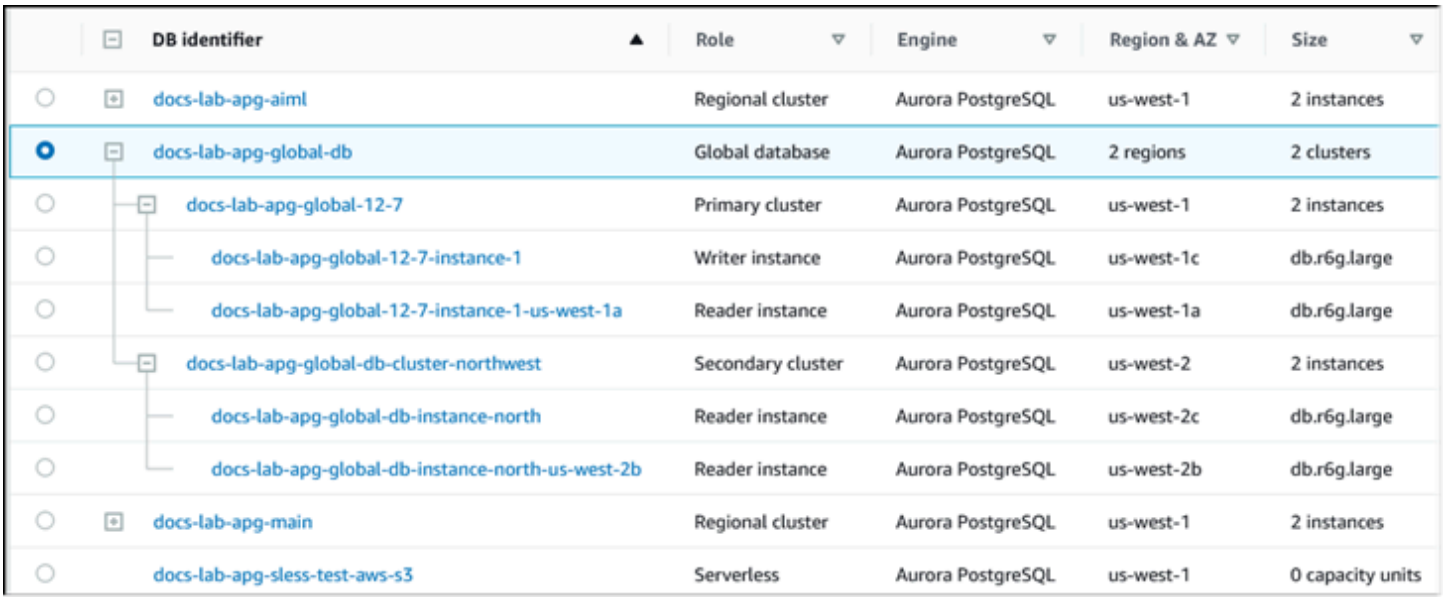
Untuk klaster basis data global Aurora, proses peningkatan akan meningkatkan semua klaster DB yang membentuk basis data global Aurora pada saat yang bersamaan. Itu dilakukan untuk memastikan bahwa masing-masing menjalankan versi Aurora PostgreSQL yang sama. Ini juga memastikan bahwa setiap perubahan pada tabel sistem, format file data, dan sebagainya secara otomatis direplikasi ke semua klaster sekunder.

Untuk meningkatkan klaster basis data global ke Aurora PostgreSQL versi mayor baru, sebaiknya Anda menguji aplikasi Anda pada versi yang ditingkatkan, seperti yang dijelaskan dalam Menguji peningkatan klaster DB produksi ke versi mayor baru. Pastikan untuk menyiapkan grup parameter cluster DB dan pengaturan grup parameter DB Anda untuk masing-masing Wilayah AWS di database global Aurora Anda sebelum peningkatan sebagaimana dirinci. step 1. Menguji peningkatan klaster DB produksi ke versi mayor baru

Jika klaster basis data global Aurora PostgreSQL Anda memiliki sasaran titik pemulihan (RPO) yang ditetapkan untuk parameter `rds.global_db_rpo`-nya, pastikan untuk mengatur ulang parameter sebelum melakukan peningkatan. Proses peningkatan versi mayor tidak berfungsi jika RPO diaktifkan. Secara default, parameter ini dinonaktifkan. Untuk informasi lebih lanjut tentang basis

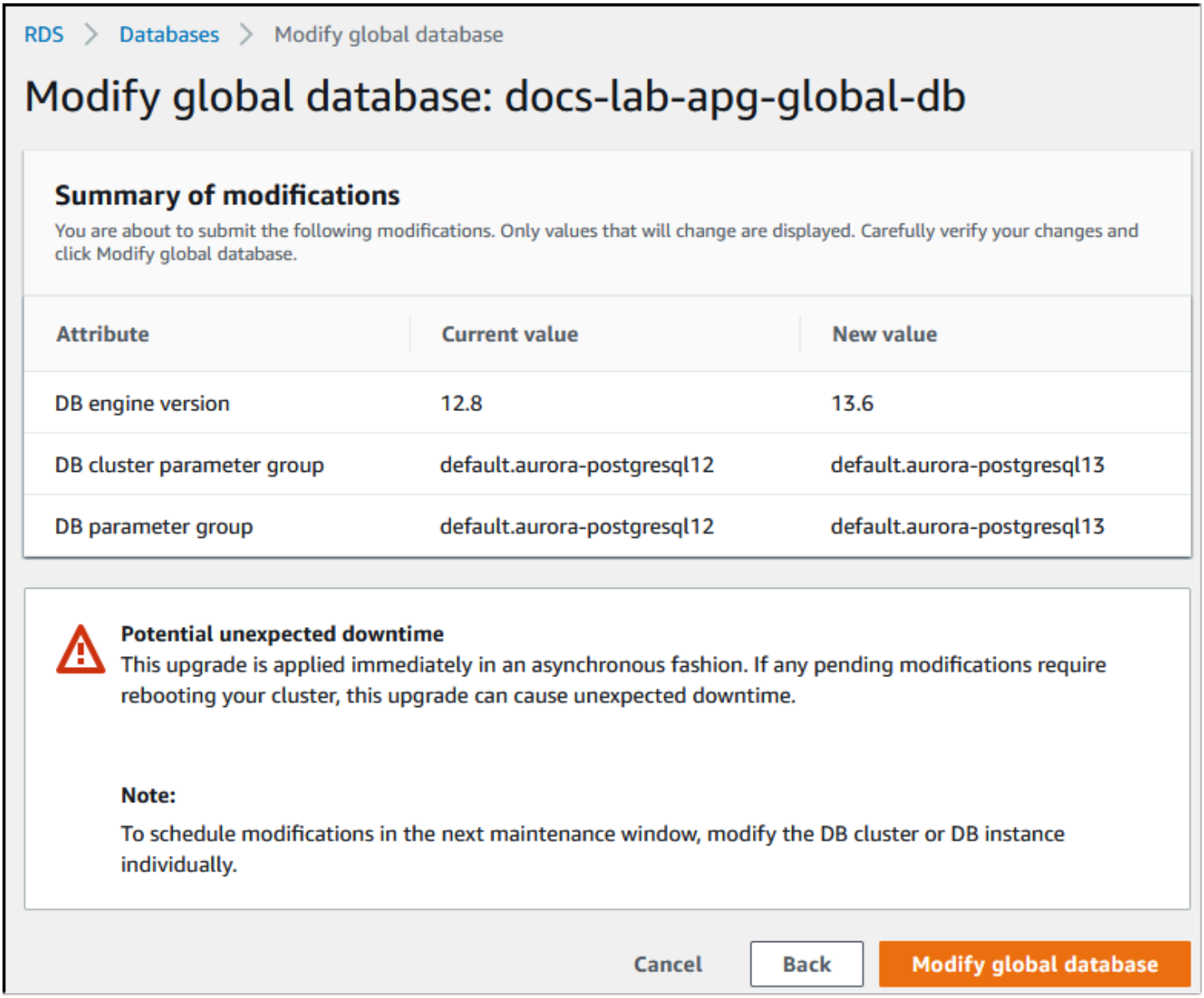
data global Aurora PostgreSQL dan RPO, lihat Mengelola RPOs database global berbasis Aurora PostgreSQL.

Jika Anda memverifikasi bahwa aplikasi Anda dapat berjalan sesuai harapan saat deployment uji coba versi baru, Anda dapat memulai proses peningkatan. Untuk melakukannya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru. Pastikan untuk memilih item tingkat atas dari daftar Basis Data di konsol RDS, Basis data global, seperti yang ditunjukkan pada gambar berikut.



| DB identifier | Role | Engine | Region & AZ | Size |
|---|---|---|---|---|
| docs-lab-apg-aiml | Regional cluster | Aurora PostgreSQL | us-west-1 | 2 instances |
| docs-lab-apg-global-db | Global database | Aurora PostgreSQL | 2 regions | 2 clusters |
| docs-lab-apg-global-12-7 | Primary cluster | Aurora PostgreSQL | us-west-1 | 2 instances |
| docs-lab-apg-global-12-7-instance-1 | Writer instance | Aurora PostgreSQL | us-west-1c | db.r6g.large |
| docs-lab-apg-global-12-7-instance-1-us-west-1a | Reader instance | Aurora PostgreSQL | us-west-1a | db.r6g.large |
| docs-lab-apg-global-db-cluster-northwest | Secondary cluster | Aurora PostgreSQL | us-west-2 | 2 instances |
| docs-lab-apg-global-db-instance-north | Reader instance | Aurora PostgreSQL | us-west-2c | db.r6g.large |
| docs-lab-apg-global-db-instance-north-us-west-2b | Reader instance | Aurora PostgreSQL | us-west-2b | db.r6g.large |
| docs-lab-apg-main | Regional cluster | Aurora PostgreSQL | us-west-1 | 2 instances |
| docs-lab-apg-sless-test-aws-s3 | Serverless | Aurora PostgreSQL | us-west-1 | 0 capacity units |

Seperti halnya modifikasi apa pun, Anda dapat mengonfirmasi bahwa Anda ingin melanjutkan proses saat diminta.




Alih-alih menggunakan konsol, Anda dapat memulai proses peningkatan dengan menggunakan AWS CLI atau RDS API. Seperti halnya konsol, Anda beroperasi pada klaster basis data global Aurora, bukan konstituennya, sebagai berikut:

- Gunakan modify-global-cluster AWS CLI perintah untuk memulai upgrade untuk database global Aurora Anda dengan menggunakan. AWS CLI
- Gunakan ModifyGlobalClusterAPI untuk memulai pemutakhiran.

## Melakukan upgrade versi minor

Anda dapat menggunakan metode berikut untuk meningkatkan versi minor klaster DB atau untuk mem-patch klaster DB:

Topik


- Sebelum melakukan upgrade versi minor
- Cara melakukan peningkatan versi minor dan menerapkan patch
- Peningkatan rilis minor dan patching nol-waktu henti
- Keterbatasan penambalan zero-downtime
- Meningkatkan mesin Aurora PostgreSQL ke versi minor baru


### Sebelum melakukan upgrade versi minor

Kami menyarankan Anda melakukan tindakan berikut untuk mengurangi waktu henti selama upgrade versi minor:

- Pemeliharaan cluster Aurora DB harus dilakukan selama periode lalu lintas rendah. Gunakan Performance Insights untuk mengidentifikasi periode waktu ini agar dapat mengonfigurasi jendela pemeliharaan dengan benar. Untuk informasi selengkapnya tentang Performance Insights, lihat Memantau pemuatan DB dengan Performance Insights di Amazon RDS. Untuk informasi lebih lanjut tentang jendela pemeliharaan cluster DB,Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.
- Gunakan dukungan AWS SDKs itu backoff eksponensial dan jitter sebagai praktik terbaik. Untuk informasi lebih lanjut, lihat Exponential Backoff And Jitter.

### Cara melakukan peningkatan versi minor dan menerapkan patch

Peningkatan dan tambalan versi minor Wilayah AWS hanya tersedia setelah pengujian yang ketat. Sebelum merilis upgrade dan patch, Aurora PostgreSQL menguji untuk memastikan bahwa masalah keamanan yang diketahui, bug, dan masalah lain yang muncul setelah rilis versi komunitas minor tidak mengganggu stabilitas armada Aurora PostgreSQL.

Jika Anda mengaktifkan Aktifkan peningkatan versi minor otomatis, Aurora PostgreSQL secara berkala meningkatkan cluster DB Anda selama jendela pemeliharaan yang ditentukan. Pastikan opsi Aktifkan pemutakhiran versi minor otomatis diaktifkan untuk semua instance di cluster DB PostgreSQL Aurora Anda. Untuk informasi tentang cara mengatur Peningkatan versi minor otomatis,

dan cara kerja pengaturan saat diterapkan di tingkat klaster dan instans, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Periksa nilai opsi Enable auto minor version upgrade untuk semua cluster Aurora PostgreSQL DB Anda dengan menggunakan perintah sebagai berikut. describe-db-instances AWS CLI

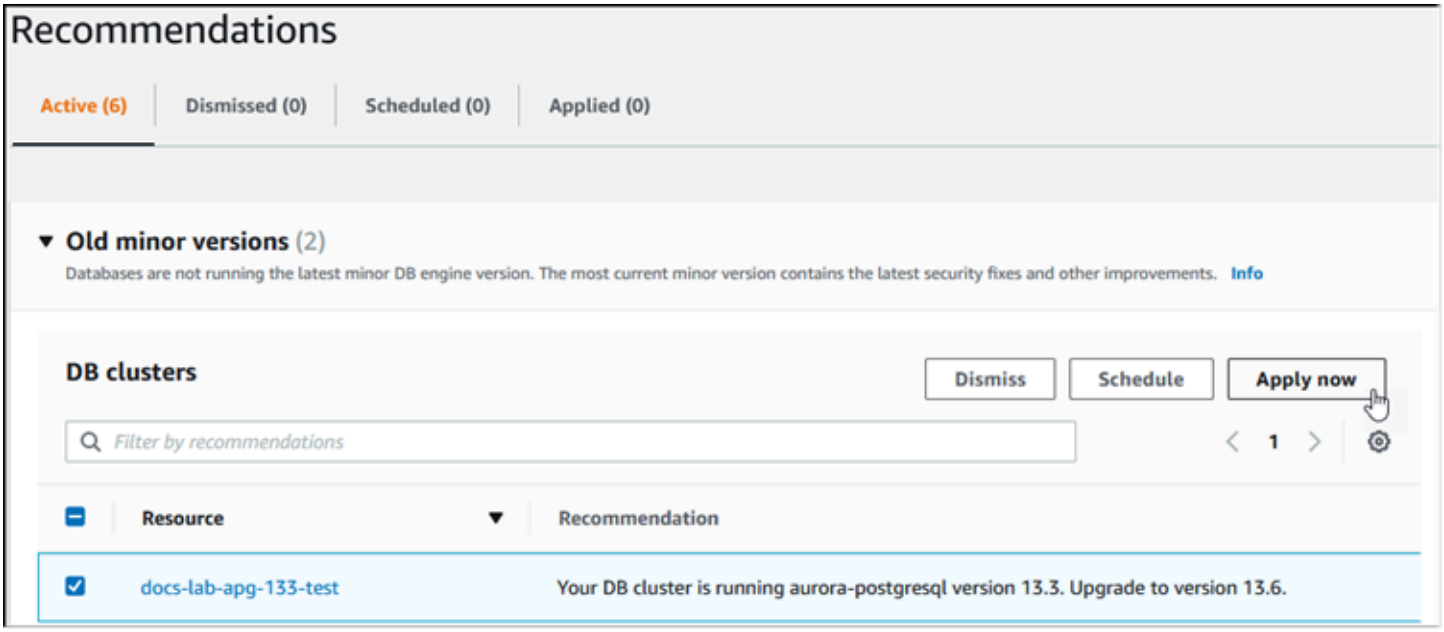
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Kueri ini menampilkan daftar semua klaster DB Aurora dan instans--nya dengan nilai `false` atau `true` untuk status pengaturan `AutoMinorVersionUpgrade`. Perintah mengasumsikan bahwa Anda telah AWS CLI dikonfigurasi untuk menargetkan default Wilayah AWS Anda.

Untuk informasi selengkapnya tentang opsi AmVU dan cara memodifikasi klaster DB Aurora untuk menggunakannya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Anda dapat meningkatkan klaster DB Aurora PostgreSQL ke versi minor baru baik dengan menanggapi tugas pemeliharaan, atau memodifikasi klaster untuk menggunakan versi baru.

Anda dapat mengidentifikasi peningkatan atau patch yang tersedia untuk klaster DB Aurora PostgreSQL dengan menggunakan konsol RDS dan membuka menu Rekomendasi. Di sana, Anda dapat menemukan daftar berbagai masalah pemeliharaan seperti Versi minor lama. Bergantung pada lingkungan produksi Anda, Anda dapat memilih untuk Menjadwalkan peningkatan atau mengambil tindakan segera, dengan memilih Terapkan sekarang, seperti yang ditunjukkan di bawah.



Untuk mempelajari lebih lanjut tentang cara memelihara klaster DB Aurora, termasuk cara menerapkan patch dan peningkatan versi minor secara manual, lihat Memelihara klaster DB Amazon Aurora.

Peningkatan rilis minor dan patching nol-waktu henti

Meningkatkan klaster DB Aurora PostgreSQL melibatkan kemungkinan pemadaman. Selama proses peningkatan, basis data dimatikan karena sedang ditingkatkan. Jika Anda memulai peningkatan saat basis data sibuk, Anda akan kehilangan semua koneksi dan transaksi yang sedang diproses oleh klaster DB. Jika Anda menunggu hingga basis data idle untuk melakukan peningkatan, Anda mungkin harus menunggu lama.

Fitur patching nol-waktu henti (ZDP) meningkatkan proses peningkatan. Dengan ZDP, baik peningkatan versi minor maupun patch dapat diterapkan dengan dampak minimal ke klaster DB Aurora PostgreSQL Anda. ZDP digunakan saat menerapkan patch atau peningkatan versi minor yang lebih baru ke versi Aurora PostgreSQL dan rilis lain yang lebih tinggi dengan versi minor ini dan versi mayor yang lebih baru. Artinya, tingkatkan ke versi minor baru dari salah satu rilis ini dan seterusnya akan menggunakan ZDP.

Tabel berikut menunjukkan versi Aurora PostgreSQL dan kelas instans DB tempat ZDP tersedia:

| Versi | Kelas instans db.r* | Kelas instans db.t* | Kelas instans db.x* | Kelas instans db.serverless |
|---|---|---|---|---|
| 10.21 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| 11.16 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| Versi 11.17 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| 12.11 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |

| Versi | Kelas instans db.r* | Kelas instans db.t* | Kelas instans db.x* | Kelas instans db.serverless |
|---|---|---|---|---|
| Versi 12.12 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| 13.7 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| Versi 13.8 dan versi yang lebih tinggi | Ya | Ya | Ya | Ya |
| 14.3 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| 14.4 dan versi yang lebih tinggi | Ya | Ya | Ya | N/A |
| Versi 14.5 dan versi yang lebih tinggi | Ya | Ya | Ya | Ya |
| Versi 15.3 dan versi yang lebih tinggi | Ya | Ya | Ya | Ya |
| 16.1 dan versi yang lebih tinggi | Ya | Ya | Ya | Ya |

Selama proses peningkatan menggunakan ZDP, mesin basis data mencari titik tenang untuk menjeda semua transaksi baru. Tindakan ini melindungi basis data selama patch dan peningkatan. Untuk memastikan bahwa aplikasi Anda berjalan lancar dengan transaksi yang dijeda, sebaiknya Anda mengintegrasikan logika coba lagi ke dalam kode Anda. Pendekatan ini memastikan bahwa sistem dapat mengelola waktu henti singkat tanpa gagal dan dapat mencoba kembali transaksi baru setelah peningkatan.

Saat ZDP berhasil diselesaikan, sesi aplikasi dipertahankan kecuali untuk sesi koneksinya hilang, dan mesin basis data dimulai ulang saat peningkatan masih berlangsung. Meskipun memulai ulang mesin basis data dapat menyebabkan penurunan throughput sementara, hal ini biasanya hanya berlangsung selama beberapa detik atau paling lama kira-kira satu menit.

Dalam beberapa kasus, patching nol-waktu henti (ZDP) mungkin tidak berhasil. Misalnya, perubahan parameter yang berada dalam status `pending` pada klaster DB Aurora PostgreSQL Anda atau instansnya mengganggu ZDP.

Anda dapat menemukan metrik dan peristiwa untuk operasi ZDP di halaman Peristiwa di konsol. Peristiwa ini termasuk dimulainya peningkatan ZDP dan penyelesaian peningkatan. Dalam hal ini Anda dapat menemukan durasi pemrosesannya, dan jumlah koneksi yang dipertahankan dan hilang yang terjadi selama mulai ulang. Anda dapat menemukan detailnya di log kesalahan basis data Anda.

Keterbatasan penambalan zero-downtime

Batasan berikut berlaku untuk patching zero-downtime:

- ZDP mencoba mempertahankan koneksi klien saat ini ke instance penulis Aurora PostgreSQL Anda di seluruh proses peningkatan PostgreSQL Aurora. Namun, dalam kasus berikut, koneksi akan dibatalkan agar ZDP selesai:
  - Kueri atau transaksi berjalan lama sedang berlangsung.
  - Pernyataan bahasa definisi data (DDL) sedang berlangsung.
  - Tabel sementara atau kunci tabel sedang digunakan.
  - Semua sesi mendengarkan di saluran pemberitahuan.
  - Kursor dalam status 'WITH HOLD' sedang digunakan.
  - TLSv1.1 koneksi sedang digunakan. Dimulai dengan versi Aurora PostgreSQL lebih lambat dari 16.1, 15.3, 14.8, 13.11, 12.15, dan 11.20, ZDP didukung dengan koneksi.3. TLSv1
- ZDP tidak didukung dalam kasus berikut:
  - Ketika cluster Aurora PostgreSQL DB dikonfigurasi sebagai Aurora Serverless v1.
  - Selama upgrade dari setiap instance pembaca Aurora.
  - Selama upgrade setiap instance pembaca Aurora yang merupakan bagian dari cluster Aurora Global Database di Wilayah sekunder.
  - Selama patch OS dan peningkatan OS.

### Meningkatkan mesin Aurora PostgreSQL ke versi minor baru

Anda dapat memutakhirkan cluster Aurora PostgreSQL DB Anda ke versi minor baru dengan menggunakan konsol, API, atau RDS. AWS CLI Sebelum melakukan peningkatan, sebaiknya Anda mengikuti praktik terbaik yang sama yang kami sarankan untuk peningkatan versi mayor. Seperti halnya versi mayor baru, versi minor baru juga dapat memiliki peningkatan pengoptimal, seperti perbaikan, yang dapat menyebabkan regresi rencana kueri. Untuk memastikan stabilitas rencana, sebaiknya Anda menggunakan ekstensi Manajemen Rencana Kueri (QPM) seperti yang dijelaskan dalam Memastikan stabilitas rencana setelah peningkatan versi mayor.

#### Konsol

Untuk meningkatkan versi mesin klaster DB Aurora PostgreSQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin Anda tingkatkan.
3. Pilih Ubah. Halaman Ubah klaster DB akan muncul.
4. Untuk Versi mesin, pilih versi baru.
5. Pilih Lanjutkan dan periksa ringkasan perubahan.
6. Untuk segera menerapkan perubahan, pilih Terapkan segera. Dalam beberapa kasus, pemadaman dapat terjadi jika opsi ini dipilih. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Ubah Klaster untuk menyimpan perubahan.

   Atau pilih Kembali untuk mengedit perubahan atau Batal untuk membatalkan perubahan.

#### AWS CLI

Untuk memutakhirkan versi mesin cluster DB, gunakan modify-db-cluster AWS CLI perintah dengan parameter berikut:

- `--db-cluster-identifier` – Nama klaster DB Aurora PostgreSQL Anda.
- `--engine-version` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan AWS CLI describe-db-engine-versionsperintah.

- `--no-apply-immediately` – Terapkan perubahan selama periode pemeliharaan berikutnya. Untuk menerapkan perubahan secara langsung, gunakan `--apply-immediately`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --no-apply-immediately
```

API RDS

Untuk meng-upgrade versi mesin dari cluster DB, gunakan DBCluster operasi Modify. Tentukan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB, misalnya *`mydbcluster`*.
- `EngineVersion` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan operasi Deskripsikan DBEngine Versi.
- `ApplyImmediately` – Apakah akan menerapkan perubahan secara langsung atau selama periode pemeliharaan berikutnya. Untuk segera menerapkan perubahan, tetapkan nilai ke `true`. Untuk menerapkan perubahan selama periode pemeliharaan berikutnya, tetapkan nilai ke `false`.

## Meningkatkan ekstensi PostgreSQL

Meningkatkan klaster DB Aurora PostgreSQL Anda ke versi mayor atau minor baru tidak akan meningkatkan ekstensi PostgreSQL secara bersamaan. Untuk sebagian besar ekstensi, Anda meningkatkan ekstensi setelah peningkatan versi mayor atau minor selesai. Namun, dalam beberapa kasus, Anda meningkatkan ekstensi sebelum meningkatkan mesin DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat list of extensions to update di Menguji peningkatan klaster DB produksi ke versi mayor baru.

Menginstal ekstensi PostgreSQL membutuhkan hak istimewa `rds_superuser`. Biasanya, `rds_superuser` mendelegasikan izin atas ekstensi tertentu untuk pengguna yang relevan (peran), untuk memfasilitasi pengelolaan ekstensi yang ditentukan. Itu berarti bahwa tugas meningkatkan semua ekstensi di klaster DB Aurora PostgreSQL Anda mungkin melibatkan banyak pengguna (peran) yang berbeda. Perhatikan hal ini terutama jika Anda ingin mengotomatiskan proses peningkatan dengan menggunakan skrip. Untuk informasi selengkapnya tentang hak istimewa dan peran PostgreSQL, lihat Keamanan dengan Amazon Aurora PostgreSQL.

> **Note**
>
> Untuk informasi selengkapnya tentang ekstensi PostGIS, lihat Mengelola data spasial dengan GIS ekstensi Post (Langkah 6: Tingkatkan GIS ekstensi Post).
> Untuk memperbarui ekstensi `pg_repack`, hapus ekstensi dan kemudian buat versi baru di instans DB yang ditingkatkan. Untuk informasi selengkapnya, lihat pg_repack installation di dokumentasi `pg_repack`.

Untuk memperbarui ekstensi setelah peningkatan mesin, gunakan perintah `ALTER EXTENSION UPDATE`.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

Untuk mencantumkan ekstensi yang saat ini terinstal, gunakan katalog pg_extension PostgreSQL dalam perintah berikut.

```
SELECT * FROM pg_extension;
```

Untuk melihat daftar versi ekstensi tertentu yang tersedia untuk penginstalan Anda, gunakan tampilan pg_available_extension_versions PostgreSQL dalam perintah berikut.

```
SELECT * FROM pg_available_extension_versions;
```

### Teknik peningkatan blue/green alternatif

Dalam situasi tertentu, prioritas utama Anda adalah melakukan switchover langsung dari klaster lama ke klaster yang ditingkatkan. Dalam situasi seperti itu, Anda dapat menggunakan proses multistep yang menjalankan cluster side-by-side lama dan baru. Di sini, Anda mereplikasi data dari klaster

lama ke klaster baru hingga Anda siap untuk mengambil alih klaster baru. Untuk detailnya, lihat Menggunakan Deployment Blue/Green untuk pembaruan basis data.

## Menggunakan rilis dukungan jangka panjang (LTS) Aurora PostgreSQL

Setiap Aurora PostgreSQL versi baru tetap tersedia sampai waktu tertentu untuk Anda gunakan saat membuat atau meningkatkan klaster DB. Setelah periode ini, Anda harus meningkatkan klaster yang menggunakan versi tersebut. Anda dapat meningkatkan klaster Anda secara manual sebelum periode dukungan berakhir, atau Aurora dapat secara otomatis meningkatkan versi untuk Anda ketika versi Aurora PostgreSQL tidak lagi didukung.

Aurora menetapkan versi Aurora PostgreSQL tertentu sebagai rilis dukungan jangka panjang (LTS). Klaster basis data yang menggunakan rilis LTS dapat bertahan pada versi yang sama lebih lama dan menjalani siklus peningkatan yang lebih sedikit dibandingkan klaster yang menggunakan rilis non-LTS. Versi minor LTS hanya mencakup perbaikan bug (melalui versi patch); versi LTS tidak menyertakan fitur baru yang dirilis setelah diperkenalkan.

Setahun sekali, klaster DB yang berjalan pada versi minor LTS di-patch ke versi patch terbaru dari rilis LTS. Kami melakukan patching ini untuk membantu memastikan bahwa Anda mendapatkan manfaat dari perbaikan keamanan dan stabilitas kumulatif. Kami mungkin memberi patch versi minor LTS lebih sering jika ada perbaikan penting, seperti untuk keamanan, yang perlu diterapkan.

> **Note**
>
> Jika Anda ingin tetap menggunakan versi minor LTS selama siklus hidupnya, pastikan untuk menonaktifkan upgrade versi minor otomatis untuk instans DB Anda. Untuk menghindari pemutakhiran cluster DB Anda secara otomatis dari versi minor LTS, kosongkan kotak centang Aktifkan pemutakhiran versi minor otomatis pada instans DB apa pun di cluster Aurora Anda.

Kami menyarankan Anda meningkatkan ke rilis terbaru, daripada menggunakan rilis LTS, untuk sebagian besar klaster Aurora PostgreSQL Anda. Melakukan hal itu memanfaatkan Aurora sebagai layanan terkelola dan memberi Anda akses ke fitur terbaru dan perbaikan bug. Rilis LTS ditujukan untuk klaster yang memiliki karakteristik berikut:

- Anda tidak dapat memberikan waktu henti pada aplikasi Aurora PostgreSQL untuk peningkatan di luar peristiwa langka untuk patch penting.

- Siklus pengujian untuk klaster dan aplikasi terkait membutuhkan waktu lama untuk setiap pembaruan ke mesin basis data Aurora PostgreSQL.
- Versi basis data untuk klaster Aurora PostgreSQL Anda memiliki semua fitur mesin DB dan perbaikan bug yang dibutuhkan aplikasi Anda.

Rilis LTS saat ini untuk Aurora PostgreSQL adalah sebagai berikut:

- PostgreSQL 15.10. Ini dirilis pada 27 Desember 2024. Untuk informasi lebih lanjut, lihat PostgreSQL 15.10 di Catatan Rilis untuk Aurora PostgreSQL.
- PostgreSQL 14.6. Versi ini dirilis pada 20 Januari 2023. Untuk informasi selengkapnya, lihat PostgreSQL 14.6 di Catatan Rilis untuk Aurora PostgreSQL.
- PostgreSQL 13.9. Versi ini dirilis pada 20 Januari 2023. Untuk informasi selengkapnya, lihat PostgreSQL 13.9 di Catatan Rilis untuk Aurora PostgreSQL.
- PostgreSQL 12.9. Versi ini dirilis pada 25 Februari 2022. Untuk informasi selengkapnya, lihat PostgreSQL 12.9 di Catatan Rilis untuk Aurora PostgreSQL.
- PostgreSQL 11.9 (Aurora PostgreSQL 3.4.) Versi ini dirilis pada 11 Desember 2020. Untuk informasi selengkapnya tentang versi ini, lihat PostgreSQL 11.9, Aurora PostgreSQL rilis 3.4 di Catatan Rilis untuk Aurora PostgreSQL.

Untuk detail tentang jadwal dukungan dan siklus rilis untuk versi LTS, lihat Kalender rilis untuk Aurora PostgreSQL.

Untuk informasi tentang cara mengidentifikasi versi mesin basis data dan Aurora, lihat Mengidentifikasi versi Amazon Aurora Postgre SQL.

# Menggunakan Basis Data Tanpa Batas Amazon Aurora Postgre SQL

Amazon Aurora Postgre SQL Limitless Database menyediakan penskalaan horizontal otomatis untuk memproses jutaan transaksi tulis per detik dan mengelola petabyte data sambil mempertahankan kesederhanaan pengoperasian di dalam satu database. Dengan Aurora Postgre SQL Limitless Database, Anda dapat fokus membangun aplikasi berskala tinggi tanpa harus membangun dan memelihara solusi kompleks untuk menskalakan data Anda di beberapa instans DB untuk mendukung beban kerja Anda.

Topik


- Arsitektur Basis Data Tanpa Batas Aurora PostgreSQL
- Memulai dengan Aurora SQL Postgre Limitless Database
- Persyaratan dan pertimbangan Basis Data Tanpa Batas Aurora PostgreSQL
- Prasyarat untuk menggunakan Aurora PostgreSQL Limitless Database
- Membuat cluster DB yang menggunakan Aurora SQL Postgre Limitless Database
- Bekerja dengan grup pecahan DB
- Membuat tabel Basis Data Tanpa Batas Aurora Postgre SQL
- Memuat data ke dalam Database Tanpa Batas Aurora Postgre SQL
- Meminta Basis Data Tanpa Batas Aurora Postgre SQL
- Mengelola Basis Data Tanpa Batas Aurora Postgre SQL
- Memantau Database Tanpa Batas Aurora PostgreSQL
- Mencadangkan dan memulihkan Database Tanpa Batas Aurora Postgre SQL
- Meningkatkan Database Tanpa Batas Amazon Aurora PostgreSQL
- Referensi Basis Data Tanpa Batas Aurora Postgre SQL

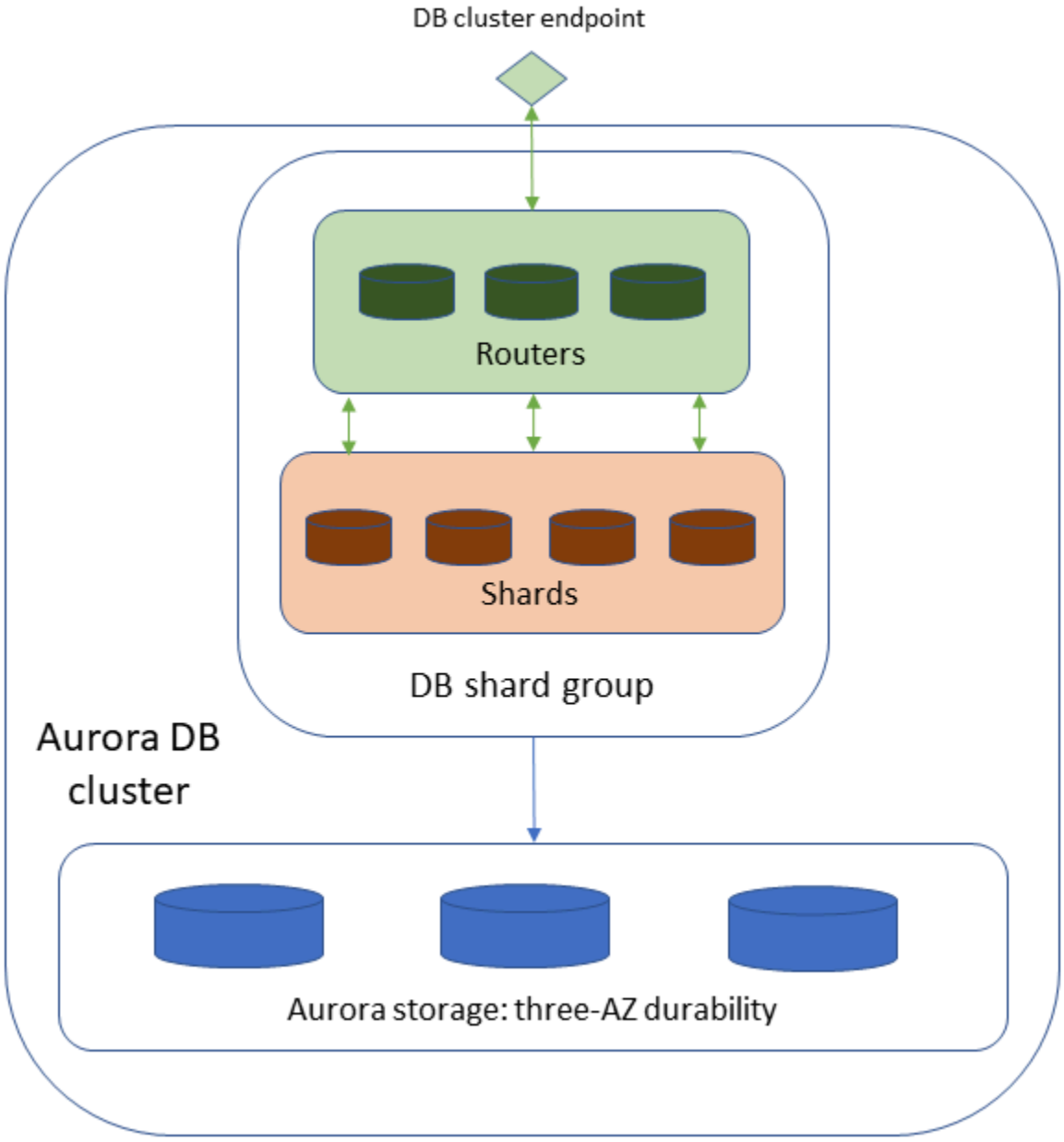
# Arsitektur Basis Data Tanpa Batas Aurora PostgreSQL

Limitless Database mencapai skala dengan arsitektur dua lapisan yang terdiri dari beberapa node database. Node adalah router atau pecahan.

- Pecahan adalah instance Aurora PostgreSQL DB yang masing-masing menyimpan subset data untuk database Anda, memungkinkan pemrosesan simultan untuk mencapai throughput penulisan yang lebih tinggi.
- Router mengelola sifat terdistribusi database dan menyajikan gambar database tunggal ke klien database. Router memelihara metadata tentang di mana data disimpan, mengurai perintah SQL yang masuk dan mengirim perintah tersebut ke pecahan. Kemudian mereka mengumpulkan data dari pecahan untuk mengembalikan satu hasil ke klien, dan mengelola transaksi terdistribusi untuk menjaga konsistensi di seluruh database terdistribusi.

Aurora PostgreSQL Limitless Database berbeda dari standar Klaster DB Aurora dengan memiliki grup shard DB alih-alih instance DB penulis dan instance DB pembaca. Semua node yang membentuk arsitektur Limitless Database Anda terkandung dalam grup shard DB. Pecahan individu dan router dalam grup pecahan DB tidak terlihat di Anda. Akun AWS Anda menggunakan endpoint cluster DB untuk mengakses Limitless Database.

Gambar berikut menunjukkan arsitektur tingkat tinggi Aurora PostgreSQL Limitless Database.




Untuk informasi selengkapnya tentang arsitektur Aurora PostgreSQL Limitless Database dan bagaimana Anda dapat menggunakannya, lihat Mencapai skala dengan Amazon Aurora PostgreSQL Limitless Database di saluran Acara. AWS YouTube

Untuk informasi lebih lanjut tentang arsitektur cluster Aurora DB standar, lihat. Klaster DB Amazon Aurora

# Istilah kunci untuk Aurora PostgreSQL Limitless Database

Kelompok pecahan DB

Sebuah wadah untuk node Limitless Database (pecahan dan router).

Router

Node yang menerima koneksi SQL dari klien, mengirimkan perintah SQL ke pecahan, mempertahankan konsistensi seluruh sistem, dan mengembalikan hasil ke klien.

Shard

Node yang menyimpan subset tabel sharded, salinan lengkap tabel referensi, dan tabel standar. Menerima kueri dari router, tetapi tidak dapat dihubungkan langsung oleh klien.

Meja sharded

Tabel dengan datanya dipartisi di seluruh pecahan.

Kunci pecahan

Kolom atau kumpulan kolom dalam tabel sharded yang digunakan untuk menentukan partisi di seluruh pecahan.

Tabel yang dikolokasikan

Dua tabel sharded yang berbagi kunci shard yang sama dan secara eksplisit dinyatakan sebagai collocated. Semua data untuk nilai kunci shard yang sama dikirim ke pecahan yang sama.

Tabel referensi

Sebuah tabel dengan datanya disalin secara penuh pada setiap pecahan.

Tabel standar

Jenis tabel default di Limitless Database. Anda dapat mengonversi tabel standar menjadi tabel sharded dan referensi.

Semua tabel standar disimpan pada pecahan yang sama yang dipilih oleh sistem, memungkinkan gabungan antara tabel standar yang akan dilakukan dalam satu pecahan. Namun, tabel standar dibatasi oleh kapasitas maksimum pecahan (128 TiB). Pecahan ini juga menyimpan data dari tabel sharded dan referensi, sehingga batas efektif untuk tabel standar lebih rendah dari 128 TiB.

## Jenis tabel untuk Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database mendukung tiga jenis tabel: sharded, reference, dan standard.

Tabel sharded memiliki data mereka didistribusikan di semua pecahan dalam grup pecahan DB. Limitless Database melakukan ini secara otomatis dengan menggunakan kunci shard, yang merupakan kolom atau kumpulan kolom yang Anda tentukan saat mempartisi tabel. Semua data dengan nilai yang sama untuk kunci shard dikirim ke pecahan yang sama. Sharding berbasis hash, bukan berbasis rentang atau daftar.

Berikut ini adalah kasus penggunaan yang baik untuk tabel sharded:

- Aplikasi ini bekerja dengan subset data yang berbeda.
- Mejanya sangat besar.
- Tabel berpotensi tumbuh lebih cepat dari tabel lainnya.

Tabel sharded dapat ditempatkan, artinya mereka berbagi kunci shard yang sama, dan bahwa semua data dari kedua tabel dengan nilai kunci shard yang sama dikirim ke shard yang sama. Jika Anda menyusun tabel dan menggabungkannya menggunakan kunci pecahan, penggabungan dapat dilakukan pada satu pecahan karena semua data yang diperlukan ada pada pecahan itu.

Tabel referensi memiliki salinan lengkap dari semua data mereka pada setiap pecahan dalam grup pecahan DB. Tabel referensi biasanya digunakan untuk tabel yang lebih kecil dengan volume tulis yang lebih rendah, tetapi itu masih perlu sering digabungkan dan tidak cocok untuk sharding. Contoh tabel referensi termasuk tabel tanggal, dan tabel data geografis seperti negara bagian, kota, dan kode pos.

Tabel standar adalah tipe tabel default di Aurora PostgreSQL Limitless Database. Mereka bukan tabel yang didistribusikan. Aurora PostgreSQL Limitless Database mendukung gabungan antara tabel standar dan tabel standar, sharded, dan referensi.

## Penagihan untuk Aurora PostgreSQL Limitless Database

Untuk informasi tentang cara Anda dikenakan biaya untuk Aurora PostgreSQL Limitless Database, lihat. DB untuk Aurora

Untuk mengetahui informasi tentang harga Aurora, lihat Halaman harga Aurora.

# Memulai dengan Aurora SQL Postgre Limitless Database

Anda melakukan tindakan berikut untuk memulai dengan Aurora SQL Postgre Limitless Database:

1. Buat cluster Aurora Postgre SQL DB dan grup pecahan DB untuk Limitless Database. Untuk informasi selengkapnya, lihat Membuat cluster DB yang menggunakan Aurora SQL Postgre Limitless Database.
2. Buat tabel sharded dan referensi dalam grup pecahan DB. Untuk informasi selengkapnya, lihat Membuat tabel Basis Data Tanpa Batas Aurora Postgre SQL.
3. Ubah kapasitas untuk grup pecahan DB Anda, pisahkan pecahan, dan tambahkan router. Untuk informasi selengkapnya, lihat Bekerja dengan grup pecahan DB.
4. Muat data ke dalam grup pecahan DB. Untuk informasi selengkapnya, lihat Memuat data ke dalam Database Tanpa Batas Aurora Postgre SQL.
5. Jalankan kueri dan SQL pernyataan lain pada grup shard DB. Untuk informasi selengkapnya, lihat Meminta Basis Data Tanpa Batas Aurora Postgre SQL.
6. Memantau kinerja Limitless Database. Untuk informasi selengkapnya, lihat Memantau Database Tanpa Batas Aurora PostgreSQL.

# Persyaratan dan pertimbangan Basis Data Tanpa Batas Aurora PostgreSQL

Aurora PostgreSQL Limitless Database memiliki persyaratan dan pertimbangan berikut.

Topik


- Persyaratan untuk Database Aurora PostgreSQL Limitless
- Pertimbangan untuk Database Aurora PostgreSQL Limitless
- Fitur tidak didukung di Aurora PostgreSQL Limitless Database


## Persyaratan untuk Database Aurora PostgreSQL Limitless

Pastikan untuk mengikuti persyaratan ini untuk Aurora PostgreSQL Limitless Database.

- Berikut ini Wilayah AWS tersedia:
  - Asia Pasifik (Hong Kong)
  - Asia Pasifik (Singapura)
  - Asia Pasifik (Sydney)
  - Asia Pasifik (Tokyo)
  - Eropa (Frankfurt)
  - Eropa (Irlandia)
  - Eropa (Stockholm)
  - AS Timur (Virginia Utara)

    > **Note**
    >
    > Jika Anda membuat cluster DB Limitless Database dalam hal ini Wilayah AWS, jangan sertakan `us-east-1e` Availability Zone (AZ) di grup subnet DB Anda. Karena keterbatasan sumber daya, Aurora Serverless v2 —dan karenanya Limitless Database— tidak didukung di AZ. `us-east-1e`

  - AS Timur (Ohio)
  - AS Barat (Oregon)

AWS CLI Prosedur dalam panduan ini mengasumsikan bahwa Anda menggunakan salah satu yang tersedia Wilayah AWS. Untuk informasi tentang cara menyetel default Wilayah AWS untuk AWS CLI, lihat Cara mengatur variabel lingkungan di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

- Aurora PostgreSQL Limitless Database hanya mendukung konfigurasi penyimpanan cluster DB. Aurora I/O-Optimized Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.
- Aurora PostgreSQL Limitless Database menggunakan Aurora PostgreSQL DB versi mesin khusus untuk Aurora PostgreSQL Limitless Database:
  - `16.4-limitless`
  - `16.6-limitless`
- Cluster DB Anda tidak dapat memiliki instans DB penulis atau pembaca.
- Anda harus menggunakan Enhanced Monitoring dan Performance Insights. Waktu retensi Performance Insights minimal harus 1 bulan (31 hari).
- Anda harus mengekspor log PostgreSQL ke Amazon Logs. CloudWatch

> **Note**
>
> Beberapa fitur yang diperlukan, seperti Enhanced Monitoring, Performance Insights, dan CloudWatch Log, dikenakan biaya tambahan. Untuk mengetahui informasi tentang harga Aurora, lihat Halaman harga Aurora.

## Pertimbangan untuk Database Aurora PostgreSQL Limitless

Pertimbangan berikut berlaku untuk grup pecahan DB di Aurora PostgreSQL Limitless Database:

- Anda hanya dapat memiliki satu grup pecahan DB per cluster DB.
- Anda dapat memiliki hingga lima grup pecahan DB per Wilayah AWS.

  Oleh karena itu, Anda dapat memiliki hingga lima cluster DB Database Aurora PostgreSQL Limitless per. Wilayah AWS Untuk informasi selengkapnya, lihat Kuota dalam Amazon Aurora.
- Anda dapat mengatur kapasitas maksimum grup pecahan DB ke 16-6144 ACUs. Untuk batas kapasitas lebih tinggi dari 6144 ACUs, hubungi. AWS

Jumlah awal router dan pecahan ditentukan oleh kapasitas maksimum yang Anda atur saat Anda membuat grup pecahan DB. Untuk informasi selengkapnya, lihat Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat.

- Jumlah router dan pecahan tidak berubah ketika Anda memodifikasi kapasitas maksimum grup shard DB.
- Pastikan subnet DB tempat Anda membuat grup shard DB memiliki alamat IP gratis yang cukup untuk terhubung dengan grup shard DB. Anda memerlukan satu alamat IP untuk setiap router dan hingga tiga alamat IP untuk setiap pecahan dalam grup pecahan DB.

  Untuk informasi selengkapnya tentang jumlah router yang dibuat saat Anda membuat grup pecahan DB, lihat. Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat
- Jika Anda membuat grup pecahan DB Anda dapat diakses publik, pastikan untuk menyiapkan gateway internet di VPC Anda.
- Anda menggunakan fungsi SQL untuk membagi pecahan dan menambahkan router.
- Penggabungan pecahan tidak didukung.
- Anda tidak dapat menghapus pecahan dan router individual.
- Anda tidak dapat memodifikasi (melakukan `UPDATE` operasi pada) kunci pecahan dengan cara apa pun, termasuk mengubah nilainya dalam baris tabel.

  Untuk mengubah kunci pecahan, hapus lalu buat ulang.
- Tingkat isolasi tanpa komitmen baca, baca yang dapat diulang, dan dibaca didukung. Anda tidak dapat mengatur tingkat isolasi ke serializable.
- Beberapa perintah SQL tidak didukung. Untuk informasi selengkapnya, lihat Referensi Basis Data Tanpa Batas Aurora Postgre SQL.
- Tidak semua ekstensi PostgreSQL didukung. Untuk informasi selengkapnya, lihat Ekstensi.
- Saat membuat grup shard, atau saat menambahkan node grup shard baru (pecahan atau router), node tersebut dibuat di salah satu Availability Zones (AZs) yang tersedia untuk cluster DB. Anda tidak dapat memilih AZ tertentu untuk node individual.
- Jika Anda menggunakan redundansi komputasi 2 (dua standby komputasi untuk grup pecahan DB), pastikan grup subnet DB Anda memiliki setidaknya tiga. AZs
- Aurora PostgreSQL Limitless Database mendukung hingga 54 karakter untuk nama tabel sharded.

Pertimbangan berikut berlaku untuk cluster DB Basis Data Aurora PostgreSQL Limitless:

- Kami menyarankan Anda menggunakan kebijakan AWS terkelola untuk membatasi izin untuk database dan aplikasi Anda kepada yang dibutuhkan pelanggan untuk kasus penggunaannya. Untuk informasi selengkapnya, lihat Praktik terbaik kebijakan.
- Saat Anda membuat cluster DB Basis Data Aurora PostgreSQL Limitless, Anda hanya mengatur parameter penskalaan untuk grup shard DB.
- Anda tidak dapat menghentikan atau memulai cluster DB yang berisi grup pecahan DB.
- Jika Anda perlu menghapus cluster DB Anda, Anda harus menghapus grup shard DB terlebih dahulu.
- Aurora PostgreSQL Limitless Database tidak dapat menjadi sumber replikasi.

# Fitur tidak didukung di Aurora PostgreSQL Limitless Database

Fitur Aurora PostgreSQL berikut tidak didukung di Aurora PostgreSQL Limitless Database:

- Otentikasi Active Directory (Kerberos)
- DevOpsGuru Amazon
- Amazon ElastiCache
- Deployment Blue/Green Amazon RDS
- Proksi Amazon RDS
- Auto Scaling Aurora (menambahkan instance pembaca ke cluster DB secara otomatis)
- Basis Data Global Aurora
- Machine Learning Aurora
- Rekomendasi Aurora
- Aurora Serverless v1
- Integrasi Aurora Nol-ETL
- AWS Backup
- AWS Lambda integrasi
- AWS Secrets Manager
- Babelfish for Aurora PostgreSQL
- Kloning cluster DB
- Titik akhir kustom
- Aliran aktivitas basis data
- Replika baca
- API Data RDS

# Prasyarat untuk menggunakan Aurora PostgreSQL Limitless Database

Untuk menggunakan Aurora PostgreSQL Limitless Database, Anda harus terlebih dahulu melakukan tugas-tugas berikut.

Topik


- Mengaktifkan operasi grup shard DB


## Mengaktifkan operasi grup shard DB

Sebelum Anda dapat membuat grup shard DB, Anda harus mengaktifkan operasi grup shard DB.

- Tambahkan bagian berikut ke kebijakan IAM dari peran IAM pengguna yang mengakses Aurora PostgreSQL Limitless Database:

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDBShardGroup",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBShardGroup",
                "rds:DescribeDBShardGroups",
                "rds:DeleteDBShardGroup",
                "rds:ModifyDBShardGroup",
                "rds:RebootDBShardGroup"
            ],
            "Resource": [
                "arn:aws:rds:*:*:shard-group:*",
                "arn:aws:rds:*:*:cluster:*"
            ]
        }
    ]
}
```

# Membuat cluster DB yang menggunakan Aurora SQL Postgre Limitless Database

Anda membuat cluster Aurora DB baru menggunakan versi Limitless Database dari Aurora PostgreSQL, dan menambahkan grup shard DB ke cluster. Saat menambahkan grup pecahan DB, Anda menentukan kapasitas komputasi maksimum untuk seluruh grup pecahan DB (jumlah kapasitas untuk semua router dan pecahan) di unit kapasitas Aurora (). ACUs Masing-masing ACU adalah kombinasi dari sekitar 2 gibibytes (GiB) memori, koresponden, dan jaringan. CPU Penskalaan meningkatkan atau mengurangi kapasitas untuk grup pecahan DB Anda, tergantung pada beban kerja aplikasi Anda, mirip dengan caranya Aurora Serverless v2 cara kerja

Topik

- Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat
- Membuat cluster DB Anda

## Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat

Jumlah awal router dan pecahan dalam grup pecahan DB ditentukan oleh kapasitas maksimum yang Anda atur saat Anda membuat grup pecahan DB. Semakin tinggi kapasitas maksimum, semakin besar jumlah router dan pecahan yang dibuat dalam grup pecahan DB.

Setiap node (shard atau router) memiliki nilai kapasitas saat ini, juga diukur dalamACUs.

- Limitless Database menskalakan node hingga kapasitas yang lebih tinggi ketika kapasitas saat ini terlalu rendah untuk menangani beban. Namun, node tidak akan meningkat ketika total kapasitas maksimum.
- Limitless Database menskalakan node ke kapasitas yang lebih rendah ketika kapasitas saat ini lebih tinggi dari yang dibutuhkan. Namun, node tidak akan menurunkan skala ketika total kapasitas minimum.

Tabel berikut menunjukkan korelasi antara kapasitas maksimum grup pecahan DB dalam unit kapasitas Aurora (ACUs) dan jumlah node (router dan pecahan) yang dibuat.

> **Note**
>
> Nilai-nilai ini dapat berubah.
> Jika Anda mengatur redundansi komputasi ke nilai bukan nol, jumlah pecahan akan digandakan atau tiga kali lipat. Ini akan dikenakan biaya tambahan.
> Node dalam standby komputasi ditingkatkan ke atas dan ke bawah ke kapasitas yang sama dengan penulis. Anda tidak mengatur rentang kapasitas secara terpisah untuk standby.

| Total node | Router | Serpihan | Kapasitas minimum default (ACUs) | Kisaran kapasitas maksimum (ACUs) |
|---|---|---|---|---|
| 4 | 2 | 2 | 16 | 16—400 |
| 5 | 2 | 3 | 20 | 401—500 |
| 6 | 2 | 4 | 24 | 501—600 |
| 7 | 3 | 4 | 28 | 601—700 |
| 8 | 3 | 5 | 32 | 701—800 |
| 9 | 3 | 6 | 36 | 801—900 |
| 10 | 4 | 6 | 40 | 901—1000 |
| 11 | 4 | 7 | 44 | 1001—1100 |
| 12 | 4 | 8 | 48 | 1101—1200 |
| 13 | 5 | 8 | 52 | 1201—1300 |
| 14 | 5 | 9 | 56 | 1301—1400 |
| 15 | 5 | 10 | 60 | 1401—1500 |
| 16 | 6 | 10 | 64 | 1501—1600 |

| Total node | Router | Serpihan | Kapasitas minimum default (ACUs) | Kisaran kapasitas maksimum (ACUs) |
|---|---|---|---|---|
| 17 | 6 | 11 | 68 | 1601—1700 |
| 18 | 6 | 12 | 72 | 1701—1800 |
| 19 | 7 | 12 | 76 | 1801—1900 |
| 20 | 7 | 13 | 80 | 1901—2000 |
| 21 | 7 | 14 | 84 | 2001—2100 |
| 22 | 8 | 14 | 88 | 2101—2200 |
| 23 | 8 | 15 | 92 | 2201—2300 |
| 24 | 8 | 16 | 96 | 2301—6144 |

Konfigurasi dinamis berbasis kapasitas maksimum untuk grup pecahan DB hanya tersedia selama pembuatan. Jumlah router dan pecahan tetap sama ketika kapasitas maksimum dimodifikasi. Untuk informasi selengkapnya, lihat Mengubah kapasitas grup pecahan DB.

Anda dapat menggunakan SQL perintah untuk menambahkan pecahan dan router ke grup pecahan DB. Untuk informasi selengkapnya, lihat berikut ini:

- Memisahkan pecahan dalam grup pecahan DB
- Menambahkan router ke grup pecahan DB

> **Note**
>
> Anda tidak dapat menghapus pecahan atau router.

# Membuat cluster DB Anda

Gunakan prosedur berikut untuk membuat cluster Aurora PostgreSQL DB yang menggunakan Aurora PostgreSQL Limitless Database.

Anda dapat menggunakan salah satu AWS Management Console atau AWS CLI untuk membuat cluster DB Anda yang menggunakan Aurora PostgreSQL Limitless Database. Anda membuat cluster DB primer dan grup shard DB.

Konsol

Bila Anda menggunakan AWS Management Console untuk membuat cluster DB primer, grup shard DB juga dibuat dalam prosedur yang sama.

Untuk membuat cluster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Buat basis data.

   Halaman Buat database ditampilkan.
3. Untuk Jenis mesin, pilih Aurora (Kompatibel dengan PostgreSQL).
4. Untuk Versi, pilih salah satu dari berikut ini:

   - Aurora PostgreSQL dengan Database Limitless (Kompatibel dengan PostgreSQL 16.4)
   - Aurora PostgreSQL dengan Database Limitless (Kompatibel dengan PostgreSQL 16.6)
5. Untuk Database Tanpa Batas Aurora PostgreSQL:

a. Masukkan pengidentifikasi grup pecahan DB.

> **Important**
>
> Setelah Anda membuat grup pecahan DB, Anda tidak dapat mengubah pengidentifikasi cluster DB atau pengidentifikasi grup shard DB.

b. Untuk rentang kapasitas grup shard DB:

   i. Masukkan kapasitas Minimum (ACUs). Gunakan nilai minimal 16 ACUs.

      Untuk lingkungan pengembangan, nilai defaultnya adalah 16 ACUs. Untuk lingkungan produksi, nilai defaultnya adalah 24 ACUs.

   ii. Masukkan kapasitas maksimum (ACUs). Gunakan nilai minimal 16 ACUs atau paling banyak 6144 ACUs.

      Untuk lingkungan pengembangan, nilai defaultnya adalah 64 ACUs. Untuk lingkungan produksi, nilai defaultnya adalah 384 ACUs.

   Untuk informasi selengkapnya, lihat Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat.

c. Untuk penyebaran grup shard DB, pilih apakah akan membuat standby untuk grup pecahan DB:

- Tidak ada redundansi komputasi - Membuat grup pecahan DB tanpa standby untuk setiap pecahan. Ini adalah nilai default.
- Hitung redundansi dengan satu target failover — Membuat grup pecahan DB dengan satu siaga komputasi di Availability Zone (AZ) yang berbeda.
- Hitung redundansi dengan dua target failover - Membuat grup pecahan DB dengan dua standby komputasi dalam dua yang berbeda. AZs

> **Note**
>
> Jika Anda menyetel redundansi komputasi ke nilai bukan nol, jumlah total pecahan instans DB akan digandakan atau tiga kali lipat. Instans DB tambahan ini adalah standby komputasi, ditingkatkan dan turun ke kapasitas yang sama dengan instance penulis. Anda tidak mengatur rentang kapasitas secara terpisah untuk standby. Oleh karena itu, penggunaan dan tagihan ACU juga berlipat ganda dan tiga kali lipat. Untuk mengetahui penggunaan ACU yang tepat yang timbul dari redundansi komputasi, lihat metrik di. `DBShardGroupComputeRedundancyCapacity` DBShardMetrik grup

d. Pilih apakah akan membuat grup pecahan DB dapat diakses publik.

> **Note**
>
> Anda tidak dapat mengubah pengaturan ini setelah Anda membuat grup shard DB.

6. Untuk Konektivitas:

a. (Opsional) Pilih Connect ke sumber daya EC2 komputasi, lalu pilih EC2 instance yang sudah ada atau buat yang baru.

> **Note**
>
> Jika Anda terhubung ke sebuah EC2 instans, Anda tidak dapat membuat grup shard DB dapat diakses publik.

b. Untuk tipe Jaringan, pilih salah satu IPv4atau mode Dual-stack.

c. Pilih grup subnet Virtual Private Cloud (VPC) dan DB, atau gunakan pengaturan default.

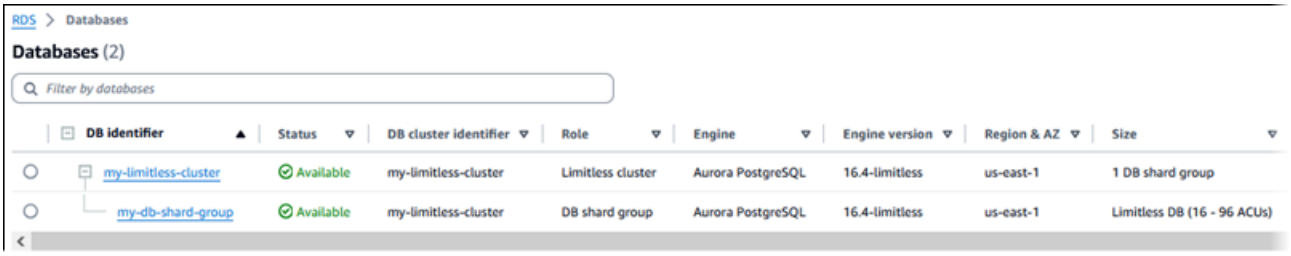
> **Note**
>
> Jika Anda membuat cluster DB Limitless Database di Wilayah AS Timur (Virginia N.), jangan sertakan `us-east-1e` Availability Zone (AZ) di grup subnet DB Anda. Karena keterbatasan sumber daya, Aurora Serverless v2 —dan karenanya Limitless Database—tidak didukung di AZ. `us-east-1e`

d. Pilih grup keamanan VPC (firewall), atau gunakan pengaturan default.

7. Untuk otentikasi Database, pilih otentikasi kata sandi atau otentikasi basis data Password dan IAM.
8. Untuk Monitoring, pastikan kotak centang Aktifkan Performance Insights dan Enable Enhanced Monitoring dipilih.

   Untuk Performance Insights, pilih waktu retensi minimal 1 bulan.
9. Perluas konfigurasi tambahan terakhir pada halaman.
10. Untuk ekspor Log, pastikan kotak centang log PostgreSQL dipilih.
11. Tentukan pengaturan lain sesuai kebutuhan. Untuk informasi selengkapnya, lihat Pengaturan untuk klaster Aurora DB.
12. Pilih Buat basis data.

Setelah cluster DB utama dan grup shard DB dibuat, mereka ditampilkan di halaman Databases.



CLI

Bila Anda menggunakan AWS CLI untuk membuat cluster DB yang menggunakan Aurora PostgreSQL Limitless Database, Anda melakukan tugas-tugas berikut:

1. Buat cluster DB utama.
2. Buat grup pecahan DB.

### Buat cluster DB utama

Parameter berikut diperlukan untuk membuat cluster DB:

- `--db-cluster-identifier` – Nama klaster DB Anda.
- `--engine`- Cluster DB harus menggunakan mesin `aurora-postgresql` DB.
- `--engine-version`- Cluster DB harus menggunakan salah satu versi mesin DB:
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`- Cluster DB harus menggunakan konfigurasi penyimpanan cluster `aurora-iopt1` DB.
- `--cluster-scalability-type`— Menentukan mode skalabilitas cluster Aurora DB. Ketika diatur `kelimitless`, cluster beroperasi sebagai Database Aurora PostgreSQL Limitless. Ketika diatur ke `standard` (default), cluster menggunakan pembuatan instans DB normal.

  > **Note**
  >
  > Anda tidak dapat mengubah pengaturan ini setelah Anda membuat cluster DB.

- `--master-username`— Nama pengguna master untuk cluster DB.
- `--master-user-password`— Kata sandi untuk pengguna utama.
- `--enable-performance-insights`— Anda harus mengaktifkan Performance Insights.
- `--performance-insights-retention-period`- Periode retensi Performance Insights minimal harus 31 hari.
- `--monitoring-interval`— Interval, dalam hitungan detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan untuk cluster DB. Nilai ini tidak bisa`0`.
- `--monitoring-role-arn`— Nama Sumber Daya Amazon (ARN) untuk peran IAM yang memungkinkan RDS mengirim metrik Pemantauan yang Ditingkatkan ke Log Amazon. CloudWatch

- `--enable-cloudwatch-logs-exports`— Anda harus mengekspor `postgresql` log ke CloudWatch Log.

Parameter berikut ini bersifat opsional:

- `--db-subnet-group-name`— Grup subnet DB untuk mengasosiasikan dengan cluster DB. Ini juga menentukan VPC yang terkait dengan cluster DB.

  > **Note**
  >
  > Jika Anda membuat cluster DB Limitless Database di Wilayah AS Timur (Virginia N.), jangan sertakan `us-east-1e` Availability Zone (AZ) di grup subnet DB Anda. Karena keterbatasan sumber daya, Aurora Serverless v2 —dan karenanya Limitless Database— tidak didukung di AZ. `us-east-1e`

- `--vpc-security-group-ids`— Daftar grup keamanan VPC untuk diasosiasikan dengan cluster DB.
- `--performance-insights-kms-key-id`— AWS KMS key Pengidentifikasi untuk enkripsi data Performance Insights. Jika Anda tidak menentukan kunci KMS, kunci default untuk Anda Akun AWS digunakan.
- `--region`— Wilayah AWS Tempat Anda membuat cluster DB. Itu harus menjadi salah satu yang mendukung Aurora PostgreSQL Limitless Database.

Untuk menggunakan grup keamanan VPC dan VPC default, hilangkan opsi dan. `--db-subnet-group-name` `--vpc-security-group-ids`

Untuk membuat cluster DB primer

- ```
  aws rds create-db-cluster \
      --db-cluster-identifier my-limitless-cluster \
      --engine aurora-postgresql \
      --engine-version 16.6-limitless \
      --storage-type aurora-iopt1 \
      --cluster-scalability-type limitless \
      --master-username myuser \
      --master-user-password mypassword \
      --db-subnet-group-name mysubnetgroup \
      --vpc-security-group-ids sg-c7e5b0d2 \
  ```
  ```

```
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
    --enable-cloudwatch-logs-exports postgresql
```

Untuk informasi selengkapnya, lihat create-db-cluster.

Buat grup pecahan DB

Selanjutnya Anda membuat grup pecahan DB di cluster DB yang baru saja Anda buat. Parameter-parameter berikut diperlukan:

- `--db-shard-group-identifier`— Nama grup pecahan DB Anda.

  Pengidentifikasi grup pecahan DB memiliki kendala berikut:

  - Itu harus unik di Akun AWS dan Wilayah AWS di mana Anda membuatnya.
  - Itu harus berisi 1—63 huruf, angka, atau tanda hubung.
  - Karakter pertama harus berupa huruf.
  - Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.
  - 

    **Important**

    Setelah Anda membuat grup pecahan DB, Anda tidak dapat mengubah pengidentifikasi cluster DB atau pengidentifikasi grup shard DB.
- `--db-cluster-identifier`— Nama cluster DB tempat Anda membuat grup shard DB.
- `--max-acu`- Kapasitas maksimum grup pecahan DB Anda. Itu harus dari 16-6144 ACUs. Untuk batas kapasitas lebih tinggi dari 6144 ACUs, hubungi. AWS

  Jumlah awal router dan pecahan ditentukan oleh kapasitas maksimum yang Anda atur saat Anda membuat grup pecahan DB. Semakin tinggi kapasitas maksimum, semakin besar jumlah router dan pecahan yang dibuat dalam grup pecahan DB. Untuk informasi selengkapnya, lihat Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat.

Parameter berikut ini bersifat opsional:

- `--compute-redundancy`— Apakah akan membuat standbys untuk grup pecahan DB. Parameter ini dapat memiliki nilai berikut:
  - `0`— Membuat grup pecahan DB tanpa standby untuk setiap pecahan. Ini adalah nilai default.
  - 1— Membuat grup pecahan DB dengan satu siaga komputasi di Availability Zone (AZ) yang berbeda.
  - 2— Membuat grup pecahan DB dengan dua standby komputasi dalam dua yang berbeda. AZs

> **Note**
>
> Jika Anda menyetel redundansi komputasi ke nilai bukan nol, jumlah pecahan akan digandakan atau tiga kali lipat. Ini akan dikenakan biaya tambahan.
> Node dalam standby komputasi ditingkatkan ke atas dan ke bawah ke kapasitas yang sama dengan penulis. Anda tidak mengatur rentang kapasitas secara terpisah untuk standby.

- `--min-acu`- Kapasitas minimum grup pecahan DB Anda. Setidaknya harus 16 ACUs, yang merupakan nilai default.
- `--publicly-accessible|--no-publicly-accessible`— Apakah akan menetapkan alamat IP yang dapat diakses publik ke grup pecahan DB. Akses ke grup shard DB dikendalikan oleh grup keamanan yang digunakan oleh cluster.

  Nilai default-nya `--no-publicly-accessible`.

> **Note**
>
> Anda tidak dapat mengubah pengaturan ini setelah Anda membuat grup shard DB.

Untuk membuat grup pecahan DB

- ```
  aws rds create-db-shard-group \
      --db-shard-group-identifier my-db-shard-group \
      --db-cluster-identifier my-limitless-cluster \
      --max-acu 1000
  ```
  ```

# Bekerja dengan grup pecahan DB

Anda melakukan tugas-tugas berikut untuk menambah dan mengelola grup shard DB untuk Aurora PostgreSQL Limitless Database.

Topik


- Menghubungkan ke cluster DB Database PostgreSQL Limitless Aurora Anda
- Menemukan jumlah router dan pecahan dalam grup pecahan DB
- Menggambarkan kelompok pecahan DB
- Mem-boot ulang grup pecahan DB
- Mengubah kapasitas grup pecahan DB
- Memisahkan pecahan dalam grup pecahan DB
- Menambahkan router ke grup pecahan DB
- Menghapus grup pecahan DB
- Menambahkan grup pecahan DB ke cluster DB Basis Data Aurora SQL Postgre Limitless yang ada

# Menghubungkan ke cluster DB Database PostgreSQL Limitless Aurora Anda

Untuk bekerja dengan Aurora PostgreSQL Limitless Database, Anda terhubung ke penulis cluster atau titik akhir pembaca. Anda dapat menggunakan `psql` atau utilitas koneksi lain yang bekerja dengan PostgreSQL:

```
$ psql -h DB_cluster_endpoint -p port_number -U database_username -d postgres_limitless
```

Contoh berikut menggunakan endpoint untuk cluster DB yang Anda buat diCLI.

```
$ psql -h my-limitless-cluster.cluster-ckifpdyyyxxx.us-east-1.rds.amazonaws.com -p 5432
 -U postgres -d postgres_limitless
```

> **Note**
>
> Database default untuk grup shard DB di Aurora PostgreSQL Limitless Database adalah. `postgres_limitless`

## Menggunakan Plugin Koneksi Tanpa Batas

Saat menghubungkan ke Aurora PostgreSQL Limitless Database, klien terhubung menggunakan titik akhir cluster, dan dialihkan ke router transaksi oleh Amazon Route 53. Namun, Route 53 terbatas dalam kemampuannya untuk memuat keseimbangan, dan dapat memungkinkan beban kerja yang tidak merata pada router transaksi. Plugin Koneksi Tanpa Batas untuk Driver AWS JDBC mengatasi hal ini dengan melakukan penyeimbangan beban sisi klien dengan kesadaran beban. Untuk informasi lebih lanjut tentang Driver AWS JDBC, lihat. Menghubungkan ke Aurora PostgreSQL dengan Driver Amazon Web Services () JDBC AWS

## Menemukan jumlah router dan pecahan dalam grup pecahan DB

Anda dapat menggunakan kueri berikut untuk menemukan jumlah router dan pecahan:

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
 6             | shard
```

## Menggambarkan kelompok pecahan DB

Gunakan `describe-db-shard-groups` AWS CLI perintah untuk menggambarkan grup shard DB Anda. Parameter berikut bersifat opsional:

- `--db-shard-group-identifier`— Nama grup pecahan DB.

Contoh berikut menjelaskan kelompok pecahan DB tertentu.

```
aws rds describe-db-shard-groups --db-shard-group-identifier my-db-shard-group
```

Outputnya menyerupai contoh berikut.

```
{
    "DBShardGroups": [
        {
            "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
            "DBShardGroupIdentifier": "my-shard-group",
            "DBClusterIdentifier": "my-limitless-cluster",
            "MaxACU": 1000.0,
            "ComputeRedundancy": 0,
            "Status": "available",
            "PubliclyAccessible": false,
            "Endpoint": "my-limitless-cluster.limitless-ccetp2abcdef.us-
east-1.rds.amazonaws.com"
        }
```

```
    ]
}
```

## Mem-boot ulang grup pecahan DB

Terkadang Anda harus me-reboot grup shard DB Anda, misalnya ketika `max_connections` parameter berubah karena perubahan kapasitas maksimum.

Anda dapat menggunakan AWS Management Console atau AWS CLI untuk mengubah kapasitas grup shard DB.

Konsol

Gunakan prosedur berikut:

Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

1. Arahkan ke halaman Basis data.
2. Pilih grup pecahan DB yang ingin Anda reboot.
3. Untuk Tindakan, pilih Boot ulang.
4. Pilih Konfirmasi.

CLI

Untuk me-reboot grup shard DB, gunakan `reboot-db-shard-group` AWS CLI perintah dengan parameter berikut:

- `--db-shard-group-identifier`— Nama grup pecahan DB.

Contoh berikut reboot grup shard DB.

```
aws rds reboot-db-shard-group --db-shard-group-identifier my-db-shard-group
```

# Mengubah kapasitas grup pecahan DB

Anda dapat menggunakan AWS Management Console atau AWS CLI untuk mengubah kapasitas grup shard DB.

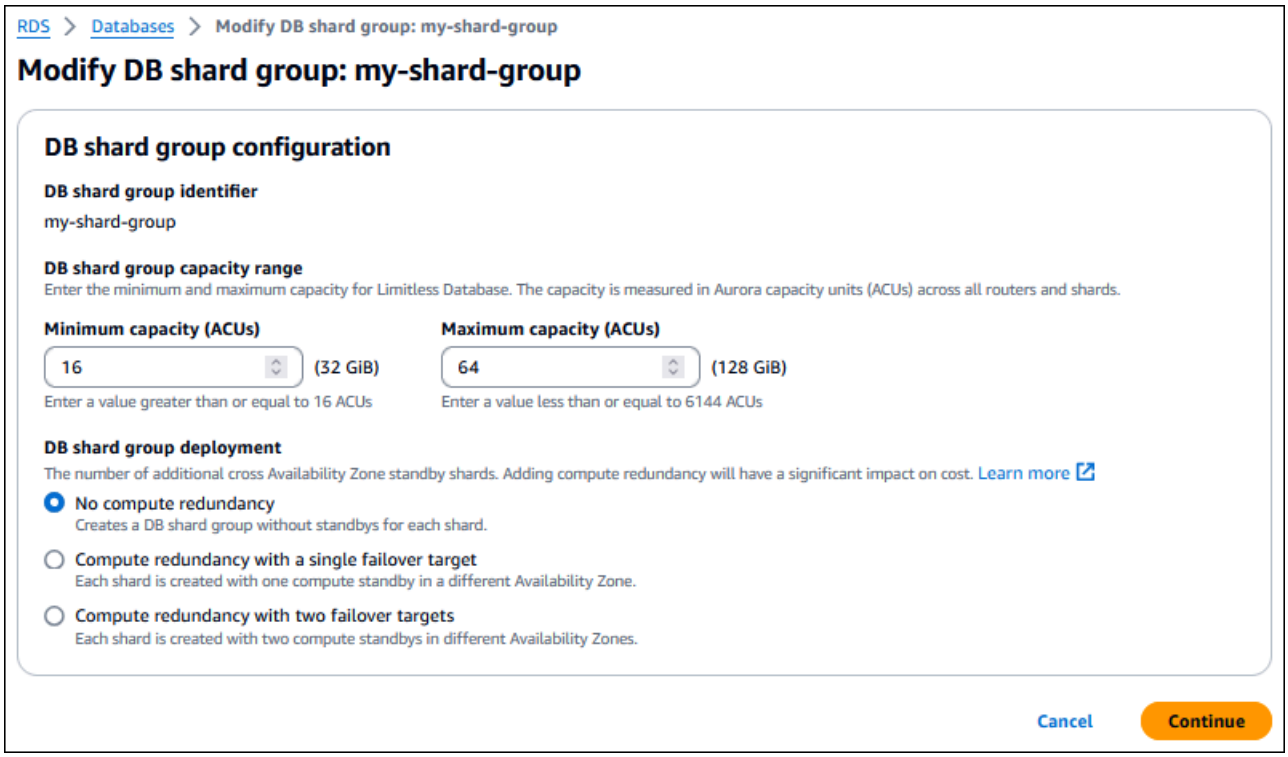
Konsol

Gunakan prosedur berikut:

Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

1. Arahkan ke halaman Basis data.
2. Pilih grup pecahan DB yang ingin Anda modifikasi.
3. Untuk Tindakan, pilih Ubah.

   Halaman grup Modify DB shard ditampilkan.




4. Masukkan nilai Kapasitas minimum (ACUs) baru, misalnya**100**.
5. Masukkan nilai Kapasitas maksimum (ACUs) baru, misalnya**1000**.
6. Pilih Lanjutkan.

   Halaman konfirmasi ditampilkan, dengan ringkasan perubahan Anda.

7. Tinjau perubahan Anda, lalu pilih Modify DB shard group.

CLI

Gunakan perintah `modify-db-shard-group` AWS CLI dengan parameter berikut:

- `--db-shard-group-identifier`— Nama grup pecahan DB.
- `--max-acu`- Kapasitas maksimum baru dari grup pecahan DB. Anda dapat mengatur kapasitas maksimum grup pecahan DB ke 16-6144ACUs. Untuk batas kapasitas lebih tinggi dari 6144ACUs, hubungi. AWS

  Jumlah router dan pecahan tidak berubah.
- `--min-acu`- Kapasitas minimum baru dari grup pecahan DB Anda. Setidaknya harus 16ACUs, yang merupakan nilai default.

CLIContoh berikut mengubah rentang kapasitas grup pecahan DB menjadi 100-1000ACUs.

```
aws rds modify-db-shard-group \
    --db-shard-group-identifier my-db-shard-group \
    --min-acu 100 \
    --max-acu 1000
```

# Memisahkan pecahan dalam grup pecahan DB

Anda dapat membagi pecahan dalam grup pecahan DB secara manual menjadi dua pecahan yang lebih kecil. Ini disebut pemecahan pecahan yang diprakarsai pengguna.

Aurora PostgreSQL Limitless Database juga dapat membagi pecahan ketika mereka memiliki jumlah data yang sangat besar atau penggunaan yang sangat tinggi. Ini disebut pemisahan pecahan yang diprakarsai sistem.

Topik


- Prasyarat
- Memisahkan pecahan
- Melacak pecahan pecahan
- Menyelesaikan pecahan pecahan
- Membatalkan pecahan pecahan


## Prasyarat

Pembagian pecahan yang diprakarsai pengguna memiliki prasyarat berikut:

- Anda harus memiliki grup pecahan DB.
- Grup pecahan DB tidak dapat kosong: harus berisi setidaknya satu tabel sharded.
- Seorang pengguna harus memiliki hak `rds_aurora_limitless_cluster_admin` istimewa. `rds_superuser`Memiliki hak istimewa ini; oleh karena itu pengguna master juga memilikinya. `rds_superuser`Dapat memberikan hak istimewa kepada pengguna lain:

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

- Anda harus mengetahui ID subcluster (node) dari pecahan yang ingin Anda pisahkan. Anda dapat memperoleh ID dengan menggunakan kueri berikut:

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
```

```
1               | router
2               | router
3               | shard
4               | shard
5               | shard
6               | shard
```

Untuk mengaktifkan pemisahan pecahan yang dimulai sistem, setel parameter cluster DB berikut dalam grup parameter cluster DB kustom yang terkait dengan cluster DB Anda:

| Parameter | Nilai |
| --- | --- |
| `rds_aurora.limitless_enable_auto_scale` | on |
| `rds_aurora.limitless_auto_scale_options` | Entah `split_shard` atau `add_router,split_shard` |
| `rds_aurora.limitless_finalize_split_shard_mode` | Parameter ini menentukan bagaimana pemisahan pecahan yang diprakarsai sistem diselesaikan. Nilai dapat menjadi salah satu dari yang berikut:<br>• `user_initiated` — Anda memutuskan kapan harus menyelesaikan pecahan pecahan. Ini adalah nilai default.<br>• `immediate` — Shard split segera diselesaikan.<br><br>Untuk informasi selengkapnya, lihat Menyelesaikan pecahan pecahan.<br><br>**Note**<br>Parameter ini hanya berlaku untuk pemisahan pecahan yang diprakarsai sistem. |

Untuk informasi selengkapnya, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

## Memisahkan pecahan

Untuk membagi pecahan dalam grup pecahan DB, gunakan fungsinya. `rds_aurora.limitless_split_shard` Fungsi ini memulai pekerjaan shard-split yang berjalan secara asinkron.

```
SELECT rds_aurora.limitless_split_shard('subcluster_id');
```

Tunggu kembalinya ID pekerjaan setelah penyerahan pekerjaan berhasil, misalnya:

```
SELECT rds_aurora.limitless_split_shard('3');

    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> Operasi pecahan pecahan bersamaan tidak didukung. Jalankan setiap operasi secara berurutan dan selesaikan setiap operasi sebelum memulai operasi penambahan lainnya.

## Melacak pecahan pecahan

Anda dapat menggunakan ID pekerjaan untuk melacak pekerjaan shard-split. Untuk mendeskripsikan pekerjaan tertentu dan mendapatkan detail lebih lanjut tentangnya, jalankan kueri berikut:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(job_id);
```

Misalnya:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(1691300000000);

    job_id     |    action    |      job_details      | status  |    submission_time
 |                 message
```

```
--------------+------------+----------------------+---------
+------------------------+-------------------------------------------
 1691300000000 | SPLIT_SHARD | Split Shard 3 by User | SUCCESS | 2023-08-06 05:33:20+00
 | Scaling job succeeded.                +
               |            |                       |         |
 | New shard instance with ID 7 was created.
(1 row)
```

Kueri mengembalikan kesalahan saat Anda melewati pekerjaan yang tidak ada sebagai masukan.

```
SELECT * from rds_aurora.limitless_list_shard_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

Anda dapat melacak status semua pekerjaan shard-split dengan menggunakan kueri yang sama tanpa ID pekerjaan, misalnya:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs();

    job_id     |   action    |  job_details          |   status    |    submission_time
     |                 message
--------------+------------+----------------------+-------------
+------------------------
+-------------------------------------------------------------
 1691200000000 | SPLIT_SHARD | Split Shard 3 by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | SPLIT_SHARD | Split Shard 4 by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded. +
               |            |                       |             |
     | New shard instance with ID 7 was created.
 1691400000000 | SPLIT_SHARD | Split Shard 5 by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add shard job 1691400000000.
               |            |                       |             |
     | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | SPLIT_SHARD | Split Shard 5 by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

Status pekerjaan dapat berupa salah satu dari yang berikut:

- `IN_PROGRESS`— Pekerjaan shard-split telah diserahkan dan sedang berlangsung. Anda hanya dapat memiliki satu pekerjaan yang sedang berlangsung pada satu waktu.

- `PENDING`— Pekerjaan shard-split menunggu Anda untuk menyelesaikannya. Untuk informasi selengkapnya, lihat Menyelesaikan pecahan pecahan.
- `CANCELLATION_IN_PROGRESS`— Pekerjaan shard-split dibatalkan oleh pengguna.
- `CANCELED`— Pekerjaan shard-split telah berhasil dibatalkan oleh pengguna atau oleh sistem.
- `SUCCESS`— Pekerjaan shard-split berhasil diselesaikan. `message`Bidang berisi ID instance dari pecahan baru.
- `FAILED`— Pekerjaan shard-split gagal. `message`Bidang berisi rincian kegagalan dan tindakan apa pun yang dapat diambil sebagai tindak lanjut dari pekerjaan yang gagal.

## Menyelesaikan pecahan pecahan

Finalisasi adalah langkah terakhir dari proses shard-split. Ini menyebabkan beberapa downtime. Jika Anda memulai pekerjaan shard-split, maka finalisasi terjadi segera setelah pekerjaan selesai dengan sukses.

Terkadang sistem membagi pecahan berdasarkan beban kerja, saat Anda mengaktifkan pemisahan pecahan yang dimulai sistem dengan menggunakan parameter. `rds_aurora.limitless_enable_auto_scale`

Dalam hal ini, Anda dapat memilih apakah finalisasi terjadi segera, atau pada waktu yang Anda pilih. Anda menggunakan parameter cluster `rds_aurora.limitless_finalize_split_shard_mode` DB untuk memilih kapan itu terjadi:

- Jika Anda menetapkan nilainya`immediate`, itu terjadi segera.
- Jika Anda menetapkan nilainya`user_initiated`, Anda harus menyelesaikan pekerjaan shard-split secara manual.

  Acara RDS dikirimkan kepada Anda, dan status pekerjaan shard-split diatur ke. `PENDING`

Saat disetel ke`user_initiated`, Anda menggunakan `rds_aurora.limitless_finalize_split_shard` fungsi untuk menyelesaikan pekerjaan shard-split:

```
SELECT * FROM rds_aurora.limitless_finalize_split_shard(job_id);
```

> **Note**
>
> Fungsi ini hanya berlaku untuk pecahan pecahan yang diprakarsai oleh sistem, bukan oleh Anda.

## Membatalkan pecahan pecahan

Anda dapat membatalkan pemisahan pecahan yang diprakarsai pengguna atau yang diprakarsai sistem yaitu atau. `IN_PROGRESS` `PENDING` Anda memerlukan ID pekerjaan untuk membatalkannya.

```
SELECT * from rds_aurora.limitless_cancel_shard_scale_jobs(job_id);
```

Tidak ada output yang dikembalikan kecuali ada kesalahan. Anda dapat melacak pembatalan menggunakan kueri pelacakan pekerjaan.

# Menambahkan router ke grup pecahan DB

Anda dapat menambahkan router ke grup pecahan DB.

Topik


- Prasyarat
- Menambahkan router
- Melacak penambahan router
- Membatalkan penambahan router


## Prasyarat

Menambahkan router memiliki prasyarat berikut:

- Anda harus memiliki grup pecahan DB.
- Seorang pengguna harus memiliki hak `rds_aurora_limitless_cluster_admin` istimewa. `rds_superuser`Memiliki hak istimewa ini; oleh karena itu pengguna master juga memilikinya. `rds_superuser`Dapat memberikan hak istimewa kepada pengguna lain:

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

> **Note**
>
> Jika Anda Akun AWS mengubah sertifikat CA default Anda setelah grup pecahan DB dibuat, router baru akan menggunakan sertifikat CA baru, yang berbeda dari sertifikat CA router yang ada. Tergantung pada toko kepercayaan Anda, beberapa koneksi mungkin gagal.

- Untuk mengaktifkan penambahan router yang dimulai sistem, atur parameter cluster DB berikut dalam grup parameter cluster DB khusus yang terkait dengan cluster DB Anda:

| Parameter | Nilai |
| --- | --- |
| `rds_aurora.limitless_enable_auto_scale` | `on` |
| `rds_aurora.limitless_auto_scale_options` | Entah `add_router` atau `add_router,split_shard` |

Untuk informasi selengkapnya, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

## Menambahkan router

Untuk menambahkan router, gunakan `rds_aurora.limitless_add_router` fungsinya. Fungsi ini memulai pekerjaan penambahan router yang berjalan secara asinkron.

```
SELECT rds_aurora.limitless_add_router();
```

Tunggu kembalinya ID pekerjaan setelah penyerahan pekerjaan berhasil, misalnya:

```
    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> Operasi penambahan router bersamaan tidak didukung. Jalankan operasi secara berurutan dan selesaikan setiap operasi sebelum memulai operasi penambahan lainnya.

## Melacak penambahan router

Anda dapat menggunakan ID pekerjaan untuk melacak pekerjaan penambahan router. Untuk mendeskripsikan pekerjaan tertentu dan mendapatkan detail lebih lanjut tentangnya, jalankan kueri berikut:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(job_id);
```

Misalnya:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(1691300000000);

    job_id     |   action   |        job_details       | status  |    submission_time
   |                   message
---------------+------------+--------------------------+---------
+------------------------+-------------------------------------------
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                   +
               |            |                          |         |
   | New router instance with ID 7 was created.
(1 row)
```

Kueri mengembalikan kesalahan saat Anda melewati pekerjaan yang tidak ada sebagai masukan.

```
SELECT * from rds_aurora.limitless_list_router_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

Anda dapat melacak status semua pekerjaan penambahan router dengan menggunakan kueri yang sama tanpa ID pekerjaan, misalnya:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs();

    job_id     |   action   |        job_details       |   status    |
 submission_time     |                  message
---------------+------------+--------------------------+-------------
+------------------------+-------------------------------------------
 1691200000000 | ADD_ROUTER | Add 1 new Router by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                 +
               |            |                          |             |
       | New router instance with ID 7 was created.
 1691400000000 | ADD_ROUTER | Add 1 new Router by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add router job 1691400000000.
               |            |                          |             |
       | Retry the command. If the issue persists, contact AWS Support.
```

```
 1691500000000 | ADD_ROUTER | Add 1 new Router by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

Status pekerjaan dapat berupa salah satu dari yang berikut:

- `IN_PROGRESS`— Pekerjaan penambahan router telah dikirimkan dan sedang berlangsung. Anda hanya dapat memiliki satu pekerjaan dalam satu waktu.
- `CANCELLATION_IN_PROGRESS`— Pekerjaan penambahan router dibatalkan oleh pengguna.
- `CANCELED`— Pekerjaan penambahan router telah berhasil dibatalkan oleh pengguna atau oleh sistem.
- `SUCCESS`— Pekerjaan penambahan router berhasil diselesaikan. `message`Bidang berisi ID instance dari router baru.
- `FAILED`— Pekerjaan penambahan router gagal. `message`Bidang berisi rincian kegagalan dan tindakan apa pun yang dapat diambil sebagai tindak lanjut dari pekerjaan yang gagal.

> **Note**
>
> Tidak ada `PENDING` status karena penambahan router tidak perlu diselesaikan. Mereka tidak mengalami downtime.

## Membatalkan penambahan router

Anda dapat membatalkan penambahan router itu`IN_PROGRESS`. Anda memerlukan ID pekerjaan untuk membatalkannya.

```
SELECT * from rds_aurora.limitless_cancel_router_scale_jobs(job_id);
```

Tidak ada output yang dikembalikan kecuali ada kesalahan. Anda dapat melacak pembatalan menggunakan kueri pelacakan pekerjaan.

# Menghapus grup pecahan DB

Anda dapat menghapus grup pecahan DB jika perlu. Menghapus grup shard DB menghapus node komputasi (pecahan dan router), tetapi bukan penyimpanan.

Konsol

Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

1. Arahkan ke halaman Basis data.
2. Pilih grup pecahan DB yang ingin Anda hapus.
3. Untuk Tindakan, pilih Hapus.
4. Masukkan **`delete me`** di dalam kotak, lalu pilih Hapus.

Grup pecahan DB dihapus.

AWS CLI

Untuk menghapus grup pecahan DB Anda, gunakan `delete-db-shard-group` AWS CLI perintah dengan parameter berikut:

- `--db-shard-group-identifier`— Nama grup pecahan DB.

Contoh berikut menghapus grup shard DB di cluster Aurora Postgre SQL DB yang Anda buat sebelumnya.

```
aws rds delete-db-shard-group --db-shard-group-identifier my-db-shard-group
```

# Menambahkan grup pecahan DB ke cluster DB Basis Data Aurora SQL Postgre Limitless yang ada

Anda dapat membuat grup pecahan DB di cluster DB yang ada, misalnya jika Anda memulihkan cluster DB atau Anda telah menghapus grup pecahan DB.

Untuk informasi lebih lanjut tentang cluster DB primer dan persyaratan grup shard DB, lihatPersyaratan dan pertimbangan Basis Data Tanpa Batas Aurora PostgreSQL.

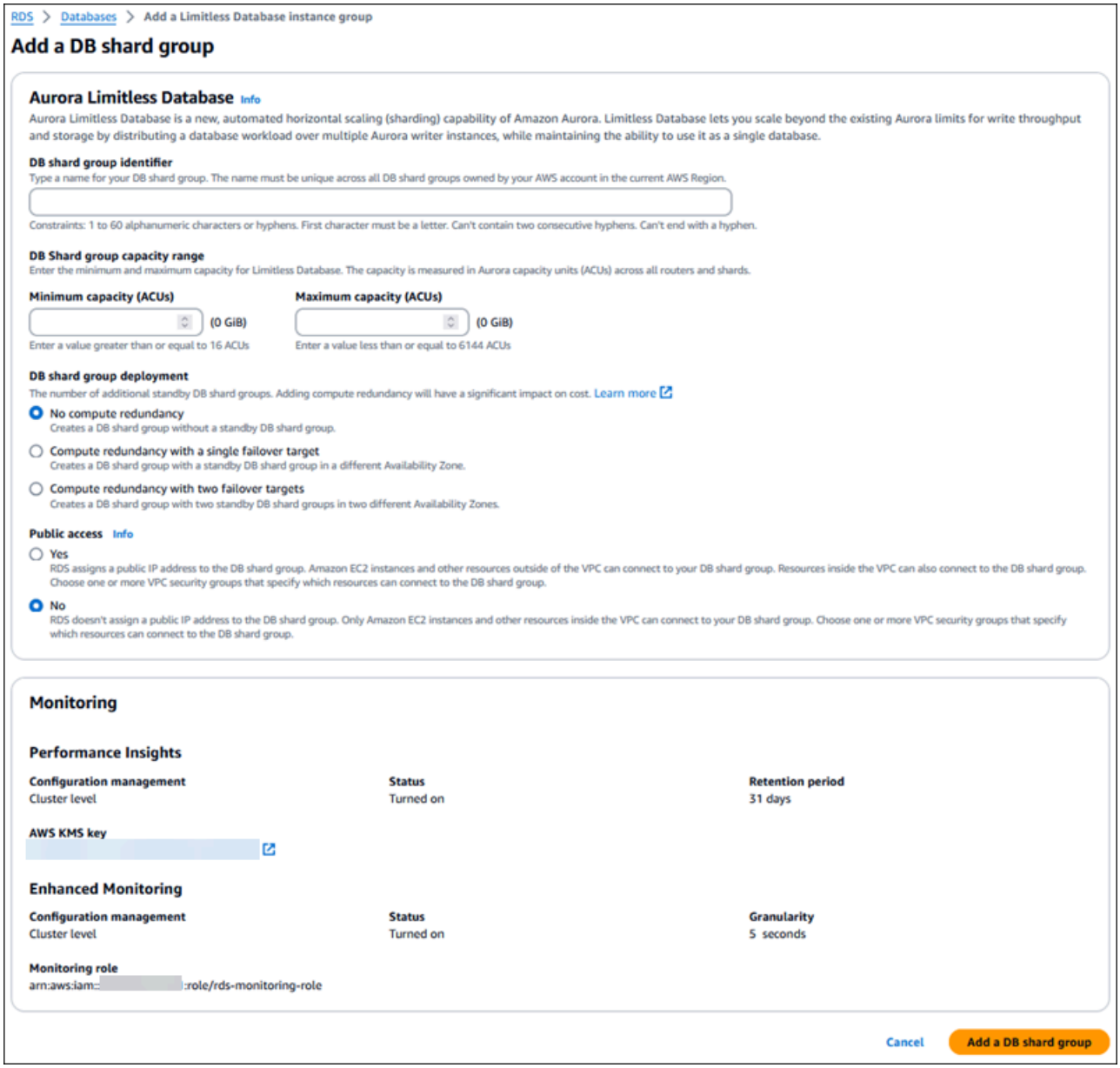
> **Note**
>
> Anda hanya dapat memiliki satu grup pecahan DB per cluster.
> Cluster DB Limitless Database harus dalam `available` keadaan sebelum Anda dapat membuat grup shard DB.

Konsol

Anda dapat menggunakan AWS Management Console untuk menambahkan grup shard DB ke cluster DB yang ada.

Untuk menambahkan grup pecahan DB

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Arahkan ke halaman Basis data.
3. Pilih cluster DB Limitless Database yang ingin Anda tambahkan grup shard DB.
4. Untuk Tindakan, pilih Tambahkan grup shard DB.




5. Masukkan pengidentifikasi grup pecahan DB.

> **Important**
>
> Setelah Anda membuat grup pecahan DB, Anda tidak dapat mengubah pengidentifikasi cluster DB atau pengidentifikasi grup shard DB.

6. Masukkan kapasitas Minimum (ACUs). Gunakan nilai minimal 16ACUs.
7. Masukkan kapasitas maksimum (ACUs). Gunakan nilai dari 16-6144ACUs.

Untuk informasi selengkapnya, lihat Mengkorelasikan kapasitas maksimum grup pecahan DB dengan jumlah router dan pecahan yang dibuat.

8. Untuk penyebaran grup shard DB, pilih apakah akan membuat standby untuk grup pecahan DB:
   - Tidak ada redundansi komputasi - Membuat grup pecahan DB tanpa standby untuk setiap pecahan. Ini adalah nilai default.
   - Hitung redundansi dengan satu target failover — Membuat grup pecahan DB dengan satu siaga komputasi di Availability Zone (AZ) yang berbeda.
   - Hitung redundansi dengan dua target failover - Membuat grup pecahan DB dengan dua standby komputasi dalam dua yang berbeda. AZs
9. Pilih apakah akan membuat grup pecahan DB dapat diakses publik.

   > **Note**
   >
   > Anda tidak dapat mengubah pengaturan ini setelah Anda membuat grup shard DB.

10. Pilih Tambahkan grup shard DB.

AWS CLI

Gunakan `create-db-shard-group` AWS CLI perintah untuk membuat grup shard DB.

Parameter-parameter berikut diperlukan:

- `--db-cluster-identifier`- Cluster DB tempat grup pecahan DB berada.
- `--db-shard-group-identifier`— Nama grup pecahan DB.

  Pengidentifikasi grup shard DB memiliki kendala berikut:

  - Itu harus unik di Akun AWS dan Wilayah AWS di mana Anda membuatnya.
  - Itu harus berisi 1—63 huruf, angka, atau tanda hubung.
  - Karakter pertama harus berupa huruf.
  - Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.

  > **Important**
  >
  > Setelah Anda membuat grup pecahan DB, Anda tidak dapat mengubah pengidentifikasi cluster DB atau pengidentifikasi grup shard DB.

- `--max-acu`- Kapasitas maksimum grup pecahan DB. Gunakan nilai dari 16-6144ACUs.

Parameter berikut ini bersifat opsional:

- `--compute-redundancy`— Apakah akan membuat standbys untuk grup pecahan DB. Parameter ini dapat memiliki nilai berikut:
  - `0`— Membuat grup pecahan DB tanpa standby untuk setiap pecahan. Ini adalah nilai default.
  - `1`— Membuat grup pecahan DB dengan satu siaga komputasi di Availability Zone (AZ) yang berbeda.
  - `2`— Membuat grup pecahan DB dengan dua standby komputasi dalam dua yang berbeda. AZs

  > **Note**
  >
  > Jika Anda mengatur redundansi komputasi ke nilai bukan nol, jumlah total node akan digandakan atau tiga kali lipat. Ini akan dikenakan biaya tambahan.

- `--min-acu`- Kapasitas minimum grup pecahan DB Anda. Setidaknya harus 16ACUs, yang merupakan nilai default.
- `--publicly-accessible|--no-publicly-accessible`— Apakah akan menetapkan alamat IP yang dapat diakses publik ke grup pecahan DB. Akses ke grup shard DB dikendalikan oleh grup keamanan yang digunakan oleh cluster.

  Default-nya adalah `--no-publicly-accessible`.

> **Note**
>
> Anda tidak dapat mengubah pengaturan ini setelah Anda membuat grup shard DB.

Contoh berikut membuat grup pecahan DB di cluster Aurora SQL Postgre DB.

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-db-cluster \
    --db-shard-group-identifier my-new-shard-group \
    --max-acu 1000
```

Outputnya menyerupai contoh berikut.

```
{
    "Status": "CREATING",
    "Endpoint": "my-db-cluster.limitless-ckifpdyyyxxx.us-east-1.rds.amazonaws.com",
    "PubliclyAccessible": false,
    "DBClusterIdentifier": "my-db-cluster",
    "MaxACU": 1000.0,
    "DBShardGroupIdentifier": "my-new-shard-group",
    "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
    "ComputeRedundancy": 0
}
```

# Membuat tabel Basis Data Tanpa Batas Aurora Postgre SQL

Ada tiga jenis tabel yang berisi data Anda di Aurora SQL Postgre Limitless Database:

- Standar — Ini adalah tipe tabel default di Aurora SQL Postgre Limitless Database. Anda membuat tabel standar menggunakan CREATETABLEperintah, dan dapat menjalankan operasi Data Description Language (DDL) dan Data Manipulation Language (DML) pada tabel tersebut.

  Tabel standar bukan tabel terdistribusi. Mereka disimpan di salah satu pecahan yang dipilih secara internal oleh sistem.
- Sharded - Tabel ini didistribusikan di beberapa pecahan. Data dibagi di antara pecahan berdasarkan nilai kolom yang ditunjuk dalam tabel. Kumpulan kolom ini disebut kunci pecahan.
- Referensi - Tabel ini direplikasi pada semua pecahan. Mereka digunakan untuk data referensi yang jarang dimodifikasi, seperti katalog produk dan kode pos.

  Kueri gabungan antara referensi dan tabel sharded dapat dijalankan pada pecahan, menghilangkan pergerakan data yang tidak perlu antara pecahan dan router.

Ada dua cara untuk membuat tabel tanpa batas:

- Membuat tabel tanpa batas dengan menggunakan variabel— Gunakan metode ini ketika Anda ingin membuat tabel sharded dan referensi baru.
- Mengonversi tabel standar ke tabel tanpa batas— Gunakan metode ini ketika Anda ingin mengubah tabel standar yang ada menjadi tabel sharded dan referensi.

Kami juga menyediakan skema sampel untuk Database Aurora SQL Postgre Limitless.

# Membuat tabel tanpa batas dengan menggunakan variabel

Anda dapat menggunakan variabel untuk membuat tabel sharded dan referensi dengan mengatur mode pembuatan tabel. Kemudian tabel yang Anda buat akan menggunakan mode ini sampai Anda mengatur mode yang berbeda.

Gunakan variabel berikut untuk membuat tabel sharded dan referensi:

- `rds_aurora.limitless_create_table_mode`— Tetapkan variabel sesi ini ke `sharded` atau`reference`. Nilai default dari variabel ini adalah`standard`.
- `rds_aurora.limitless_create_table_shard_key`— Atur variabel sesi ini ke array nama kolom untuk digunakan sebagai kunci shard. Variabel ini diabaikan ketika `rds_aurora.limitless_create_table_mode` tidak`sharded`.

  Format nilai sebagai`untyped array literal`, mirip dengan ketika Anda menyisipkan literal ke dalam kolom array. Untuk informasi selengkapnya, lihat Array dalam dokumentasi SQL Postgre.
- `rds_aurora.limitless_create_table_collocate_with`— Tetapkan variabel sesi ini ke nama tabel tertentu untuk menyusun tabel yang baru dibuat dengan tabel itu.

  Jika dua atau lebih tabel di-sharded menggunakan kunci shard yang sama, Anda dapat secara eksplisit menyelaraskan (collocate) tabel tersebut. Ketika dua atau lebih tabel ditempatkan, baris dari tabel dengan nilai kunci shard yang sama ditempatkan pada pecahan yang sama. Kolokasi membantu membatasi beberapa operasi ke pecahan tunggal, yang menghasilkan kinerja yang lebih baik.

> **Note**
>
> Semua kunci primer dan unik harus menyertakan kunci pecahan. Ini berarti bahwa kunci shard adalah bagian dari kunci primer atau unik.
> Tabel tanpa batas memiliki beberapa keterbatasan. Untuk informasi selengkapnya, lihat Keterbatasan DDL dan informasi lainnya untuk Aurora PostgreSQL Limitless Database.

Topik

- Contoh menggunakan variabel untuk membuat tabel tanpa batas
- Tampilan tabel Basis Data Tanpa Batas Aurora Postgre SQL

## Contoh menggunakan variabel untuk membuat tabel tanpa batas

Contoh berikut menunjukkan bagaimana menggunakan variabel-variabel ini untuk membuat tabel sharded dan referensi.

Buat tabel sharded bernama`items`, dengan kunci shard. `id`

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items(id int, val int, item text);
COMMIT;
```

Buat tabel sharded bernama`items`, dengan kunci pecahan terdiri dari `item_id` dan `item_cat` kolom.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
CREATE TABLE items(item_id int, item_cat varchar, val int, item text);
COMMIT;
```

Buat tabel sharded bernama`item_description`, dengan kunci pecahan terdiri dari `item_id` dan `item_cat` kolom, dan kolokasikan dengan `items` tabel dari contoh sebelumnya.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
SET LOCAL rds_aurora.limitless_create_table_collocate_with='items';
CREATE TABLE item_description(item_id int, item_cat varchar, color_id int);
COMMIT;
```

Buat tabel referensi bernama`colors`.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE colors(color_id int primary key, color varchar);
COMMIT;
```

Untuk mengatur ulang variabel `rds_aurora.limitless_create_table_mode` sesi ke`standard`, gunakan pernyataan berikut:

```
RESET rds_aurora.limitless_create_table_mode;
```

Setelah Anda mengatur ulang variabel ini, tabel dibuat sebagai tabel standar, yang merupakan default. Untuk informasi lebih lanjut tentang tabel standar, lihatMengonversi tabel standar ke tabel tanpa batas.

## Tampilan tabel Basis Data Tanpa Batas Aurora Postgre SQL

Anda dapat menemukan informasi tentang tabel Limitless Database dengan menggunakan tampilan berikut.

rds_aurora.limitless_tables

`rds_aurora.limitless_tables`Tampilan berisi informasi tentang tabel tanpa batas dan jenisnya.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | standard    | active       | standard    |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     |
 HASH (order_id)
         2 |     18579 | public      | customer    | active       | sharded     |
 HASH (cust_id)
(4 rows)
```

rds_aurora.limitless_table_collocations

`rds_aurora.limitless_table_collocations`Tampilan berisi informasi tentang tabel sharded collocated. Misalnya, `customers` tabel `orders` dan ditempatkan, dan memiliki yang sama`collocation_id`. `followers`Tabel `users` dan ditempatkan, dan memiliki hal yang sama`collocation_id`.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
          16002 | public      | orders
          16002 | public      | customers
          16005 | public      | users
          16005 | public      | followers
(4 rows)
```

rds_aurora.limitless_table_collocation_distributions

Ini `rds_aurora.limitless_table_collocation_distributions` menunjukkan distribusi kunci untuk setiap kolokasi.

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
          16002 |             6 | -9223372036854775808 | -4611686018427387904
          16002 |             5 | -4611686018427387904 |                    0
          16002 |             4 |                    0 |  4611686018427387904
          16002 |             3 |  4611686018427387904 |  9223372036854775807
          16005 |             6 | -9223372036854775808 | -4611686018427387904
          16005 |             5 | -4611686018427387904 |                    0
          16005 |             4 |                    0 |  4611686018427387904
          16005 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

# Mengonversi tabel standar ke tabel tanpa batas

Anda dapat mengonversi tabel standar menjadi tabel sharded atau referensi. Selama konversi, data dipindahkan dari tabel standar ke tabel terdistribusi, kemudian tabel standar sumber dihapus. Data dipindahkan menggunakan `INSERT INTO SELECT FROM` perintah.

Daftar Isi


- Membuat tabel sharded
- Membuat tabel kolokasi
- Membuat tabel referensi


## Membuat tabel sharded

Anda membuat tabel sharded dengan menjalankan `rds_aurora.limitless_alter_table_type_sharded` prosedur pada tabel standar. Prosedur ini mengambil tabel standar dan daftar kolom, kemudian mendistribusikan tabel yang diberikan menggunakan daftar kolom sebagai kunci pecahan. Prosedur berjalan secara serempak, dan memperoleh `ACCESS EXCLUSIVE` kunci di atas meja.

Setelah prosedur selesai dengan sukses, tabel standar sumber dihapus, dan tabel sharded dengan nama yang sama menjadi tersedia.

`rds_aurora.limitless_alter_table_type_sharded`Prosedur ini menggunakan sintaks berikut:

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('schema.table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn']);
```

Prosedur ini membutuhkan parameter berikut:

- `schema`— Skema database yang berisi tabel yang akan di-sharded. Jika skema tidak ditentukan, prosedur menggunakan. `search_path`
- `table`— Meja yang akan dibelah.
- `shard_key`*n*— Sebuah array kolom tabel untuk digunakan sebagai kunci shard.

  Nilai kunci shard adalah literal string, dan oleh karena itu peka huruf besar/kecil. Jika kunci pecahan berisi satu kutipan ('), gunakan kutipan tunggal lain untuk menghindarinya. Misalnya,

jika kolom tabel diberi nama`customer's id`, gunakan `customer''s id` sebagai kunci shard. Backslashes (\) dan tanda kutip ganda (“) tidak perlu diloloskan.

> **Note**
>
> Semua kunci primer dan unik harus menyertakan kunci shard. Ini berarti bahwa kunci shard adalah bagian dari kunci primer atau unik.
> Dalam tabel sharded, `CHECK` kendala tidak mendukung ekspresi.
> Untuk informasi selengkapnya, lihat Batasan.

Untuk membuat tabel sharded

Contoh berikut menunjukkan cara membuat tabel `customer` sharded dengan kunci shard. `customer_id`

1. Buat tabel standar.

```
CREATE TABLE customer (customer_id INT PRIMARY KEY NOT NULL, zipcode INT, email
 VARCHAR);
```

2. Ubah tabel standar menjadi tabel sharded.

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.customer',
 ARRAY['customer_id']);

postgres=> \d

                   List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
(6 rows)
```

## Membuat tabel kolokasi

Jika dua atau lebih tabel di-sharded menggunakan kunci shard yang sama, Anda dapat secara eksplisit menyelaraskan (collocate) tabel tersebut. Ketika dua atau lebih tabel ditempatkan, baris dari tabel dengan nilai kunci shard yang sama ditempatkan pada pecahan yang sama. Kolokasi membantu membatasi beberapa operasi ke pecahan tunggal, yang menghasilkan kinerja yang lebih baik.

Anda menggunakan `rds_aurora.limitless_alter_table_type_sharded` prosedur dengan sintaks berikut:

```
postgres=> CALL
 rds_aurora.limitless_alter_table_type_sharded('schema.collocated_table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn'], 'schema.sharded_table');
```

Prosedur ini membutuhkan parameter berikut:

- `schema`— Skema database yang berisi tabel yang akan ditempatkan. Jika skema tidak ditentukan, prosedur menggunakan. `search_path`
- `collocated_table`— Meja yang akan ditempatkan.
- `shard_key`*n*— Sebuah array kolom tabel untuk digunakan sebagai kunci shard.

  Anda harus menggunakan kunci shard yang sama seperti untuk tabel sharded asli, termasuk nama kolom dan jenis kolom yang sama.
- `sharded_table`— Meja sharded yang Anda gunakan untuk mengkolokasi. `collocated_table`

Untuk membuat tabel berkolokasi

1. Buat tabel sharded pertama dengan mengikuti prosedur diMembuat tabel sharded.
2. Buat tabel standar untuk tabel collocated.

   ```
   CREATE TABLE mytable2 (customer_id INT PRIMARY KEY NOT NULL, column1 INT, column2
    VARCHAR);
   ```
3. Ubah tabel standar menjadi tabel collocated.

   ```
   postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.mytable2',
   ARRAY['customer_id'], 'public.customer');
   ```

```
postgres=> \d

                    List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | mytable2     | partitioned table | postgres_limitless
 public | mytable2_fs1 | foreign table     | postgres_limitless
 public | mytable2_fs2 | foreign table     | postgres_limitless
 public | mytable2_fs3 | foreign table     | postgres_limitless
 public | mytable2_fs4 | foreign table     | postgres_limitless
 public | mytable2_fs5 | foreign table     | postgres_limitless
(12 rows)
```

## Membuat tabel referensi

Anda membuat tabel referensi dengan menjalankan `rds_aurora.limitless_alter_table_type_reference` prosedur pada tabel standar. Prosedur ini mereplikasi tabel yang diberikan ke semua pecahan dalam grup pecahan DB, dan mengubah jenis tabel menjadi referensi. Prosedur berjalan secara serempak, dan memperoleh `ACCESS EXCLUSIVE` kunci di atas meja.

Setelah prosedur selesai dengan sukses, tabel standar sumber dihapus, dan tabel referensi dengan nama yang sama menjadi tersedia.

`rds_aurora.limitless_alter_table_type_reference`Prosedur ini menggunakan sintaks berikut:

```
postgres=> CALL rds_aurora.limitless_alter_table_type_reference('schema.table');
```

Prosedur yang disimpan membutuhkan parameter berikut:

- `schema`— Skema database yang berisi tabel yang akan direplikasi. Jika skema tidak ditentukan, prosedur menggunakan. `search_path`

- `table`— Tabel yang akan direplikasi.

> **Note**
>
> Tabel standar dari mana Anda membuat tabel referensi harus memiliki kunci utama.
> Dalam tabel referensi, CHECK kendala tidak mendukung ekspresi.
> Fungsi sebelumnya,`limitless_table_alter_type_reference`, sudah usang.

Untuk membuat tabel referensi

Contoh berikut menunjukkan cara membuat tabel `zipcodes` referensi.

1. Buat tabel standar.

```
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);
```

2. Ubah tabel standar menjadi tabel referensi.

```
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcodes');

postgres=> \d

                   List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | zipcodes     | foreign table     | postgres_limitless
(7 rows)
```

   Output menunjukkan tabel `customer` sharded dan tabel `zipcodes` referensi.

## Skema sampel Basis Data Tanpa Batas Aurora Postgre SQL

Kami menyediakan skema sampel berikut untuk Aurora SQL Postgre Limitless Database:

- Skema sampel E-Commerce Tanpa Batas
- Pgbench tanpa batas

Anda dapat menggunakan skema ini untuk dengan cepat membuat database sampel dan memuat data ke dalam tabel Aurora SQL Postgre Limitless Database. Untuk informasi lebih lanjut, lihat GitHub repositori.

# Memuat data ke dalam Database Tanpa Batas Aurora Postgre SQL

Anda dapat memuat data ke dalam tabel Aurora Postgre SQL Limitless Database dengan menggunakan `COPY` perintah atau dengan menggunakan utilitas pemuatan data.

> **Note**
>
> Anda dapat memuat data ke dalam tabel standar, sharded, dan referensi.

Daftar Isi


- Menggunakan COPY perintah dengan Aurora SQL Postgre Limitless Database
  - Menggunakan COPY perintah untuk memuat data ke Aurora SQL Postgre Limitless Database
    - Memisahkan data menjadi beberapa file
  - Menggunakan COPY perintah untuk menyalin data Limitless Database ke file
- Menggunakan utilitas pemuatan data Database Aurora PostgreSQL Limitless
  - Batasan
  - Prasyarat
  - Mempersiapkan database sumber
  - Mempersiapkan database tujuan
  - Membuat kredensil basis data
    - Buat kredensi basis data sumber
    - Buat kredenal database tujuan
  - Menyiapkan otentikasi database dan akses sumber daya menggunakan skrip
    - Skrip pengaturan untuk utilitas pemuatan data
    - Output dari skrip pengaturan utilitas pemuatan data
    - Membersihkan sumber daya yang gagal
  - Menyiapkan otentikasi database dan akses sumber daya secara manual
    - Membuat yang dikelola pelanggan AWS KMS key
    - Membuat rahasia database
    - Membuat peran IAM
    - Memperbarui yang dikelola pelanggan AWS KMS key
    - Menambahkan kebijakan izin peran IAM

- Memuat data dari cluster Aurora Postgre SQL DB atau RDS untuk instans Postgre DB SQL
- Memantau pemuatan data
  - Daftar pekerjaan pemuatan data
  - Melihat detail pekerjaan pemuatan data menggunakan ID pekerjaan
  - Memantau grup CloudWatch log Amazon
  - Pemantauan peristiwa RDS
- Membatalkan pemuatan data

# Menggunakan COPY perintah dengan Aurora SQL Postgre Limitless Database

Anda dapat menggunakan fungsionalitas\ copy di `psql` utilitas untuk mengimpor data ke dan mengekspor data dari Aurora Postgre Limitless Database SQL

## Menggunakan COPY perintah untuk memuat data ke Aurora SQL Postgre Limitless Database

Aurora Postgre SQL Limitless Database kompatibel dengan fungsi\ copy di utilitas untuk mengimpor data. `psql`

Di Limitless Database seperti di Aurora SQL Postgre, berikut ini tidak didukung:

- SSHAkses langsung ke instans DB - Anda tidak dapat menyalin file data (seperti dalam format.csv) ke host instans DB dan dijalankan `COPY` dari file.
- Menggunakan file lokal pada instans DB — Gunakan `COPY ... FROM STDIN` dan`COPY ... TO STDOUT`.

`COPY`Perintah di Postgre SQL memiliki opsi untuk bekerja dengan file lokal (`FROM/TO`) dan mentransmisikan data menggunakan koneksi antara klien dan server (). `STDIN/STDOUT` Untuk informasi lebih lanjut, lihat COPYdi dokumentasi PostgreSQL.

`\copy`Perintah dalam SQL `psql` utilitas Postgre bekerja dengan file lokal di komputer tempat Anda menjalankan klien. `psql` Ini memanggil masing-masing `COPY ... FROM STDIN` atau `COPY ... FROM STDOUT` perintah pada server remote (misalnya, Limitless Database) yang Anda sambungkan. Ia membaca data dari file lokal ke `STDIN` atau menulis dari file tersebut`STDOUT`.

Memisahkan data menjadi beberapa file

Data disimpan di beberapa pecahan di Aurora SQL Postgre Limitless Database. Untuk mempercepat pemuatan data menggunakan`\copy`, Anda dapat membagi data Anda menjadi beberapa file. Kemudian impor secara independen untuk setiap file data dengan menjalankan `\copy` perintah terpisah secara paralel.

Misalnya, Anda memiliki file data input dalam CSV format dengan 3 juta baris untuk diimpor. Anda dapat membagi file menjadi beberapa bagian yang masing-masing memegang 200.000 baris (15 potongan):

```
split -l200000 data.csv data_ --additional-suffix=.csv -d
```

Ini menghasilkan file `data_00.csv` melalui`data_14.csv`. Anda kemudian dapat mengimpor data menggunakan 15 `\copy` perintah paralel, misalnya:

```
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table from '/tmp/data_00.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_01.csv';" postgres_limitless &
...
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_13.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_14.csv';" postgres_limitless
```

Dengan menggunakan teknik ini, jumlah data yang sama diimpor sekitar 10 kali lebih cepat daripada menggunakan satu `\copy` perintah.

## Menggunakan COPY perintah untuk menyalin data Limitless Database ke file

Anda dapat menggunakan perintah\ copy untuk menyalin data dari tabel tanpa batas ke file, seperti yang ditunjukkan pada contoh berikut:

```
postgres_limitless=> \copy test_table TO '/tmp/test_table.csv' DELIMITER ',' CSV
 HEADER;
```

# Menggunakan utilitas pemuatan data Database Aurora PostgreSQL Limitless

Aurora menyediakan utilitas untuk memuat data langsung ke Limitless Database dari cluster Aurora PostgreSQL DB atau RDS untuk instance PostgreSQL DB.

Anda melakukan langkah-langkah berikut untuk menggunakan utilitas pemuatan data:

1. Prasyarat
2. Mempersiapkan database sumber
3. Mempersiapkan database tujuan
4. Membuat kredensil basis data
5. Salah satu dari yang berikut:
   - Menyiapkan otentikasi database dan akses sumber daya menggunakan skrip (direkomendasikan)
   - Menyiapkan otentikasi database dan akses sumber daya secara manual
6. Memuat data dari cluster Aurora Postgre SQL DB atau RDS untuk instans Postgre DB SQL

## Batasan

Utilitas pemuatan data memiliki batasan berikut:

- Tipe data berikut tidak didukung:`enum`,`ARRAY`,`BOX`,`CIRCLE`,`LINE`,`LSEG`,`PATH`, `PG_LSNPG_SNAPSHOT`,`POLYGON`,`TSQUERY`,`TSVECTOR`, dan`TXID_SNAPSHOT`.
- Leading zeroes (`0`) dilucuti dari tipe `VARBIT` data selama pemuatan.
- Migrasi data gagal ketika ada kunci primer komposit pada tabel sumber.
- Migrasi data gagal ketika ada kunci asing pada tabel tujuan.
- Memuat data dari RDS untuk cluster DB PostgreSQL Multi-AZ tidak didukung.

## Prasyarat

Utilitas pemuatan data memiliki prasyarat berikut:

- Database sumber menggunakan Aurora PostgreSQL atau RDS untuk PostgreSQL versi 11.x dan lebih tinggi.

- Database sumber sama Akun AWS dan Wilayah AWS sebagai grup shard DB tujuan.
- Cluster DB sumber atau instans DB berada dalam `available` keadaan.
- Tabel pada database sumber dan database tak terbatas memiliki nama tabel, nama kolom, dan tipe data kolom yang sama.
- Tabel sumber dan tujuan memiliki kunci utama yang menggunakan kolom dan urutan kolom yang sama.
- Anda harus memiliki lingkungan untuk menghubungkan ke database tanpa batas untuk menjalankan perintah pemuatan data. Perintah yang tersedia adalah sebagai berikut:
  - `rds_aurora.limitless_data_load_start`
  - `rds_aurora.limitless_data_load_cancel`
- Untuk CDC:
  - Baik database sumber dan grup pecahan DB tujuan harus menggunakan grup subnet DB, grup keamanan VPC, dan port database yang sama. Pengaturan ini adalah untuk koneksi jaringan ke database sumber dan router dalam grup shard DB.
  - Anda harus mengaktifkan replikasi logis pada database sumber. Pengguna database sumber harus memiliki hak istimewa untuk membaca replikasi logis.

## Mempersiapkan database sumber

Untuk mengakses database sumber untuk pemuatan data, Anda harus mengizinkan lalu lintas jaringan masuk ke sana. Lakukan langkah-langkah berikut ini.

Untuk memungkinkan lalu lintas jaringan ke database sumber

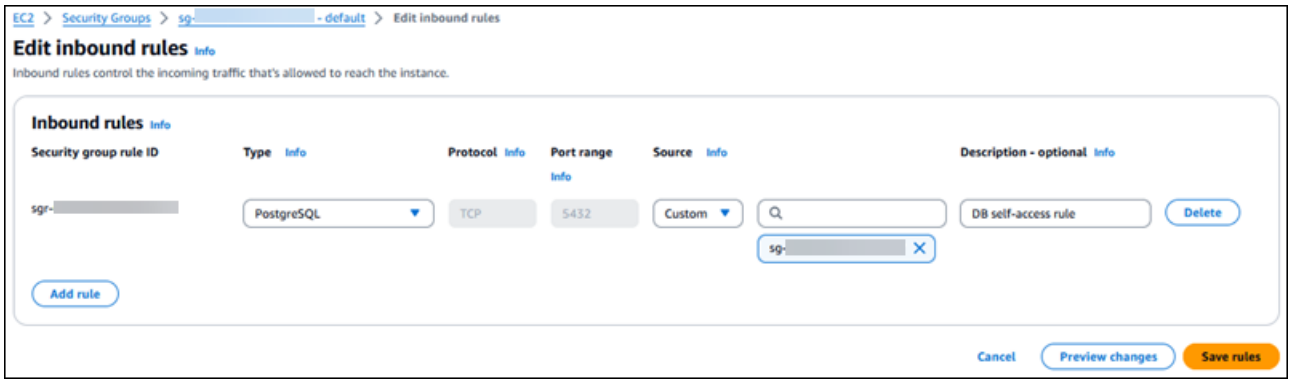
1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Arahkan ke halaman Grup keamanan.
3. Pilih ID grup Keamanan untuk grup keamanan yang digunakan oleh cluster atau instance DB sumber.

   Misalnya, ID grup keamanannya adalah`sg-056a84f1712b77926`.
4. Pada tab Aturan masuk:

   a. Pilih Edit aturan masuk.

   b. Tambahkan aturan masuk baru untuk cluster atau instance DB sumber:

- Rentang port — Port database untuk database sumber, biasanya `5432`
- ID grup keamanan - `sg-056a84f1712b77926` dalam contoh ini

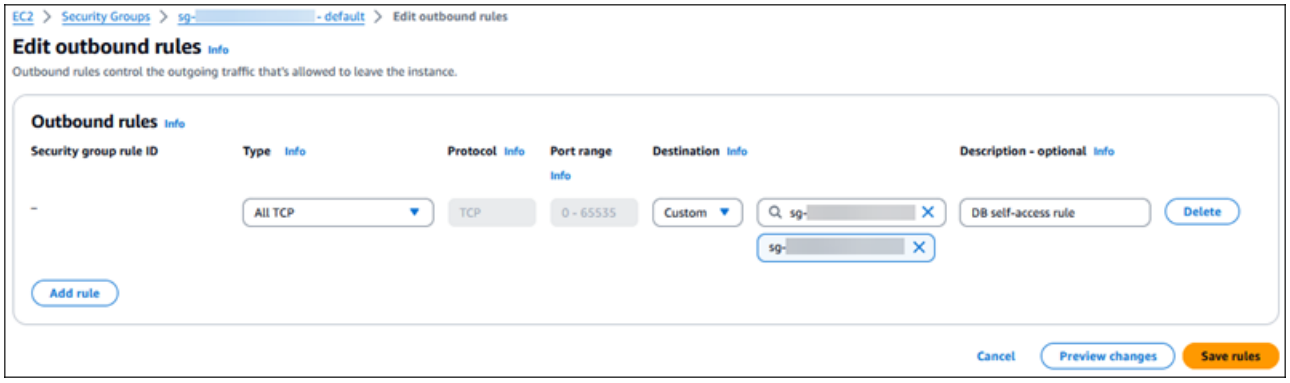



5. Pada tab Aturan keluar:

   a. Pilih Edit aturan keluar.

   b. Tambahkan aturan keluar baru untuk cluster atau instance DB sumber:

   - Port database - `All traffic` (termasuk port`0-65535`)
   - ID grup keamanan - `sg-056a84f1712b77926` dalam contoh ini




6. Masuk ke AWS Management Console dan buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
7. Arahkan ke ACLs halaman Jaringan.
8. Tambahkan konfigurasi ACL jaringan default seperti yang diuraikan dalam ACL jaringan Default.

## Mempersiapkan database tujuan

Ikuti prosedur Membuat tabel Basis Data Tanpa Batas Aurora Postgre SQL untuk membuat tabel tujuan dalam grup pecahan DB.

Tabel tujuan Anda harus memiliki skema, nama tabel, dan kunci utama yang sama dengan tabel sumber.

## Membuat kredensil basis data

Anda harus membuat pengguna basis data di database sumber dan tujuan, dan memberikan hak istimewa yang diperlukan kepada pengguna. Untuk informasi selengkapnya, lihat CREATE USER dan GRANT di dokumentasi PostgreSQL.

### Buat kredensi basis data sumber

Pengguna basis data sumber dilewatkan dalam perintah untuk mulai memuat. Pengguna ini harus memiliki hak istimewa untuk melakukan replikasi dari database sumber.

1. Gunakan pengguna master database (atau pengguna lain dengan `rds_superuser` peran) untuk membuat pengguna database sumber dengan `LOGIN` hak istimewa.

   ```
   CREATE USER source_db_username WITH PASSWORD 'source_db_user_password';
   ```

2. Berikan `rds_superuser` peran tersebut kepada pengguna basis data sumber Anda.

   ```
   GRANT rds_superuser to source_db_username;
   ```

3. Jika Anda menggunakan `full_load_and_cdc` mode, berikan `rds_replication` peran tersebut kepada pengguna database sumber Anda. Peran `rds_replication` memberikan izin untuk mengelola slot logis dan mengalirkan data menggunakan slot logis.

   ```
   GRANT rds_replication to source_db_username;
   ```

### Buat kredenal database tujuan

Pengguna database tujuan harus memiliki izin untuk menulis ke tabel tujuan dalam grup shard DB.

1. Gunakan pengguna master database (atau pengguna lain dengan `rds_superuser` peran) untuk membuat pengguna database tujuan dengan `LOGIN` hak istimewa.

```
CREATE USER destination_db_username WITH PASSWORD 'destination_db_user_password';
```

2. Berikan `rds_superuser` peran tersebut kepada pengguna database tujuan Anda.

```
GRANT rds_superuser to destination_db_username;
```

## Menyiapkan otentikasi database dan akses sumber daya menggunakan skrip

Skrip pengaturan menciptakan satu peran yang dikelola pelanggan AWS KMS key, satu AWS Identity and Access Management (IAM), dan dua AWS Secrets Manager rahasia.

Lakukan langkah-langkah berikut untuk menggunakan skrip pengaturan:

1. Pastikan Anda telah AWS CLI menginstal dan dikonfigurasi dengan Akun AWS kredensi Anda.
2. Instal prosesor `jq` baris perintahJSON. Untuk informasi lebih lanjut, lihat jqlang/jq.
3. Salin file data_loading_script.zip ke komputer Anda, dan ekstrak `data_load_aws_setup_script.sh` file dari itu.
4. Edit skrip untuk mengganti variabel placeholder dengan nilai yang sesuai untuk yang berikut:
   - Anda Akun AWS
   - The Wilayah AWS
   - Kredensi basis data sumber
   - Kredensi basis data tujuan
5. Buka terminal baru di komputer Anda dan jalankan perintah berikut:

```
bash ./data_load_aws_setup_script.sh
```

### Skrip pengaturan untuk utilitas pemuatan data

Kami menyediakan teks `data_load_aws_setup_script.sh` file di sini untuk referensi.

```
#!/bin/bash
# Aurora Limitless data loading - AWS resources setup script #
# Set up the account credentials in advance. #
# Update the following script variables. #

###################################
#### Start of variable section ####

ACCOUNT_ID="12-digit_AWS_account_ID"
REGION="AWS_Region"
DATE=$(date +'%m%d%H%M%S')
RANDOM_SUFFIX="${DATE}"
SOURCE_SECRET_NAME="secret-source-${DATE}"
SOURCE_USERNAME="source_db_username"
```

```
SOURCE_PASSWORD="source_db_password"
DESTINATION_SECRET_NAME="secret-destination-${DATE}"
DESTINATION_USERNAME="destination_db_username"
DESTINATION_PASSWORD="destination_db_password"
DATA_LOAD_IAM_ROLE_NAME="aurora-data-loader-${RANDOM_SUFFIX}"
TMP_WORK_DIR="./tmp_data_load_aws_resource_setup/"

#### End of variable section ####
################################

# Main logic start
echo "DATE - [${DATE}]"
echo "RANDOM_SUFFIX - [${RANDOM_SUFFIX}]"
echo 'START!'

mkdir -p $TMP_WORK_DIR

# Create the symmetric KMS key for encryption and decryption.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_key_response.txt"
aws kms create-key --region $REGION | tee $TMP_FILE_PATH
KMS_KEY_ARN=$(cat $TMP_FILE_PATH | jq -r '.KeyMetadata.Arn')
aws kms create-alias \
    --alias-name alias/"${DATA_LOAD_IAM_ROLE_NAME}-key" \
    --target-key-id $KMS_KEY_ARN \
    --region $REGION

# Create the source secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_source_secret_response.txt"
aws secretsmanager create-secret \
    --name $SOURCE_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$SOURCE_USERNAME\",\"password\":\"$SOURCE_PASSWORD
\"}" \
    --region $REGION \
    | tee $TMP_FILE_PATH
SOURCE_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the destination secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_destination_secret_response.txt"
aws secretsmanager create-secret \
    --name $DESTINATION_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$DESTINATION_USERNAME\",\"password\":
\"$DESTINATION_PASSWORD\"}" \
```

```
    --region $REGION \
    | tee $TMP_FILE_PATH
DESTINATION_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the RDS trust policy JSON file.
# Use only rds.amazonaws.com for RDS PROD use cases.
TRUST_POLICY_PATH="${TMP_WORK_DIR}rds_trust_policy.json"
echo '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}' > $TRUST_POLICY_PATH

# Create the IAM role.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_iam_role_response.txt"
aws iam create-role \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --assume-role-policy-document "file://${TRUST_POLICY_PATH}" \
    --tags Key=assumer,Value=aurora_limitless_table_data_load \
    --region $REGION \
    | tee $TMP_FILE_PATH
IAM_ROLE_ARN=$(cat $TMP_FILE_PATH | jq -r '.Role.Arn')

# Create the permission policy JSON file.
PERMISSION_POLICY_PATH="${TMP_WORK_DIR}data_load_permission_policy.json"
permission_json_policy=$(cat &lt;&lt;EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "$SOURCE_SECRET_ARN",
                "$DESTINATION_SECRET_ARN"
            ]
        },
        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "$KMS_KEY_ARN"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
```

```
            "Resource": "*"
        }
    ]
}
EOF
)
echo $permission_json_policy > $PERMISSION_POLICY_PATH

# Add the inline policy.
aws iam put-role-policy \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --policy-name aurora-limitless-data-load-policy \
    --policy-document "file://${PERMISSION_POLICY_PATH}" \
    --region $REGION

# Create the key policy JSON file.
KEY_POLICY_PATH="${TMP_WORK_DIR}data_load_key_policy.json"
key_json_policy=$(cat &lt;&lt;EOF
{
    "Id": "key-aurora-limitless-data-load-$RANDOM_SUFFIX",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "$IAM_ROLE_ARN"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*"
        }
```

```
    ]
}
EOF
)
echo $key_json_policy > $KEY_POLICY_PATH

# Add the key policy.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_put_key_policy_response.txt"
sleep 10 # sleep 10 sec for IAM role ready
aws kms put-key-policy \
    --key-id $KMS_KEY_ARN \
    --policy-name default \
    --policy "file://${KEY_POLICY_PATH}" \
    --region $REGION \
    | tee $TMP_FILE_PATH

echo 'DONE!'

echo "ACCOUNT_ID : [${ACCOUNT_ID}]"
echo "REGION : [${REGION}]"
echo "RANDOM_SUFFIX : [${RANDOM_SUFFIX}]"
echo "IAM_ROLE_ARN : [${IAM_ROLE_ARN}]"
echo "SOURCE_SECRET_ARN : [${SOURCE_SECRET_ARN}]"
echo "DESTINATION_SECRET_ARN : [${DESTINATION_SECRET_ARN}]"

# Example of a successful run:
# ACCOUNT_ID : [012345678912]
# REGION : [ap-northeast-1]
# RANDOM_SUFFIX : [0305000703]
# IAM_ROLE_ARN : [arn:aws:iam::012345678912:role/aurora-data-loader-0305000703]
# SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-source-0305000703-yQDtow]
# DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-destination-0305000703-5d5Jy8]

# If you want to manually clean up failed resource,
# please remove them in the following order:
# 1. IAM role.
 # aws iam delete-role-policy --role-name Test-Role --policy-name ExamplePolicy --
region us-east-1
# aws iam delete-role --role-name Test-Role --region us-east-1
# 2. Source and destination secrets.
# aws secretsmanager delete-secret --secret-id MyTestSecret --force-delete-without-
recovery --region us-east-1
```

```
# 3. KDM key.
# aws kms schedule-key-deletion --key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab --pending-window-in-days 7
 --region us-east-1
```

Output dari skrip pengaturan utilitas pemuatan data

Contoh berikut menunjukkan output dari menjalankan script yang berhasil.

```
% bash ./data_load_aws_setup_script.sh
DATE - [0305000703]
RANDOM_SUFFIX - [0305000703]
START!
{
    "KeyMetadata": {
        "AWSAccountId": "123456789012",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:ap-
northeast-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-03-05T00:07:49.852000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow",
    "Name": "secret-source-0305000703",
    "VersionId": "a017bebe-a71b-4220-b923-6850c2599c26"
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
destination-0305000703-5d5Jy8",
    "Name": "secret-destination-0305000703",
```

```
    "VersionId": "32a1f989-6391-46b1-9182-f65d242f5eb6"
}
{
    "Role": {
        "Path": "/",
        "RoleName": "aurora-data-loader-0305000703",
        "RoleId": "AROAYPX63ITQOYORQSC6U",
        "Arn": "arn:aws:iam::123456789012:role/aurora-data-loader-0305000703",
        "CreateDate": "2024-03-05T00:07:54+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "rds.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Tags": [
            {
                "Key": "assumer",
                "Value": "aurora_limitless_table_data_load"
            }
        ]
    }
}
DONE!
ACCOUNT_ID : [123456789012]
REGION : [ap-northeast-1]
RANDOM_SUFFIX : [0305000703]
IAM_ROLE_ARN : [arn:aws:iam::123456789012:role/aurora-data-loader-0305000703]
SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow]
DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:123456789012:secret:secret-destination-0305000703-5d5Jy8]
```

### Membersihkan sumber daya yang gagal

Jika Anda ingin membersihkan sumber daya yang gagal secara manual, hapus dengan urutan berikut:

1. IAMperan, misalnya:

```
aws iam delete-role-policy \
--role-name Test-Role \
--policy-name ExamplePolicy

aws iam delete-role \
--role-name Test-Role
```

2. Rahasia sumber dan tujuan, misalnya:

```
aws secretsmanager delete-secret \
--secret-id MyTestSecret \
--force-delete-without-recovery
```

3. KMSkunci, misalnya:

```
aws kms schedule-key-deletion \
--key-id arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab
 \
--pending-window-in-days 7
```

Kemudian Anda dapat mencoba kembali skrip.

## Menyiapkan otentikasi database dan akses sumber daya secara manual

Proses manual untuk menyiapkan otentikasi database dan akses sumber daya memiliki langkah-langkah berikut:

1. Membuat yang dikelola pelanggan AWS KMS key
2. Menambahkan kebijakan izin peran IAM
3. Membuat rahasia database
4. Membuat peran IAM
5. Memperbarui yang dikelola pelanggan AWS KMS key

Proses ini opsional, dan melakukan tugas yang sama seperti diMenyiapkan otentikasi database dan akses sumber daya menggunakan skrip. Kami merekomendasikan menggunakan skrip.

Membuat yang dikelola pelanggan AWS KMS key

Ikuti prosedur dalam Membuat kunci enkripsi simetris untuk membuat kunci KMS yang dikelola pelanggan. Anda juga dapat menggunakan kunci yang ada jika memenuhi persyaratan ini.

Untuk membuat kunci KMS yang dikelola pelanggan

1. Masuk ke AWS Management Console dan buka AWS KMS konsol di https://console.aws.amazon.com/kms.
2. Arahkan ke halaman kunci terkelola Pelanggan.
3. Pilih Buat kunci.
4. Pada halaman tombol Konfigurasi:
    a. Untuk Key type, pilih Symmetric.
    b. Untuk penggunaan Kunci, pilih Enkripsi dan dekripsi.
    c. Pilih Berikutnya.
5. Pada halaman Tambahkan label, masukkan Alias seperti**`limitless`**, lalu pilih Berikutnya.
6. Pada halaman Tentukan izin administratif kunci, pastikan kotak centang Izinkan administrator kunci untuk menghapus kunci ini dipilih, lalu pilih Berikutnya.
7. Pada halaman Tentukan izin penggunaan kunci, pilih Berikutnya.
8. Pada halaman Ulasan, pilih Selesai.

Anda memperbarui kebijakan kunci nanti.

Rekam Nama Sumber Daya Amazon (ARN) dari kunci KMS untuk digunakan. Menambahkan kebijakan izin peran IAM

Untuk informasi tentang penggunaan AWS CLI untuk membuat kunci KMS yang dikelola pelanggan, lihat create-key dan create-alias.

Membuat rahasia database

Untuk memungkinkan utilitas pemuatan data mengakses tabel database sumber dan tujuan, Anda membuat dua rahasia di AWS Secrets Manager: satu untuk database sumber dan satu untuk database tujuan. Rahasia ini menyimpan nama pengguna dan kata sandi untuk mengakses basis data sumber dan tujuan.

Ikuti prosedur di Buat AWS Secrets Manager rahasia untuk membuat rahasia pasangan kunci-nilai.

Untuk membuat rahasia database

1. Buka konsol Secrets Manager di https://console.aws.amazon.com/secretsmanager/.
2. Pilih Simpan rahasia baru.
3. Pada halaman Pilih jenis rahasia:
   a. Untuk tipe Rahasia, pilih Jenis rahasia lainnya.
   b. Untuk pasangan kunci/nilai, pilih tab Plaintext.
   c. Masukkan kode JSON berikut, di mana *sourcedbreader* dan *sourcedbpassword* merupakan kredensil untuk pengguna basis data sumber dari. Buat kredensi basis data sumber

```
{
    "username":"sourcedbreader",
    "password":"sourcedbpassword"
}
```

   d. Untuk kunci Enkripsi, pilih kunci KMS yang Anda buatMembuat yang dikelola pelanggan AWS KMS key, misalnya`limitless`.
   e. Pilih Berikutnya.

4. Pada halaman Konfigurasi rahasia, masukkan nama Rahasia, seperti**source_DB_secret**, lalu pilih Berikutnya.
5. Pada halaman Konfigurasi rotasi - opsional, pilih Berikutnya.
6. Pada halaman Review, pilih Store.
7. Ulangi prosedur untuk rahasia database tujuan:

    a. Masukkan kode JSON berikut, dari mana *destinationdbwriter* dan *destinationdbpassword* merupakan kredensil untuk pengguna database tujuan. Buat kredenal database tujuan

    ```
    {
        "username":"destinationdbwriter",
        "password":"destinationdbpassword"
    }
    ```

    b. Masukkan nama Rahasia, seperti**destination_DB_secret**.

ARNs Catat rahasia untuk digunakanMenambahkan kebijakan izin peran IAM.

Membuat peran IAM

Pemuatan data mengharuskan Anda untuk menyediakan akses ke AWS sumber daya. Untuk memberikan akses, Anda membuat peran `aurora-data-loader` IAM dengan mengikuti prosedur dalam Membuat peran untuk mendelegasikan izin ke pengguna IAM.

Untuk membuat peran IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Arahkan ke halaman Peran.
3. Pilih Buat peran.
4. Pada halaman Pilih entitas tepercaya:

    a. Untuk jenis entitas Tepercaya, pilih Kebijakan kepercayaan khusus.

    b. Masukkan kode JSON berikut untuk kebijakan kepercayaan khusus:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

c. Pilih Berikutnya.

5. Pada halaman Tambahkan izin, pilih Berikutnya.
6. Pada halaman Nama, tinjau, dan buat:

   a. Untuk nama Peran, masukkan **`aurora-data-loader`** atau nama lain yang Anda inginkan.

   b. Pilih Tambah tag, dan masukkan tag berikut:

   - Kunci: **`assumer`**
   - Nilai: **`aurora_limitless_table_data_load`**

> **Important**
>
> Database Aurora PostgreSQL Limitless hanya dapat mengambil peran IAM yang memiliki tag ini.

   c. Pilih Buat peran.

Memperbarui yang dikelola pelanggan AWS KMS key

Ikuti prosedur di Mengubah kebijakan kunci untuk menambahkan peran IAM `aurora-data-loader` ke kebijakan kunci default.

Untuk menambahkan peran IAM ke kebijakan kunci

1. Masuk ke AWS Management Console dan buka AWS KMS konsol di https://console.aws.amazon.com/kms.
2. Arahkan ke halaman kunci terkelola Pelanggan.
3. Pilih tombol KMS yang Anda buatMembuat yang dikelola pelanggan AWS KMS key, misalnya`limitless`.
4. Pada tab Kebijakan kunci, untuk pengguna kunci, pilih Tambah.
5. Di jendela Tambah pengguna kunci, pilih nama peran IAM yang Anda buatMembuat peran IAM, misalnya aurora-data-loader.
6. Pilih Tambahkan.

Menambahkan kebijakan izin peran IAM

Anda harus menambahkan kebijakan izin ke peran IAM yang Anda buat. Hal ini memungkinkan utilitas pemuatan data Database Aurora PostgreSQL Limitless untuk mengakses AWS sumber daya terkait untuk membangun koneksi jaringan dan mengambil rahasia kredensi DB sumber dan tujuan.

Untuk informasi selengkapnya, lihat Memodifikasi peran.

Untuk menambahkan kebijakan izin

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Arahkan ke halaman Peran.
3. Pilih peran IAM yang Anda buatMembuat peran IAM, misalnya aurora-data-loader.
4. Pada tab Izin, untuk kebijakan Izin pilih Tambahkan izin, lalu Buat kebijakan sebaris.
5. Pada halaman Tentukan izin, pilih editor JSON.
6. Salin dan tempel template berikut ke editor JSON, ganti placeholder dengan rahasia database dan ARNs kunci KMS Anda.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:source_DB_secret-ABC123",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:destination_DB_secret-456DEF"
            ]
        },        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/aa11bb22-
####-####-####-fedcba123456"
```

```
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

7. Periksa kesalahan dan perbaiki.
8. Pilih Berikutnya.
9. Pada halaman Tinjau dan buat, masukkan nama Kebijakan seperti**`data_loading_policy`**, lalu pilih Buat kebijakan.

## Memuat data dari cluster Aurora Postgre SQL DB atau RDS untuk instans Postgre DB SQL

Setelah Anda menyelesaikan pengaturan sumber daya dan otentikasi, sambungkan ke titik akhir cluster dan panggil prosedur `rds_aurora.limitless_data_load_start` tersimpan dari database tanpa batas, seperti. `postgres_limitless` Basis data tanpa batas adalah database pada grup pecahan DB tempat Anda ingin memigrasikan data.

Fungsi ini menghubungkan secara asinkron di latar belakang ke database sumber yang ditentukan dalam perintah, membaca data dari sumber, dan memuat data ke pecahan. Untuk kinerja yang lebih baik, data dimuat menggunakan thread paralel. Fungsi mengambil snapshot point-in-time tabel dengan menjalankan `SELECT` perintah untuk membaca data tabel yang disediakan dalam perintah.

Anda dapat memuat data ke dalam tabel sharded, referensi, dan standar.

Anda dapat memuat data di database, skema, atau tingkat tabel dalam `rds_aurora.limitless_data_load_start` panggilan.

- Database - Anda dapat memuat satu database pada satu waktu di setiap panggilan, tanpa batas pada skema atau jumlah tabel dalam database.
- Skema - Anda dapat memuat maksimum 15 skema di setiap panggilan, tanpa batas jumlah tabel dalam setiap skema.
- Tabel — Anda dapat memuat maksimal 15 tabel di setiap panggilan.

> **Note**
>
> Fitur ini tidak menggunakan RDS snapshot Amazon atau point-in-time isolasi database. Untuk konsistensi di seluruh tabel, kami sarankan untuk mengkloning database sumber dan menunjuk ke database kloning itu sebagai sumbernya.

Prosedur tersimpan menggunakan sintaks berikut:

```
CALL rds_aurora.limitless_data_load_start('source_type',
    'source_DB_cluster_or_instance_ID',
    'source_database_name',
    'streaming_mode',
    'data_loading_IAM_role_arn',
    'source_DB_secret_arn',
    'destination_DB_secret_arn',
    'ignore_primary_key_conflict_boolean_flag',
    'is_dry_run',
    (optional parameter) schemas/tables => ARRAY['name1', 'name2', ...]);
```

Parameter input adalah sebagai berikut:

- `source_type`— Jenis sumber: `aurora_postgresql` atau `rds_postgresql`
- `source_DB_cluster_or_instance_ID`— Sumber pengidentifikasi cluster Aurora Postgre SQL DB atau RDS untuk pengidentifikasi instans Postgre DB SQL
- `source_database_name`Nama database sumber, seperti *`postgres`*
- `streaming_mode`— Apakah akan menyertakan change data capture (CDC): `full_load` atau `full_load_and_cdc`
- `data_loading_IAM_role_arn`— IAM Peran Nama Sumber Daya Amazon (ARN) untuk `aurora-data-loader`
- `source_DB_secret_arn`— Rahasia sumber DB ARN
- `destination_DB_secret_arn`— Rahasia DB tujuan ARN
- `ignore_primary_key_conflict_boolean_flag`— Apakah akan melanjutkan jika konflik kunci utama terjadi:
  - Jika disetel ke`true`, pemuatan data mengabaikan perubahan baru untuk baris dengan konflik kunci primer.
  - Jika disetel ke`false`, pemuatan data menimpa baris yang ada pada tabel tujuan saat menghadapi konflik kunci primer.
- `is_dry_run`— Apakah akan menguji bahwa pekerjaan pemuatan data dapat terhubung ke database sumber dan tujuan:
  - Jika disetel ke`true`, uji koneksi tanpa memuat data
  - Jika diatur ke`false`, memuat data

- (opsional) `schemas` atau `tables` — Sebuah array skema atau tabel untuk memuat. Anda dapat menentukan salah satu dari berikut ini:
  - Daftar tabel dalam format `tables => ARRAY['schema1.table1', 'schema1.table2', 'schema2.table1', ...]`
  - Daftar skema dalam format `schemas => ARRAY['schema1', 'schema2', ...]`

  Jika Anda tidak menyertakan parameter ini, seluruh database sumber yang ditentukan akan dimigrasikan.

Parameter output adalah ID pekerjaan dengan pesan.

Contoh berikut menunjukkan bagaimana menggunakan prosedur `rds_aurora.limitless_data_load_start` tersimpan untuk memuat data dari cluster Aurora SQL Postgre DB.

```
CALL rds_aurora.limitless_data_load_start('aurora_postgresql',
    'my-db-cluster',
    'postgres',
    'full_load_and_cdc',
    'arn:aws:iam::123456789012:role/aurora-data-loader-8f2c66',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-source-8f2c66-EWrr0V',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-destination-8f2c66-
d04fbD',
    'true',
    'false',
    tables => ARRAY['public.customer', 'public.order', 'public.orderdetails']);

INFO: limitless data load job id 1688761223647 is starting.
```

## Memantau pemuatan data

Aurora Postgre SQL Limitless Database menyediakan beberapa cara untuk memantau pekerjaan pemuatan data:

- Daftar pekerjaan pemuatan data
- Melihat detail pekerjaan pemuatan data menggunakan ID pekerjaan
- Memantau grup CloudWatch log Amazon
- Pemantauan peristiwa RDS

### Daftar pekerjaan pemuatan data

Anda dapat terhubung ke titik akhir cluster dan menggunakan `rds_aurora.limitless_data_load_jobs` tampilan untuk mencantumkan pekerjaan pemuatan data.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_jobs LIMIT 6;

    job_id     |  status   | message |     source_db_identifier      | source_db_name |
 full_load_complete_time |
   progress_details                                                                 |
     start_time       |   last_updated_time    |  streaming_mode   | source_engine_type
 | ignore_primary_key_conflict | is_dryrun
---------------+-----------+---------+-------------------------------+---------------
+-------------------------
+-------------------------------------------------------------------------------------------------------------------------
+------------------------+-----------------------+------------------
+--------------------+-----------------------------+-----------
 1725697520693 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:48:15+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "9
 of 9 tables loaded", "COMPLETED_AT": "2024/09/07 08:48:15+00", "RECORDS_MIGRATED":
 600003}} | 2024-09-07 08:47:13+00 | 2024-09-07 08:48:15+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725696114225 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:24:20+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "3
 of 3 tables loaded", "COMPLETED_AT": "2024/09/07 08:24:20+00", "RECORDS_MIGRATED":
 200001}} | 2024-09-07 08:23:56+00 | 2024-09-07 08:24:20+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725696067630 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:23:45+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "6
 of 6 tables loaded", "COMPLETED_AT": "2024/09/07 08:23:45+00", "RECORDS_MIGRATED":
```

```
 400002}} | 2024-09-07 08:23:10+00 | 2024-09-07 08:23:45+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725694221753 | CANCELED  |         | persistent-kdm-auto-source-01 | postgres
    |                         | {}

           | 2024-09-07 07:31:18+00 | 2024-09-07 07:51:49+00 | full_load_and_cdc |
 aurora_postgresql  | t                           | f
 1725691698210 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:51+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:51+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:42+00 | 2024-09-07 07:10:52+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725691695049 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:48+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:48+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:41+00 | 2024-09-07 07:10:48+00 | full_load         |
 aurora_postgresql  | t                           | f
(6 rows)
```

Catatan Job dihapus setelah 90 hari.

Melihat detail pekerjaan pemuatan data menggunakan ID pekerjaan

Jika Anda mengetahui ID pekerjaan, Anda dapat menyambung ke titik akhir klaster dan menggunakan `rds_aurora.limitless_data_load_job_details` tampilan untuk melihat detail pekerjaan pemuatan data tersebut, termasuk nama tabel, status pekerjaan, dan jumlah baris yang dimuat. Anda bisa mendapatkan ID pekerjaan dalam respons terhadap fungsi awal pemuatan data, atau dari `rds_aurora.limitless_data_load_jobs` tampilan.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_job_details WHERE
 job_id='1725696114225';

job_id        | destination_table_name | destination_schema_name | start_time
      | status    | full_load_rows | full_load_total_rows | full_load_complete_time |
 cdc_insert | cdc_update | cdc_delete
--------------+------------------------+-------------------------
+-------------------------+-----------+----------------+---------------------
+-------------------------+------------+------------+------------
1725696114225 | standard_1             | public                  | 2024-09-07
 08:23:57+00 | COMPLETED | 100000         | 100000              | 2024-09-07
 08:24:08+00  | 0          | 0          | 0
```

```
1725696114225 | standard_2             | public                  | 2024-09-07
 08:24:08+00 | COMPLETED | 100000        | 100000               | 2024-09-07
 08:24:17+00  | 0          | 0          | 0
1725696114225 | standard_3             | public                  | 2024-09-07
 08:24:18+00 | COMPLETED | 1             | 1                    | 2024-09-07
 08:24:20+00  | 0          | 0          | 0
1725696114225 | standard_4             | public                  | 2024-09-07
 08:23:58+00 | PENDING   | 0             | 0                    |
    | 0          | 0          | 0
(4 rows)
```

Catatan Job dihapus setelah 90 hari.

Memantau grup CloudWatch log Amazon

Setelah status pekerjaan pemuatan data berubah`RUNNING`, Anda dapat memeriksa progres runtime menggunakan Amazon CloudWatch Logs.

Untuk memantau aliran CloudWatch log

Masuk ke AWS Management Console dan buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.

1. Arahkan ke Log, lalu Log grup.
2. Pilih grup log/aws/rds/aurora-limitless-database.
3. Cari aliran log pekerjaan pemuatan data Anda dengan job_id.

   Aliran log memiliki pola Data-Load-Job -. *job_id*
4. Pilih aliran log untuk melihat peristiwa log.

Setiap aliran log menunjukkan peristiwa yang berisi status pekerjaan dan jumlah baris yang dimuat ke tabel tujuan Aurora Postgre SQL Limitless Database. Jika pekerjaan pemuatan data gagal, log kesalahan juga dibuat yang menunjukkan status kegagalan dan alasannya.

Catatan Job dihapus setelah 90 hari.

Pemantauan peristiwa RDS

Pekerjaan pemuatan data juga menerbitkan RDS peristiwa, misalnya ketika pekerjaan berhasil, gagal, atau dibatalkan. Anda dapat melihat peristiwa dari database tujuan.

Untuk informasi selengkapnya, lihat Acara grup shard DB.

## Membatalkan pemuatan data

Untuk membatalkan pekerjaan pemuatan data, panggil prosedur `rds_aurora.limitless_data_load_cancel` tersimpan, dengan ID pekerjaan sebagai parameter input. Anda memanggil prosedur tersimpan ini dari database yang sama di grup pecahan DB tempat pekerjaan pemuatan data tertentu dimulai. Sebagai contoh:

```
CALL rds_aurora.limitless_data_load_cancel(12345);

INFO: limitless data load job with id 12345 is canceling without rollback.
```

Anda tidak dapat membatalkan pekerjaan pemuatan data yang tidak ada atau tidak berjalan di grup pecahan DB yang sama.

Utilitas pemuatan data Aurora Postgre SQL Limitless Database meninggalkan data yang dimuat di tabel tujuan tanpa rollback, seperti yang ditunjukkan respons. Jika Anda tidak ingin menyimpan data yang dimuat, sebaiknya potong tabel tujuan.

# Meminta Basis Data Tanpa Batas Aurora Postgre SQL

Aurora Postgre SQL Limitless Database kompatibel dengan sintaks Postgre untuk kueri. SQL Anda dapat menanyakan Database Limitless Anda menggunakan `psql` atau utilitas koneksi lain yang bekerja dengan SQL Postgre. Untuk menjalankan kueri, Anda terhubung ke titik akhir tanpa batas seperti yang ditunjukkan pada. Menghubungkan ke cluster DB Database PostgreSQL Limitless Aurora Anda

Semua SQL `SELECT` kueri Postgre didukung di Aurora Postgre Limitless Database. SQL Namun, kueri dilakukan pada dua lapisan:

1. Router tempat klien mengirimkan kueri
2. Pecahan tempat data berada

Kinerja tergantung pada kueri database dengan cara yang memungkinkannya untuk mencapai tingkat tinggi pemrosesan simultan dari kueri yang berbeda pada pecahan yang berbeda. Query pertama kali diuraikan di layer transaksi terdistribusi (router). Sebelum merencanakan eksekusi kueri, ada fase analisis untuk mengidentifikasi lokasi untuk semua hubungan yang berpartisipasi dalam kueri. Jika semua relasi adalah tabel sharded dengan kunci shard yang difilter pada shard yang sama, atau tabel referensi, maka perencanaan kueri dilewati pada lapisan router dan sepenuhnya didorong ke bawah ke pecahan untuk perencanaan dan eksekusi. Proses ini mengurangi jumlah perjalanan pulang pergi antara node yang berbeda (router dan shard) dan menghasilkan kinerja yang lebih baik dalam banyak kasus. Untuk informasi selengkapnya, lihat Kueri pecahan tunggal di Aurora PostgreSQL Limitless Database.

> **Note**
>
> Mungkin ada kasus tertentu, seperti produk Cartesian (cross join), di mana kueri berkinerja lebih baik dengan mengambil data secara terpisah dari pecahan.

Untuk informasi selengkapnya tentang rencana eksekusi kueri, lihat EXPLAIN diReferensi Basis Data Tanpa Batas Aurora Postgre SQL. Untuk informasi umum tentang kueri, lihat Pertanyaan dalam dokumentasi SQL Postgre.

Topik

- Kueri pecahan tunggal di Aurora PostgreSQL Limitless Database

- Kueri terdistribusi di Aurora SQL Postgre Limitless Database
- Penelusuran kueri terdistribusi di log Postgre di Aurora Postgre SQL Limitless Database SQL
- Kebuntuan yang didistribusikan di Aurora SQL Postgre Limitless Database

## Kueri pecahan tunggal di Aurora PostgreSQL Limitless Database

Kueri shard tunggal adalah kueri yang dapat dijalankan langsung pada pecahan sambil mempertahankan semantik SQL ACID. Ketika kueri seperti itu ditemui oleh perencana kueri pada router, perencana mendeteksinya dan melanjutkan untuk menekan seluruh kueri SQL ke pecahan yang sesuai.

Optimalisasi ini mengurangi jumlah perjalanan pulang pergi jaringan dari router ke pecahan, meningkatkan kinerja. Saat ini optimasi ini dilakukan untuk`INSERT`,`SELECT`,`UPDATE`, dan `DELETE` kueri.

Topik

- Contoh kueri pecahan tunggal
- Pembatasan untuk kueri pecahan tunggal
- Bergabung sepenuhnya memenuhi syarat (eksplisit)
- Menyetel kunci pecahan aktif

### Contoh kueri pecahan tunggal

Dalam contoh berikut, kita memiliki tabel sharded`customers`, dengan kunci shard`customer_id`, dan tabel referensi. `zipcodes`

SELECT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM customers WHERE
 customer_id = 100;

                        QUERY PLAN
-----------------------------------------------------------
 Foreign Scan
   Output: customer_id, other_id, customer_name, balance
   Remote SQL:  SELECT customer_id,
     other_id,
     customer_name,
```

```
     balance
    FROM public.customers
   WHERE (customer_id = 100)
 Single Shard Optimized
(9 rows)
```

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM orders
    LEFT JOIN zipcodes ON orders.zipcode_id = zipcodes.zipcode_id
    WHERE customer_id = 11;

                                              QUERY PLAN

-----------------------------------------------------------------------------------------------
 Foreign Scan
   Output: customer_id, order_id, zipcode_id, customer_name, balance,
 zipcodes.zipcode_id, zipcodes.city
   Remote SQL:  SELECT orders.customer_id,
     orders.order_id,
     orders.zipcode_id,
     orders.customer_name,
     orders.balance,
     zipcodes.zipcode_id,
     zipcodes.city
    FROM (public.orders
      LEFT JOIN public.zipcodes ON ((orders.zipcode_id = zipcodes.zipcode_id)))
   WHERE (orders.customer_id = 11)
 Single Shard Optimized
(13 rows)
```

## INSERT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) INSERT INTO customers
    (customer_id, other_id, customer_name, balance)
    VALUES (1, 10, 'saikiran', 1000);

                      QUERY PLAN
------------------------------------------------------
 Insert on public.customers
   ->  Result
         Output: 1, 10, 'saikiran'::text, '1000'::real
 Single Shard Optimized
(4 rows)
```

UPDATE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) UPDATE orders SET balance = balance +
 100
    WHERE customer_id = 100;

                                        QUERY PLAN

-----------------------------------------------------------------------------------------------
 Update on public.orders
   Foreign Update on public.orders_fs00002 orders_1
   ->  Foreign Update
         Remote SQL:  UPDATE public.orders SET balance = (balance + (100)::double
 precision)
   WHERE (customer_id = 100)
 Single Shard Optimized
(6 rows)
```

DELETE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) DELETE FROM orders
    WHERE customer_id = 100 and balance = 0;

                             QUERY PLAN
---------------------------------------------------------------------
 Delete on public.orders
   Foreign Delete on public.orders_fs00002 orders_1
   ->  Foreign Delete
         Remote SQL:  DELETE FROM public.orders
   WHERE ((customer_id = 100) AND (balance = (0)::double precision))
 Single Shard Optimized
(6 rows)
```

## Pembatasan untuk kueri pecahan tunggal

Kueri single-shard memiliki batasan berikut:

Fungsi

Jika kueri pecahan tunggal berisi fungsi, kueri memenuhi syarat untuk pengoptimalan pecahan tunggal hanya jika salah satu dari kondisi berikut berlaku:

- Fungsinya tidak dapat diubah. Untuk informasi selengkapnya, lihat Volatilitas fungsi.
- Fungsi ini bisa berubah, tetapi terdaftar dalam `rds_aurora.limitless_distributed_functions` tampilan. Untuk informasi selengkapnya, lihat Distribusi fungsi.

Tampilan

Jika kueri berisi satu atau beberapa tampilan, optimasi shard tunggal dinonaktifkan untuk kueri jika memiliki salah satu kondisi berikut:

- Setiap tampilan memiliki `security_barrier` atribut.
- Objek yang digunakan dalam kueri memerlukan beberapa hak istimewa pengguna. Misalnya, kueri berisi dua tampilan, dan tampilan dijalankan di bawah dua pengguna yang berbeda.

```
CREATE VIEW v1 AS SELECT customer_name FROM customers c WHERE c.customer_id =  1;
CREATE VIEW v2 WITH (security_barrier) AS SELECT customer_name FROM customers c
 WHERE c.customer_id =  1;

postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v1;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Output: customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 c  (cost=0.00..24.12 rows=6 width=32)
           Output: c.customer_name
           Filter: (c.customer_id = 1)
         Query Identifier: -6005737533846718506
   Remote SQL:  SELECT customer_name
    FROM ( SELECT c.customer_name
             FROM public.customers c
            WHERE (c.customer_id = 1)) v1
 Query Identifier: -5754424854414896228
(12 rows)


postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v2;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Foreign Scan on public.customers_fs00001 c  (cost=100.00..128.41 rows=7 width=32)
   Output: c.customer_name
   Remote Plans from Shard postgres_s3:
```

```
         Seq Scan on public.customers_ts00001 customers  (cost=0.00..24.12 rows=6
 width=32)
           Output: customers.customer_name
           Filter: (customers.customer_id = 1)
         Query Identifier: 4136563775490008117
   Remote SQL: SELECT customer_name FROM public.customers WHERE ((customer_id = 1))
 Query Identifier: 5056054318010163757
(9 rows)
```

MEMPERSIAPKAN dan MENGEKSEKUSI pernyataan

Aurora PostgreSQL Limitless Database mendukung optimasi single-shard untuk disiapkan,, dan pernyataan. `SELECT` `UPDATE` `DELETE`

Namun, jika Anda menggunakan pernyataan yang disiapkan untuk `PREPARE` dan `EXECUTE` dengan `plan_cache_mode` disetel ke`'force_generic_plan'`, perencana kueri menolak pengoptimalan pecahan tunggal untuk kueri tersebut.

PL/PGSQL

Kueri dengan PL/pgSQL variables are run as implicitly prepared statements. If a query contains any PL/pgSQL variabel, perencana kueri menolak optimasi shard tunggal.

Optimasi didukung dalam PL/pgSQL block if the statement doesn't contain any PL/pgSQL variabel.

## Bergabung sepenuhnya memenuhi syarat (eksplisit)

Optimasi single-shard didasarkan pada eliminasi partisi. Pengoptimal PostgreSQL menghilangkan partisi berdasarkan kondisi konstan. Jika Aurora PostgreSQL Limitless Database menemukan bahwa semua partisi dan tabel yang tersisa berada pada pecahan yang sama, ini menandai kueri yang memenuhi syarat untuk optimasi shard tunggal. Semua kondisi filter harus eksplisit agar eliminasi partisi berfungsi. Aurora PostgreSQL Limitless Database tidak dapat menghilangkan partisi tanpa satu atau lebih predikat gabungan atau memfilter predikat pada kunci pecahan setiap tabel sharded dalam pernyataan.

Asumsikan bahwa kita telah mempartisi`customers`,`orders`, dan `order_details` tabel berdasarkan kolom. `customer_id` Dalam skema ini, aplikasi mencoba menyimpan semua data untuk pelanggan pada satu pecahan.

Pertimbangkan kueri berikut:

```
SELECT * FROM
    customers c, orders o, order_details od
WHERE c.customer_id = o.customer_id
    AND od.order_id = o.order_id
    AND c.customer_id = 1;
```

Kueri ini mengambil semua data untuk pelanggan (`c.customer_id = 1`). Data untuk pelanggan ini ada pada satu pecahan, tetapi Aurora PostgreSQL Limitless Database tidak memenuhi syarat kueri ini sebagai kueri pecahan tunggal. Proses pengoptimal untuk kueri adalah sebagai berikut:

1. Pengoptimal dapat menghilangkan partisi untuk `customers` dan `orders` berdasarkan kondisi berikut:

   ```
   c.customer_id = 1
   c.customer_id = o.customer_id
   o.customer_id =  1 (transitive implicit condition)
   ```

2. Pengoptimal tidak dapat menghilangkan partisi apa pun`order_details`, karena tidak ada kondisi konstan di atas meja.
3. Optimizer menyimpulkan bahwa ia telah membaca semua partisi dari. `order_details` Oleh karena itu, kueri tidak dapat memenuhi syarat untuk pengoptimalan pecahan tunggal.

Untuk menjadikan ini kueri pecahan tunggal, kami menambahkan kondisi gabungan eksplisit berikut:

```
o.customer_id = od.customer_id
```

Kueri yang diubah terlihat seperti ini:

```
SELECT * FROM
    customers c, orders o,  order_details od
WHERE c.customer_id = o.customer_id
     AND o.customer_id = od.customer_id
     AND od. order_id = o. order_id
 AND c.customer_id =  1;
```

Sekarang pengoptimal dapat menghilangkan partisi untuk. `order_details` Kueri baru menjadi kueri pecahan tunggal dan memenuhi syarat untuk pengoptimalan.

## Menyetel kunci pecahan aktif

Fitur ini memungkinkan Anda untuk mengatur kunci shard tunggal saat menanyakan database, menyebabkan semua SELECT dan kueri DHTML ditambahkan dengan kunci shard sebagai predikat konstan. Fitur ini berguna jika Anda telah bermigrasi ke Aurora PostgreSQL Limitless Database dan telah mendenormalisasi skema dengan menambahkan kunci shard ke tabel.

Anda dapat menambahkan predikat kunci shard secara otomatis ke logika SQL yang ada, tanpa mengubah semantik kueri. Menambahkan predikat kunci shard aktif dilakukan hanya untuk tabel yang kompatibel.

Fitur kunci pecahan aktif menggunakan `rds_aurora.limitless_active_shard_key` variabel, yang memiliki sintaks berikut:

```
SET [session | local] rds_aurora.limitless_active_shard_key = '{"col1_value",
 "col2_value", ...}';
```

Beberapa pertimbangan tentang kunci pecahan aktif dan kunci asing:

- Sebuah tabel sharded dapat memiliki kendala kunci asing jika tabel induk dan anak ditempatkan dan kunci asing adalah superset dari kunci shard.
- Tabel sharded dapat memiliki kendala kunci asing ke tabel referensi.
- Tabel referensi dapat memiliki kendala kunci asing ke tabel referensi lain.

Asumsikan bahwa kita memiliki `customers` tabel yang di-sharded pada `customer_id` kolom.

```
BEGIN;
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE customers(customer_id int PRIMARY KEY, name text , email text);
COMMIT;
```

Dengan set kunci pecahan aktif, kueri memiliki transformasi berikut.

SELECT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
SELECT * FROM customers;
```

```
-- This statement is changed to:
SELECT * FROM customers WHERE customer_id = '123'::int;
```

INSERT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
INSERT INTO customers(name, email) VALUES('Alex', 'alex@example.com');

-- This statement is changed to:
INSERT INTO customers(customer_id, name, email) VALUES('123'::int, 'Alex',
 'alex@example.com');
```

UPDATE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
UPDATE customers SET email = 'alex_new_email@example.com';

-- This statement is changed to:
UPDATE customers SET email = 'alex_new_email@example.com' WHERE customer_id =
 '123'::int;
```

DELETE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
DELETE FROM customers;

-- This statement is changed to:
DELETE FROM customers WHERE customer_id = '123'::int;
```

Gabungan

Saat melakukan operasi gabungan pada tabel dengan kunci pecahan aktif, predikat kunci shard secara otomatis ditambahkan ke semua tabel yang terlibat dalam gabungan. Penambahan otomatis predikat kunci shard ini hanya terjadi ketika semua tabel dalam kueri milik grup kolokasi yang sama. Jika kueri melibatkan tabel dari grup kolokasi yang berbeda, kesalahan akan muncul sebagai gantinya.

Asumsikan bahwa kita juga memiliki `orders` dan `order_details` tabel yang ditempatkan dengan `customers` tabel.

```
SET local rds_aurora.limitless_create_table_mode='sharded';
```

```
SET local rds_aurora.limitless_create_table_collocate_with='customers';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE orders (id int , customer_id int, total_amount int, date date);
CREATE TABLE order_details (id int , order_id int, customer_id int, product_name
 VARCHAR(100), price int);
COMMIT;
```

Ambil 10 faktur pesanan terakhir untuk pelanggan yang ID pelanggannya 10.

```
SET rds_aurora.limitless_active_shard_key = '{"10"}';
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    order_details.order_id = orders.order_id AND
    customers.customer_id = 10
    order by order_date limit 10;
```

Kueri ini diubah menjadi sebagai berikut:

```
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    orders.order_id = order_details.order_id AND
    customers.customer_id = 10 AND
    order_details.customer_id = 10 AND
    orders.customer_id = 10 AND
    ORDER BY "order_date" LIMIT 10;
```

Tabel yang kompatibel dengan kunci pecahan aktif

Predikat kunci shard ditambahkan hanya ke tabel yang kompatibel dengan kunci pecahan aktif. Sebuah tabel dianggap kompatibel jika memiliki jumlah kolom yang sama dalam kunci shard seperti yang ditentukan dalam `rds_aurora.limitless_active_shard_key` variabel. Jika kueri melibatkan tabel yang tidak kompatibel dengan kunci pecahan aktif, sistem akan memunculkan kesalahan alih-alih melanjutkan kueri.

Misalnya:

```
-- Compatible table
SET rds_aurora.limitless_active_shard_key = '{"10"}';

-- The following query works because the customers table is sharded on one column.
SELECT * FROM customers;
```

```
-- Incompatible table
SET rds_aurora.limitless_active_shard_key = '{"10","20"}';

-- The following query raises a error because the customers table isn't sharded on
 two columns.
 SELECT * FROM customers;
```

# Kueri terdistribusi di Aurora SQL Postgre Limitless Database

Kueri terdistribusi berjalan pada router dan lebih dari satu pecahan. Kueri diterima oleh salah satu router. Router membuat dan mengelola transaksi terdistribusi, yang dikirim ke pecahan yang berpartisipasi. Pecahan membuat transaksi lokal dengan konteks yang disediakan oleh router, dan kueri dijalankan.

Ketika transaksi dilakukan, router menggunakan protokol komit dua fase yang dioptimalkan jika diperlukan, dan Multi Version Concurrency Concurrency Control (MVCC) berbasis waktu untuk menyediakan ACIDsemantik dalam sistem database terdistribusi.

Berbasis waktu MVCC mencatat waktu komit untuk setiap transaksi dan menggunakan waktu mulai transaksi untuk menghasilkan waktu snapshot data. Untuk mengidentifikasi apakah transaksi dilakukan (terlihat) dengan snapshot pembaca, database membandingkan waktu komitmennya dengan waktu snapshot. Jika waktu komitmennya kurang dari waktu snapshot pembaca, itu terlihat; jika tidak, tidak terlihat. Di bawah protokol ini, Anda akan selalu berharap untuk melihat data yang sangat konsisten di Aurora SQL Postgre Limitless Database.

# Penelusuran kueri terdistribusi di log Postgre di Aurora Postgre SQL Limitless Database SQL

Penelusuran kueri terdistribusi adalah alat untuk melacak dan mengkorelasikan kueri dalam SQL log Postgre di seluruh Database Tanpa Batas Aurora Postgre. SQL Di Aurora PostgreSQL, Anda menggunakan ID transaksi untuk mengidentifikasi transaksi. Namun di Aurora Postgre SQL Limitless Database, ID transaksi dapat diulang di berbagai router. Oleh karena itu, kami menyarankan Anda menggunakan ID penelusuran sebagai gantinya di Limitless Database.

Kasus penggunaan utama adalah sebagai berikut:

- Gunakan `rds_aurora.limitless_get_last_trace_id()` fungsi untuk menemukan ID penelusuran unik dari kueri terakhir yang dijalankan di sesi saat ini. Kemudian cari grup log cluster DB di Amazon CloudWatch Logs menggunakan ID penelusuran itu untuk menemukan semua log terkait.

  Anda dapat menggunakan `log_min_error_statement` parameter `log_min_messages` dan bersama-sama untuk mengontrol volume log yang dicetak dan mencetak pernyataan yang berisi ID penelusuran.
- Gunakan `log_min_duration_statement` parameter untuk menentukan waktu proses di mana semua kueri mencetak durasi proses dan ID penelusuran. Waktu berjalan ini kemudian dapat dicari di grup log cluster DB di CloudWatch Log untuk membantu menentukan node bottleneck dan upaya pengoptimalan perencana.

  `log_min_duration_statement`Parameter memungkinkan ID penelusuran untuk semua node terlepas dari nilai `log_min_messages` dan `log_min_error_statement` parameter.

Topik


- ID Penelusuran
- Menggunakan penelusuran kueri
- Contoh log


## ID Penelusuran

Inti dari fitur ini adalah pengenal unik yang dikenal sebagai ID penelusuran. ID penelusuran adalah string 31 digit yang ditambahkan ke baris STATEMENT log log Postgre, bertindak

sebagai pengidentifikasi yang tepat untuk menghubungkan SQL log yang terkait dengan eksekusi kueri tertentu. Contohnya adalah `1126253375719408504000000000011` dan `1126253375719408495000000000090`.

ID penelusuran terdiri dari yang berikut:

- Pengenal transaksi — 20 digit pertama, mengidentifikasi transaksi secara unik.
- Command identifier — 30 digit pertama, yang menunjukkan kueri individu dalam suatu transaksi.

  Jika lebih dari 4.294.967.294 kueri dijalankan di dalam blok transaksi eksplisit, perintah Identifier membungkusnya. 1 Ketika ini terjadi, Anda akan diberi tahu melalui `LOG` pesan berikut di log PostgreSQL:

  ```
  wrapping around the tracing ID back to 1 after running 4294967294 (4.2 billion or
   2^32-2) queries inside of an explicit transaction block
  ```
- Pengidentifikasi tipe node — Digit terakhir, yang menunjukkan apakah node berfungsi sebagai router koordinator (`1`) atau node peserta (`0`).

Contoh berikut menggambarkan komponen ID penelusuran:

- `1126253375719408504000000000011`:
  - Pengidentifikasi transaksi — `1126253375719408504`
  - Command identifier — `1126253375719408504000000000001` menunjukkan perintah pertama di blok transaksi
  - Pengidentifikasi tipe node - `1` menunjukkan router koordinator
- `1126253375719408495000000000090`:
  - Pengidentifikasi transaksi — `1126253375719408495`
  - Command identifier — `1126253375719408495000000000009` menunjukkan perintah kesembilan di blok transaksi
  - Pengidentifikasi tipe node - `0` menunjukkan node peserta

## Menggunakan penelusuran kueri

Lakukan tugas-tugas berikut untuk menggunakan penelusuran kueri:

1. Pastikan penelusuran diaktifkan.

Anda dapat memeriksa menggunakan perintah berikut:

```
SHOW rds_aurora.limitless_log_distributed_trace_id;
```

Ini diaktifkan secara default (on). Jika tidak diaktifkan, atur menggunakan perintah berikut:

```
SET rds_aurora.limitless_log_distributed_trace_id = on;
```

2. Kontrol volume log yang dicetak dengan mengonfigurasi tingkat keparahan log.

   Volume log dikendalikan oleh `log_min_messages` parameter. `log_min_error_statement`Parameter digunakan untuk mencetak `STATEMENT` garis dengan ID penelusuran. Keduanya diatur secara `ERROR` default. Anda dapat memeriksa menggunakan perintah berikut:

   ```
   SHOW log_min_messages;
   SHOW log_min_error_statement;
   ```

   Untuk memperbarui tingkat keparahan dan mencetak `STATEMENT` baris untuk sesi saat ini, gunakan perintah berikut dengan salah satu tingkat keparahan ini:

```
SET log_min_messages = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO | NOTICE
 | WARNING | ERROR | LOG | FATAL | PANIC';
SET log_min_error_statement = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO
 | NOTICE | WARNING | ERROR | LOG | FATAL | PANIC';
```

   Sebagai contoh:

   ```
   SET log_min_messages = 'WARNING';
   SET log_min_error_statement = 'WARNING';
   ```

3. Aktifkan pencetakan ID penelusuran di log di atas waktu proses tertentu.

   `log_min_duration_statement`Parameter dapat diubah ke waktu proses kueri minimum di atas mana kueri mencetak baris log dengan durasi proses, bersama dengan penelusuran IDs melintasi cluster DB. Parameter ini diatur ke secara `-1` default, yang berarti itu dinonaktifkan. Anda dapat memeriksa menggunakan perintah berikut:

```
SHOW log_min_duration_statement;
```

Mengubahnya untuk `0` mencetak durasi dan melacak ID di log untuk setiap kueri di seluruh cluster DB. Anda dapat mengaturnya `0` untuk sesi saat ini dengan menggunakan perintah berikut:

```
SET log_min_duration_statement = 0;
```

4. Dapatkan ID penelusuran.

   Setelah menjalankan kueri (bahkan di dalam blok transaksi eksplisit), panggil `rds_aurora.limitless_get_last_trace_id` fungsi untuk mendapatkan ID penelusuran dari kueri terakhir yang dijalankan:

   ```
   SELECT * FROM rds_aurora.limitless_get_last_trace_id();
   ```

   Fungsi ini mengembalikan identifier transaksi dan pengidentifikasi perintah. Itu tidak mengembalikan pengidentifikasi tipe simpul.

   ```
   => SELECT * FROM customers;
    customer_id | fname | lname
   -------------+-------+-------
   (0 rows)

   => SELECT * FROM rds_aurora.limitless_get_last_trace_id();
    transaction_identifier |       command_identifier
   ------------------------+--------------------------------
    10104661421959001813   | 10104661421959001813000000001
   (1 row)
   ```

   Fungsi mengembalikan baris kosong untuk kueri yang tidak didistribusikan, karena mereka tidak memiliki ID penelusuran.

   ```
   => SET search_path = public;
   SET

   => SELECT * FROM rds_aurora.limitless_get_last_trace_id();
    transaction_identifier | command_identifier
   ------------------------+--------------------
                           |
   ```

```
(1 row)
```

> **Note**
>
> Untuk VACUUM dan ANALYZE kueri, pernyataan durasi tidak dicatat dengan ID penelusuran. Oleh karena itu, `limitless_get_last_trace_id()` tidak mengembalikan ID penelusuran. Jika VACUUM atau ANALYZE merupakan operasi yang berjalan lama, Anda dapat menggunakan kueri berikut untuk mendapatkan ID penelusuran untuk operasi tersebut:
>
> ```
> SELECT * FROM rds_aurora.limitless_stat_activity
> WHERE distributed_tracing_id IS NOT NULL;
> ```
>
> Jika server berhenti sebelum Anda dapat menemukan ID penelusuran terakhir, Anda harus mencari SQL log Postgre secara manual untuk menemukan penelusuran IDs di log dari tepat sebelum kegagalan.

5. Cari ID penelusuran di seluruh log cluster DB menggunakan CloudWatch.

   Gunakan CloudWatch Wawasan untuk menanyakan grup log cluster DB, seperti yang ditunjukkan pada contoh berikut.

   - Kueri untuk pengidentifikasi transaksi tertentu dan semua perintah berjalan di dalamnya:

     ```
     fields @timestamp, @message
     | filter @message like /10104661421959001813/
     | sort @timestamp desc
     ```

   - Kueri untuk pengidentifikasi perintah tertentu:

     ```
     fields @timestamp, @message
     | filter @message like /10104661421959001813000000001/
     | sort @timestamp desc
     ```

6. Periksa semua log di seluruh cluster DB yang dihasilkan oleh kueri terdistribusi.

## Contoh log

Contoh berikut menunjukkan penggunaan query tracing.

Mengkorelasikan log untuk kueri rawan kesalahan

Dalam contoh ini `TRUNCATE` perintah dijalankan di atas `customers` meja ketika tabel itu tidak ada.

Tanpa penelusuran kueri

File SQL log Postgre pada router koordinator:

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:CONTEXT: remote
 SQL command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:STATEMENT:
 truncate customers;
```

File SQL log Postgre pada pecahan peserta:

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:STATEMENT:
 truncate customers;
```

Log ini khas. Mereka tidak memiliki pengidentifikasi tepat yang diperlukan untuk dengan mudah menghubungkan kueri di seluruh cluster DB.

Dengan penelusuran kueri

File SQL log Postgre pada router koordinator:

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:CONTEXT: remote SQL
 command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT: /* tid =
 1126253375719408502700000000011 */ truncate customers;
```

File SQL log Postgre pada pecahan peserta:

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR:  failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT:  /* tid
 = 1126253375719408502700000000010 */ truncate customers;
```

Dengan adanya penelusuran kueri, setiap baris log ditambahkan dengan pengidentifikasi unik 31 digit. Di sini, `1126253375719408502700000000011` dan `1126253375719408502700000000010` mewakili penelusuran IDs untuk koordinator dan node peserta, masing-masing.

- Pengidentifikasi transaksi — `11262533757194085027`
- Pengidentifikasi perintah: - `1126253375719408502700000000011`
- Pengidentifikasi tipe node — Digit terakhir, 1 atau`0`, menunjukkan router koordinator dan node peserta, masing-masing.

Mengkorelasikan log untuk menemukan waktu proses kueri pada berbagai node

Dalam contoh ini, `log_min_duration_statement` parameter telah diperbarui `0` untuk mencetak durasi untuk semua kueri.

Tanpa penelusuran kueri

```
2024-01-15 07:28:46 UTC:[local]:postgres@postgres_limitless:[178322]:LOG: duration:
 12.779 ms statement: select * from customers;
```

Dengan penelusuran kueri

File SQL log Postgre pada router koordinator:

```
2024-01-15 07:32:08 UTC:[local]:postgres@postgres_limitless:[183877]:LOG: duration:
 12.618 ms statement: /* tid = 045766956624049708840000000011 */ select * from
 customers;
```

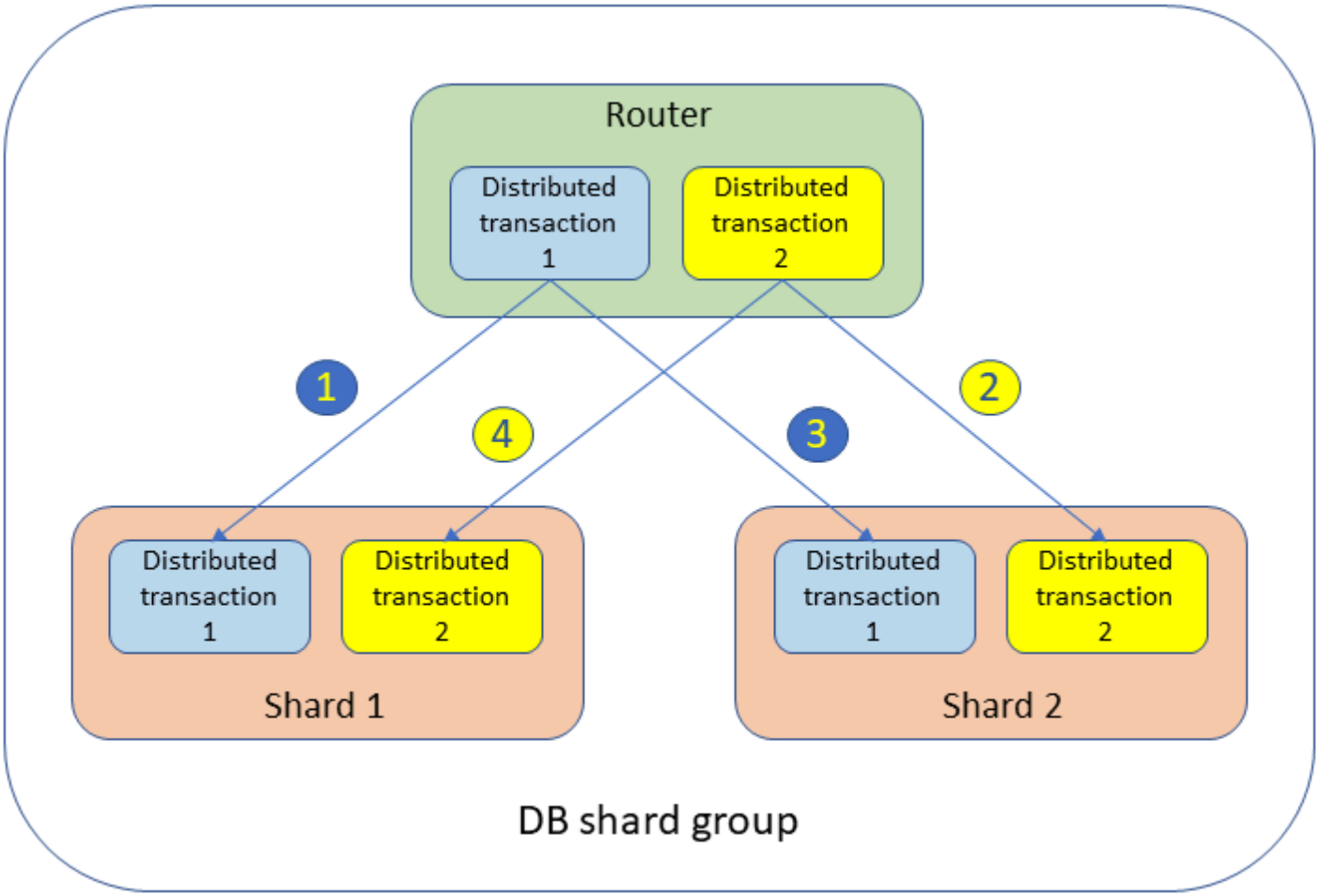
File SQL log Postgre pada pecahan peserta:

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.279 ms statement: /* tid = 045766956624049708840000000010 */ START
 TRANSACTION ISOLATION LEVEL READ COMMITTED
```

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.249 ms parse <unnamed>: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.398 ms bind <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.019 ms execute <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.073 ms statement: /* tid = 045766956624049708840000000010 */ COMMIT
 TRANSACTION
```

# Kebuntuan yang didistribusikan di Aurora SQL Postgre Limitless Database

Dalam grup pecahan DB, kebuntuan dapat terjadi antara transaksi yang didistribusikan di antara berbagai router dan pecahan. Misalnya, dua transaksi terdistribusi bersamaan yang menjangkau dua pecahan dijalankan, seperti yang ditunjukkan pada gambar berikut.




Transaksi mengunci tabel dan membuat acara tunggu di dua pecahan sebagai berikut:

1. Transaksi terdistribusi 1:

```
UPDATE table SET value = 1 WHERE key = 'shard1_key';
```

   Ini memegang kunci pada pecahan 1.

2. Transaksi terdistribusi 2:

```
UPDATE table SET value = 2 WHERE key = 'shard2_key';
```

Ini memegang kunci pada pecahan 2.

3. Transaksi terdistribusi 1:

```
UPDATE table SET value = 3 WHERE key = 'shard2_key';
```

Transaksi terdistribusi 1 sedang menunggu di shard 2.

4. Transaksi terdistribusi 2:

```
UPDATE table SET value = 4 WHERE key = 'shard1_key';
```

Transaksi terdistribusi 2 sedang menunggu di shard 1.

Dalam skenario ini, baik shard 1 maupun shard 2 tidak melihat masalah: transaksi 1 sedang menunggu transaksi 2 pada shard 2, dan transaksi 2 sedang menunggu transaksi 1 pada shard 1. Dari pandangan global, transaksi 1 sedang menunggu transaksi 2, dan transaksi 2 sedang menunggu transaksi 1. Situasi di mana dua transaksi pada dua pecahan berbeda menunggu satu sama lain disebut kebuntuan terdistribusi.

Aurora Postgre SQL Limitless Database dapat mendeteksi dan menyelesaikan kebuntuan terdistribusi secara otomatis. Router dalam grup pecahan DB diberitahu ketika transaksi menunggu terlalu lama untuk memperoleh sumber daya. Router yang menerima notifikasi mulai mengumpulkan informasi yang diperlukan dari semua router dan pecahan dalam grup pecahan DB. Router kemudian melanjutkan untuk mengakhiri transaksi yang berpartisipasi dalam kebuntuan terdistribusi, sampai sisa transaksi dalam grup shard DB dapat dilanjutkan tanpa diblokir satu sama lain.

Anda menerima kesalahan berikut ketika transaksi Anda adalah bagian dari kebuntuan terdistribusi, dan kemudian diakhiri oleh router:

```
ERROR: aborting transaction participating in a distributed deadlock
```

Parameter cluster `rds_aurora.limitless_distributed_deadlock_timeout` DB menetapkan waktu untuk setiap transaksi untuk menunggu sumber daya sebelum memberi tahu router untuk memeriksa kebuntuan terdistribusi. Anda dapat meningkatkan nilai parameter jika beban kerja Anda kurang rentan terhadap situasi kebuntuan. Defaultnya adalah `1000` milidetik (1 detik).

Siklus kebuntuan terdistribusi dipublikasikan ke SQL log Postgre ketika kebuntuan lintas-simpul ditemukan dan diselesaikan. Informasi tentang setiap proses yang merupakan bagian dari kebuntuan meliputi:

- Node koordinator yang memulai transaksi
- ID transaksi virtual (xid) dari transaksi pada node koordinator, dalam format *backend_id*/*backend_local_xid*
- ID sesi transaksi yang didistribusikan

# Mengelola Basis Data Tanpa Batas Aurora Postgre SQL

Topik berikut menjelaskan cara mengelola cluster DB Basis Data Aurora Postgre SQL Limitless Anda.

Topik


- Database dan pertimbangan ukuran tabel
- Merebut kembali ruang penyimpanan dengan menyedot debu


## Database dan pertimbangan ukuran tabel

Untuk Aurora Postgre SQL Limitless Database, pada setiap pecahan tabel sharded dibagi menjadi sejumlah irisan tabel yang bervariasi tergantung pada berapa banyak pecahan yang tersedia dalam kelompok pecahan DB. Setiap irisan meja dapat tumbuh hingga 32 TiB, tetapi setiap pecahan memiliki kapasitas maksimum 128 TiB. Tabel referensi memiliki batas ukuran 32 TiB untuk seluruh grup pecahan DB.

> **Note**
>
> Kapasitas maksimum setiap node (router atau shard) adalah 128 TiB, karena ini adalah kapasitas maksimum untuk cluster Aurora SQL Postgre DB.

Jumlah maksimum hubungan per database (termasuk tabel, tampilan, dan indeks) di Aurora Postgre dan Aurora SQL Postgre Limitless Database adalah SQL 1.431.650.303.

Untuk informasi lebih lanjut, lihat SQLbatas Lampiran K. Postgre dalam dokumentasi Postgre dan. SQL Batas ukuran Amazon Aurora

# Merebut kembali ruang penyimpanan dengan menyedot debu

Postgre SQL Multiversion Concurrency Control (MVCC) membantu menjaga integritas data dengan menyimpan salinan internal baris yang diperbarui atau dihapus sampai transaksi dilakukan atau dibatalkan. Salinan ini, juga disebut tupel, dapat menyebabkan kembung meja jika tidak dibersihkan secara teratur. Postgre SQL memesan transaksi berdasarkan transaksinyaIDs, dan Postgre SQL menggunakan ID transaksi berbasis MVCC untuk mengontrol visibilitas Tuple dan memberikan isolasi transaksi. Setiap transaksi menetapkan snapshot data, dan setiap tuple memiliki versi. Baik snapshot maupun versi berbasis ID transaksi.

Untuk membersihkan data, `VACUUM` utilitas melakukan empat fungsi utama di PostgreSQL:

- `VACUUM`— Menghapus versi baris kedaluwarsa, membuat ruang tersedia untuk digunakan kembali.
- `VACUUM FULL`— Menyediakan defragmentasi lengkap dengan menghapus versi baris mati dan memadatkan tabel, mengurangi ukuran dan meningkatkan efisiensi.
- `VACUUM FREEZE`— Melindungi dari masalah sampul ID transaksi dengan menandai versi baris yang lebih lama sebagai beku.
- `VACUUM ANALYZE`— Menghapus versi baris mati dan memperbarui statistik perencanaan kueri database. Ini adalah kombinasi dari fungsi `VACUUM` dan `ANALYZE` fungsi. Untuk informasi lebih lanjut tentang cara `ANALYZE` kerja di Aurora Postgre SQL Limitless Database, lihat. ANALYZE

Seperti halnyaMVCC, menyedot debu di Aurora Postgre berbasis ID transaksi. SQL Jika ada transaksi yang sedang berlangsung saat penyedot debu dimulai, baris yang masih terlihat oleh transaksi tersebut tidak akan dihapus.

Untuk informasi lebih lanjut tentang `VACUUM` utilitas, lihat VACUUMdi dokumentasi PostgreSQL. Untuk informasi lebih lanjut tentang `VACUUM` dukungan di Aurora Postgre SQL Limitless Database, lihat. VACUUM

Topik

- AUTOVACUUM
- Penyedot debu berbasis waktu di Aurora Postgre Limitless Database SQL
- Menggunakan statistik database untuk menyedot debu
- Perbedaan perilaku menyedot debu antara Aurora Postgre dan Aurora SQL Postgre Limitless Database SQL

## AUTOVACUUM

Aurora Postgre SQL menggunakan VACUUM dan AUTOVACUUM utilitas untuk menghapus tupel yang tidak dibutuhkan. Mekanisme yang mendasari AUTOVACUUM dan manual VACUUM adalah sama; satu-satunya perbedaan adalah otomatisasi.

AUTOVACUUMdi Aurora Postgre SQL dan Aurora SQL Postgre Limitless Database adalah kombinasi dari dan utilitas. VACUUM ANALYZE AUTOVACUUMmenentukan database dan tabel mana yang harus dibersihkan, sesuai dengan aturan yang telah ditentukan, seperti persentase tupel mati dan jumlah sisipan.

Misalnya, AUTOVACUUM "bangun" secara berkala untuk melakukan pembersihan. Interval dikendalikan oleh autovacuum_naptime parameter. Nilai default adalah 1 menit. Nilai default untuk AUTOVACUUM dan parameter VACUUM konfigurasi sama untuk Aurora Postgre SQL Limitless Database seperti untuk Aurora Postgre. SQL

AUTOVACUUMDaemon, jika diaktifkan, secara otomatis mengeluarkan ANALYZE perintah setiap kali konten tabel telah berubah secara memadai. Di Aurora Postgre SQL Limitless Database, AUTOVACUUM masalah ANALYZE pada router dan pecahan.

Untuk informasi selengkapnya tentang parameter penyimpanan AUTOVACUUM daemon dan tabel yang terkaitAUTOVACUUM, lihat Daemon Autovacuum dan Parameter penyimpanan dalam dokumentasi Postgre. SQL

## Penyedot debu berbasis waktu di Aurora Postgre Limitless Database SQL

Aurora Postgre SQL Limitless Database adalah sistem terdistribusi, yang berarti bahwa beberapa contoh dapat terlibat dalam transaksi. Oleh karena itu, visibilitas berbasis ID transaksi tidak berlaku. Sebaliknya, Aurora Postgre SQL Limitless Database menggunakan visibilitas berbasis waktu, karena transaksi IDs tidak "disatukan" di seluruh instance, tetapi waktu dapat "disatukan" di seluruh instance. Setiap snapshot transaksi dan setiap versi Tuple mematuhi waktu, bukan ID transaksi. Untuk lebih spesifik, snapshot transaksi memiliki waktu mulai snapshot, dan Tuple memiliki waktu pembuatan (ketika INSERT atau UPDATE terjadi) dan waktu penghapusan (ketika terjadi). DELETE

Untuk menjaga konsistensi data di seluruh instance dalam grup pecahan DB, Aurora Postgre SQL Limitless Database harus memastikan bahwa penyedot debu tidak menghapus tupel apa pun yang masih terlihat oleh transaksi aktif apa pun di grup pecahan DB. Oleh karena itu, menyedot debu di Aurora SQL Postgre Limitless Database juga berbasis waktu. Aspek lain VACUUM tetap sama, termasuk bahwa untuk berjalan VACUUM pada tabel tertentu, pengguna harus memiliki akses ke tabel itu.

> **Note**
>
> Kami sangat menyarankan agar Anda tidak membiarkan transaksi terbuka untuk jangka waktu yang lama.
> Penyedot debu berbasis waktu mengkonsumsi lebih banyak memori daripada penyedot debu berbasis ID transaksi.

Contoh berikut menggambarkan cara kerja penyedot debu berbasis waktu.

1. Tabel pelanggan didistribusikan di empat pecahan.
2. Transaksi 1 dimulai dengan pembacaan berulang, dan menargetkan hanya satu pecahan (pecahan 1). Transaksi ini tetap terbuka.

   Transaksi 1 lebih tua dari transaksi lain yang dimulai setelahnya.
3. Transaksi 2 dimulai kemudian, dan menghapus semua tupel dari tabel, lalu melakukan.
4. Jika `AUTOVACUUM` atau manual `VACUUM` mencoba membersihkan tupel mati (mati karena transaksi 2), itu tidak menghapus apa pun.

   Hal ini berlaku tidak hanya untuk pecahan 1, tetapi juga untuk pecahan 2-4, karena transaksi 1 mungkin masih perlu mengakses tupel ini. Mereka masih terlihat untuk transaksi 1 karenaMVCC.

Langkah terakhir dicapai melalui sinkronisasi, sehingga semua pecahan menyadari transaksi 1, meskipun transaksi 1 tidak menyentuh semuanya.

## Menggunakan statistik database untuk menyedot debu

Untuk mendapatkan informasi tentang tupel yang mungkin perlu Anda bersihkan, gunakan tampilan limitless_stat_all_tables, yang bekerja mirip dengan pg_stat_all_tables. Contoh berikut menanyakan tampilan.

```
SELECT * FROM rds_aurora.limitless_stat_all_tables WHERE relname LIKE '%customer%';
```

Demikian pula, untuk statistik database gunakan limitless_stat_database alih-alih pg_stat_database, dan limitless_stat_activity alih-alih pg_stat_activity.

Untuk memeriksa penggunaan disk tabel, gunakan fungsi limitless_stat_relation_sizes, yang bekerja mirip dengan pg_relation_size. Contoh berikut query fungsi.

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customer');
```

Untuk melacak kemajuan VACUUM operasi pada Aurora Postgre SQL Limitless Database, gunakan tampilan limitless_stat_progress_vacuum alih-alih pg_stat_progress_vacuum. Contoh berikut menanyakan tampilan.

```
SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;
```

Untuk informasi selengkapnya, silakan lihat Tampilan Database Tanpa Batas Aurora PostgreSQL dan Fungsi Basis Data Tanpa Batas Aurora Postgre SQL.

## Perbedaan perilaku menyedot debu antara Aurora Postgre dan Aurora SQL Postgre Limitless Database SQL

Beberapa perbedaan lain antara Aurora Postgre SQL dan Aurora SQL Postgre Limitless Database dalam cara kerja penyedot debu adalah sebagai berikut:

- Aurora Postgre SQL melakukan VACUUM operasi pada transaksi IDs hingga transaksi tertua yang sedang berlangsung. Jika tidak ada transaksi yang sedang berlangsung dalam databaseVACUUM, lakukan operasi hingga transaksi terakhir.
- Aurora Postgre SQL Limitless Database menyinkronkan snapshot waktu tertua setiap 10 detik. Oleh karena itu, VACUUM mungkin tidak melakukan operasi pada setiap transaksi yang dijalankan dalam 10 detik terakhir.

Untuk informasi tentang dukungan VACUUM di Aurora Postgre SQL Limitless Database, lihat di. VACUUM Referensi Basis Data Tanpa Batas Aurora Postgre SQL

# Memantau Database Tanpa Batas Aurora PostgreSQL

Anda dapat menggunakan Amazon CloudWatch, Enhanced Monitoring, dan Performance Insights untuk memantau Database Aurora PostgreSQL Limitless. Ada juga fungsi dan tampilan statistik baru, dan acara tunggu, untuk Aurora PostgreSQL Limitless Database yang dapat Anda gunakan untuk pemantauan dan diagnostik.

Topik


- Memantau Database Tanpa Batas Aurora PostgreSQL dengan Amazon CloudWatch
- Memantau Database Tanpa Batas Aurora PostgreSQL dengan Wawasan Database CloudWatch
- Memantau Basis Data SQL Tanpa Batas Aurora Postgre dengan Log Amazon CloudWatch
- Memantau Basis Data SQL Tanpa Batas Aurora Postgre dengan Pemantauan yang Ditingkatkan
- Memantau Database Limitless PostgreSQL Aurora dengan Performance Insights
- Memantau Basis Data SQL Tanpa Batas Aurora Postgre dengan Perlindungan Amazon GuardDuty RDS
- Fungsi dan tampilan untuk Aurora SQL Postgre Limitless Database
- Tunggu acara untuk Aurora SQL Postgre Limitless Database

# Memantau Database Tanpa Batas Aurora PostgreSQL dengan Amazon CloudWatch

CloudWatch metrik untuk Aurora PostgreSQL Limitless Database dilaporkan dalam dimensi berikut:

- DBShardKelompok
- DBShardGroupRouterAggregation
- DBShardGroupInstance
- DBClusterPengenal

Untuk informasi selengkapnya tentang CloudWatch metrik, lihatMemantau Amazon Aurora dengan Amazon CloudWatch.

## DBShardMetrik grup

Untuk melihat **`DBShardGroup`** metrik Aurora PostgreSQL Limitless Database CloudWatch di konsol, pilih RDS, lalu pilih Grup. DBShard

Anda dapat melacak CloudWatch metrik berikut:

- `DBShardGroupACUUtilization`— Penggunaan unit kapasitas Aurora (ACU) sebagai persentase yang dihitung dari `DBShardGroupCapacity` dibagi dengan. `DBShardGroupMaxACU`
- `DBShardGroupCapacity`— Jumlah yang ACUs dikonsumsi oleh contoh penulis dari kelompok pecahan DB.
- `DBShardGroupComputeRedundancyCapacity`— Jumlah yang ACUs dikonsumsi oleh instance siaga grup shard DB.
- `DBShardGroupMaxACU`— Jumlah maksimum yang ACUs dikonfigurasi untuk grup pecahan DB.
- `DBShardGroupMinACU`— Jumlah minimum yang ACUs dibutuhkan oleh grup pecahan DB.

Kunci `DBShardGroupIdentifier` dimensi tersedia untuk menggabungkan metrik. `DBShardGroup`

## DBShardGroupRouterAggregation metrik

Untuk melihat **`DBShardGroupRouterAggregation`** metrik Aurora PostgreSQL Limitless Database CloudWatch di konsol, pilih RDS, lalu pilih. DBShardGroupRouterAggregation

Anda dapat melacak CloudWatch metrik berikut:

- `CommitThroughput`— Jumlah rata-rata operasi komit per detik di semua node router dalam grup shard DB.
- `DatabaseConnections`— Jumlah semua koneksi di semua node router dalam grup shard DB.

## DBShardGroupInstance metrik

A DBShardGroupInstanceadalah instance DB individu dalam setiap shard atau subcluster router.

Untuk melihat **`DBShardGroupInstance`** metrik Aurora PostgreSQL Limitless Database CloudWatch di konsol, pilih RDS, lalu pilih. DBShardGroupInstance

Anda dapat melacak CloudWatch metrik berikut:

- `ACUUtilization`— Persentase dihitung sebagai `ServerlessDatabaseCapacity` metrik dibagi dengan nilai ACU maksimum yang ditetapkan dari subcluster.
- `AuroraReplicaLag`- Untuk redundansi komputasi yang diaktifkan Limitless cluster, ini adalah jumlah lag saat mereplikasi pembaruan dari instance utama di subcluster.
- `AuroraReplicaLagMaximum`— Untuk redundansi komputasi yang diaktifkan Limitless cluster, ini adalah jumlah maksimum lag saat mereplikasi pembaruan dari instance utama di subcluster. Ketika replika baca dihapus atau diganti namanya, mungkin ada lonjakan sementara dalam kelambatan replikasi saat sumber daya lama didaur ulang. Gunakan metrik ini untuk mengetahui apakah failover terjadi karena kelambatan replikasi yang tinggi pada salah satu pembacanya.
- `AuroraReplicaLagMinimum`— Untuk redundansi komputasi yang diaktifkan Limitless cluster, ini adalah jumlah minimum lag saat mereplikasi pembaruan dari instance utama di subcluster.
- `BufferCacheHitRatio`— Persentase data dan indeks yang disajikan dari cache memori instance (sebagai lawan dari volume penyimpanan).
- `CommitLatency`— Durasi rata-rata untuk mesin dan penyimpanan untuk menyelesaikan operasi komit untuk node tertentu (router atau shard).
- `CommitThroughput`— Jumlah rata-rata operasi komit per detik.
- `CPUUtilization`— Penggunaan CPU sebagai persentase dari nilai ACU maksimum yang ditetapkan dari subcluster.
- `FreeableMemory`— Jumlah memori yang tidak terpakai yang tersedia ketika kelompok shard diskalakan ke kapasitas maksimumnya. Ini ditentukan oleh yang ditugaskan ACUs dari kelompok pecahan. Untuk setiap ACU kapasitas saat ini di bawah kapasitas maksimum, nilai ini meningkat sekitar 2 GiB. Dengan demikian, metrik ini tidak mendekati nol sampai kelompok pecahan DB diskalakan hingga batas maksimum.

- `MaximumUsedTransactionIDs`— Usia ID transaksi tertua yang tidak disedot, dalam transaksi. Jika nilai ini mencapai 2.146.483.648 (2^31 - 1.000.000), basis data akan dipaksa ke mode hanya baca untuk menghindari wraparound ID transaksi. Untuk informasi selengkapnya, lihat Preventing transaction ID wraparound failures dalam dokumentasi PostgreSQL.
- `NetworkReceiveThroughput`— Jumlah throughput jaringan yang diterima dari klien oleh setiap instance dalam grup shard DB. Throughput ini tidak termasuk lalu lintas jaringan antara instance dalam grup shard DB dan volume cluster.
- `NetworkThroughput`— Throughput jaringan agregat (baik ditransmisikan dan diterima) antara klien dan router, dan router dan pecahan dalam grup pecahan DB. Throughput ini tidak termasuk lalu lintas jaringan antara instance dalam grup shard DB dan volume cluster.
- `NetworkTransmitThroughput`— Jumlah throughput jaringan yang dikirim ke klien oleh setiap instance dalam grup shard DB. Throughput ini tidak termasuk lalu lintas jaringan antara instance dalam grup shard DB dan volume cluster.
- `ReadIOPS`— Jumlah rata-rata operasi input/output baca disk per detik (IOPS).
- `ReadLatency`— Jumlah rata-rata waktu yang dibutuhkan per disk readinput/output (I/O) operasi.
- `ReadThroughput`— Jumlah rata-rata byte yang dibaca dari disk per detik.
- `ServerlessDatabaseCapacity`- Kapasitas saat ini dari pecahan DB atau subcluster router dalam grup pecahan DB.
- `StorageNetworkReceiveThroughput`— Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instance dalam grup pecahan DB.
- `StorageNetworkThroughput`— Throughput jaringan agregat yang ditransmisikan ke dan diterima dari subsistem penyimpanan Aurora oleh setiap instance dalam grup pecahan DB.
- `StorageNetworkTransmitThroughput`— Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance dalam grup pecahan DB.
- `SwapUsage`— Jumlah ruang swap yang digunakan oleh grup shard DB.
- `TempStorageIOPS`— Jumlah rata-rata operasi I/O yang dilakukan pada penyimpanan lokal yang melekat pada instans DB. Ini mencakup operasi baca dan tulis I/O.

  `TempStorageIOPS`dapat digunakan `TempStorageThroughput` untuk mendiagnosis kasus yang jarang terjadi di mana aktivitas jaringan untuk transfer antara instans DB Anda dan perangkat penyimpanan lokal bertanggung jawab atas peningkatan kapasitas yang tidak terduga.
- `TempStorageThroughput`— Jumlah data yang ditransfer ke dan dari penyimpanan lokal yang terkait dengan router atau pecahan.
- `WriteIOPS`— Jumlah rata-rata disk menulis IOPS.

- `WriteLatency`— Jumlah rata-rata waktu yang dibutuhkan per disk menulis operasi I/O.
- `WriteThroughput`— Jumlah rata-rata byte yang ditulis ke disk per detik.

Kunci dimensi berikut tersedia untuk menggabungkan metrik: `DBShardGroupInstance`

- `DBClusterIdentifier`— Cluster DB PostgreSQL Aurora.
- `DBShardGroupIdentifier`— Grup pecahan DB tempat instance tersebut berada.
- `DBShardGroupSubClusterType`— Tipe node, baik `Distributed Transaction Router` (router) atau `Data Access Shard` (shard).
- `DBShardGroupSubClusterIdentifier`— Nama router atau pecahan tempat instance tersebut berada.

Berikut ini adalah contoh metrik agregasi CloudWatch :

- Total `CPUUtilization` semua instance yang termasuk dalam shard atau router tertentu dalam grup shard DB.
- Total `CPUUtilization` semua instance dalam grup pecahan DB.

## DBClusterMetrik pengenal

Untuk melihat **`DBClusterIdentifier`** metrik Aurora PostgreSQL Limitless Database CloudWatch di konsol, pilih RDS, lalu pilih Identifier. DBCluster

Bila Anda menggunakan Aurora PostgreSQL Limitless Database, Anda mungkin memiliki lebih banyak) input/output (I/O operasi daripada yang Anda lakukan untuk cluster Aurora DB. Anda dapat melacak CloudWatch metrik berikut untuk klaster Limitless Database Anda:

- `VolumeReadIops`— Jumlah operasi I/O baca yang ditagih dari volume cluster, dilaporkan pada interval 5 menit.
- `VolumeWriteIops`— Jumlah operasi I/O disk tulis ke volume cluster, dilaporkan pada interval 5 menit.

Aurora PostgreSQL Limitless Database menggunakan konfigurasi penyimpanan cluster. Aurora I/O-Optimized DenganAurora I/O-Optimized, Anda membayar satu harga bulanan untuk semua operasi I/O, daripada membayar untuk setiap satu juta permintaan I/O. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

Anda mungkin juga menggunakan lebih banyak penyimpanan daripada yang Anda lakukan untuk cluster Aurora DB. Anda dapat melacak CloudWatch metrik berikut untuk penyimpanan:

- `BackupRetentionPeriodStorageUsed`— Total penggunaan penyimpanan cadangan berkelanjutan yang ditagih dari cluster Database Aurora PostgreSQL Limitless Anda.
- `SnapshotStorageUsed`— Total penggunaan penyimpanan snapshot yang ditagih dari cluster Database Aurora PostgreSQL Limitless Anda.
- `TotalBackupStorageBilled`— Jumlah biaya Anda untuk retensi cadangan otomatis dan snapshot cluster DB.

  Untuk informasi selengkapnya tentang biaya penyimpanan cadangan, lihatMemahami penggunaan penyimpanan cadangan Amazon Aurora.
- `VolumeBytesUsed`— Jumlah penyimpanan yang digunakan oleh cluster Database Aurora PostgreSQL Limitless Anda, dilaporkan pada interval 5 menit.

# Memantau Database Tanpa Batas Aurora PostgreSQL dengan Wawasan Database CloudWatch

Mode Standar Database Insights diperlukan sebagai bagian dari mengaktifkan Aurora PostgreSQL Limitless Database. Anda dapat menggunakannya untuk memantau beban database (DB Load) dari instans DB Limitless Database Anda secara real time. DB Load mengukur tingkat aktivitas sesi dalam database. Anda dapat menggunakan Database Insights untuk menganalisis dan memecahkan masalah kinerja instans DB Database PostgreSQL Limitless Aurora Anda dalam skala besar.

Untuk informasi selengkapnya tentang CloudWatch Database Insights, lihat berikut ini.

- Memantau database Amazon Aurora dengan Database CloudWatch Insights
- CloudWatch Wawasan Database di  CloudWatch Panduan Pengguna Amazon
- Memulai Wawasan CloudWatch Database di  CloudWatch Panduan Pengguna Amazon
-

Untuk informasi tentang mengaktifkan mode Lanjutan atau Mode Standar Wawasan Database, lihat topik berikut.

Topik

- Mengaktifkan mode Advanced Database Insights untuk Aurora PostgreSQL Limitless Database
- Mengaktifkan mode Standar Wawasan Database untuk Aurora PostgreSQL Limitless Database

## Mengaktifkan mode Advanced Database Insights untuk Aurora PostgreSQL Limitless Database

Untuk mengaktifkan mode Advanced Database Insights untuk Aurora PostgreSQL Limitless Database Anda, gunakan prosedur berikut.

Mengaktifkan mode Advanced Database Insights saat membuat cluster DB untuk Aurora PostgreSQL Limitless Database

Aktifkan mode Advanced Database Insights saat membuat database untuk Aurora PostgreSQL Limitless Database.

Console

Di konsol, Anda dapat mengaktifkan mode Advanced Database Insights saat membuat cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Advanced Database Insights saat membuat cluster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih Buat basis data.
4. Di bagian Database Insights, pilih Mode lanjutan. Kemudian, pilih opsi berikut:
   - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Periode retensi harus 15-24 bulan untuk mode Advanced Database Insights.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Buat basis data.

AWS CLI

Untuk mengaktifkan mode Advanced Database Insights saat membuat cluster DB, panggil create-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--db-cluster-identifier`— Pengidentifikasi untuk cluster DB.
- `--database-insights-mode advanced`untuk mengaktifkan mode Advanced Database Insights.
- `--engine`- Cluster DB harus menggunakan mesin `aurora-postgresql` DB.
- `--engine-version`- Cluster DB harus menggunakan salah satu versi mesin DB:
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`- Cluster DB harus menggunakan konfigurasi penyimpanan cluster `aurora-iopt1` DB.

- `--cluster-scalability-type`— Menentukan mode skalabilitas cluster Aurora DB. Ketika diatur `kelimitless`, cluster beroperasi sebagai Database Aurora PostgreSQL Limitless. Ketika diatur ke `standard` (default), cluster menggunakan pembuatan instans DB normal.

  > **Note**
  >
  > Anda tidak dapat mengubah pengaturan ini setelah Anda membuat cluster DB.

- `--master-username`— Nama pengguna master untuk cluster DB.
- `--master-user-password`— Kata sandi untuk pengguna utama.
- `--enable-performance-insights`untuk mengaktifkan Performance Insights untuk Database Insights.
- `--performance-insights-retention-period`— Periode retensi untuk data untuk cluster DB Anda. Untuk mengaktifkan Database Insights, periode retensi harus minimal 465 hari.
- `--monitoring-interval`— Interval, dalam hitungan detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan untuk cluster DB. Nilai ini tidak bisa`0`.
- `--monitoring-role-arn`— Nama Sumber Daya Amazon (ARN) untuk peran IAM yang memungkinkan RDS mengirim metrik Pemantauan yang Ditingkatkan ke Log Amazon. CloudWatch
- `--enable-cloudwatch-logs-exports`— Anda harus mengekspor `postgresql` log ke CloudWatch Log.

Contoh berikut memungkinkan mode Advanced Database Insights saat membuat cluster DB.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode advanced \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 465 \
--monitoring-interval 5 \
```

```
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

Untuk Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode advanced ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 465 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

Untuk mengaktifkan mode Advanced Database Insights saat Anda membuat , tentukan parameter berikut untuk operasi Amazon RDS API.

- `DatabaseInsightsMode` untuk `advanced`
- `Engine` untuk `aurora-postgresql`
- `EngineVersion`ke versi mesin yang tersedia untuk Limitless Database
- `StorageType` untuk `aurora-iopt1`
- `ClusterScalabilityType` untuk `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` untuk `True`
- `PerformanceInsightsRetentionPeriod`setidaknya 465 berhari-hari
- `MonitoringInterval`Untuk nilai yang tidak `0`
- `MonitoringRoleArn`ke Nama Sumber Daya Amazon (ARN) untuk peran IAM yang memungkinkan RDS mengirim metrik Pemantauan yang Ditingkatkan ke Log Amazon CloudWatch

### Mengaktifkan mode Advanced Database Insights saat memodifikasi cluster DB untuk Aurora PostgreSQL Limitless Database

Aktifkan Database Insights saat memodifikasi database untuk Aurora PostgreSQL Limitless Database.

> **Note**
>
> Untuk mengaktifkan Database Insights, setiap instans DB di klaster AdB harus memiliki pengaturan Performance Insights dan Enhanced Monitoring yang sama.

Console

Di konsol, Anda dapat mengaktifkan mode Advanced Database Insights saat Anda memodifikasi cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Advanced Database Insights saat memodifikasi cluster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih cluster DB, dan pilih Modify.
4. Di bagian Database Insights, pilih Mode lanjutan. Kemudian, pilih opsi berikut:
   - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Periode retensi harus 15-24 bulan untuk mode Advanced Database Insights.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih Langsung terapkan. Jika Anda memilih Terapkan selama jendela pemeliharaan terjadwal berikutnya, database Anda akan mengabaikan pengaturan ini dan segera mengaktifkan mode Advanced Database Insights.

7. Pilih Ubah klaster.

AWS CLI

Untuk mengaktifkan mode Advanced Database Insights saat memodifikasi cluster DB, panggil modify-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--database-insights-mode advanced`untuk mengaktifkan mode Advanced Database Insights.
- `--db-cluster-identifier`— Pengidentifikasi untuk cluster DB.
- `--enable-performance-insights`untuk mengaktifkan Performance Insights untuk Database Insights.
- `--performance-insights-retention-period`— Periode retensi untuk data untuk Anda. Untuk mengaktifkan mode Advanced Database Insights, periode retensi harus minimal 465 hari.

Contoh berikut memungkinkan mode Advanced Database Insights saat memodifikasi cluster DB.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

Untuk mengaktifkan mode Advanced Database Insights saat Anda memodifikasi kluster DB, tentukan parameter berikut untuk operasi Modify DBCluster Amazon RDS API.

- `DatabaseInsightsMode` untuk `advanced`

- `EnablePerformanceInsights` untuk `True`
- `PerformanceInsightsRetentionPeriod`setidaknya `465` berhari-hari

## Mengaktifkan mode Standar Wawasan Database untuk Aurora PostgreSQL Limitless Database

Untuk mengaktifkan mode Standar Wawasan Database untuk Database Aurora PostgreSQL Limitless Anda, gunakan prosedur berikut.

Mengaktifkan mode Standar Database Insights saat membuat cluster DB untuk Aurora PostgreSQL Limitless Database

Aktifkan mode Standar Database Insights saat membuat database untuk Aurora PostgreSQL Limitless Database.

Console

Di konsol, Anda dapat mengaktifkan mode Standar Database Insights saat membuat cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Standar Database Insights saat membuat cluster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih Buat basis data.
4. Di bagian Database Insights, pilih Mode standar. Kemudian, pilih opsi berikut:
   - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Untuk membuat cluster DB untuk Aurora PostgreSQL Limitless Database, periode retensi harus minimal 31 hari.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Buat basis data.

AWS CLI

Untuk mengaktifkan mode Standar Database Insights saat membuat cluster DB, panggil create-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--db-cluster-identifier`— Pengidentifikasi untuk cluster DB.
- `--database-insights-mode standard`untuk mengaktifkan mode Standar Wawasan Database.
- `--engine`- Cluster DB harus menggunakan mesin `aurora-postgresql` DB.
- `--engine-version`- Cluster DB harus menggunakan salah satu versi mesin DB:
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`- Cluster DB harus menggunakan konfigurasi penyimpanan cluster `aurora-iopt1` DB.
- `--cluster-scalability-type`— Menentukan mode skalabilitas cluster Aurora DB. Ketika diatur `kelimitless`, cluster beroperasi sebagai Database Aurora PostgreSQL Limitless. Ketika diatur ke `standard` (default), cluster menggunakan pembuatan instans DB normal.

  > **Note**
  >
  > Anda tidak dapat mengubah pengaturan ini setelah Anda membuat cluster DB.

- `--master-username`— Nama pengguna master untuk cluster DB.
- `--master-user-password`— Kata sandi untuk pengguna utama.
- `--enable-performance-insights`untuk mengaktifkan Performance Insights untuk Database Insights.
- `--performance-insights-retention-period`— Periode retensi untuk data untuk cluster DB Anda. Untuk membuat cluster DB untuk Aurora PostgreSQL Limitless Database, periode retensi harus minimal 31 hari.
- `--monitoring-interval`— Interval, dalam hitungan detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan untuk cluster DB. Nilai ini tidak bisa`0`.
- `--monitoring-role-arn`— Nama Sumber Daya Amazon (ARN) untuk peran IAM yang memungkinkan RDS mengirim metrik Pemantauan yang Ditingkatkan ke Log Amazon. CloudWatch

- `--enable-cloudwatch-logs-exports`— Anda harus mengekspor `postgresql` log ke CloudWatch Log.

Contoh berikut memungkinkan mode Standar Database Insights saat membuat cluster DB.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode standard \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 31 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

Untuk Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode standard ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 31 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

Untuk mengaktifkan mode Standar Database Insights saat Anda membuat , tentukan parameter berikut untuk operasi Amazon RDS API Anda.

- `DatabaseInsightsMode` untuk `standard`
- `Engine` untuk `aurora-postgresql`
- `EngineVersion`ke versi mesin yang tersedia untuk Limitless Database
- `StorageType` untuk `aurora-iopt1`
- `ClusterScalabilityType` untuk `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` untuk `True`
- `PerformanceInsightsRetentionPeriod`setidaknya `31` berhari-hari
- `MonitoringInterval`Untuk nilai yang tidak `0`
- `MonitoringRoleArn`ke Nama Sumber Daya Amazon (ARN) untuk peran IAM yang memungkinkan RDS mengirim metrik Pemantauan yang Ditingkatkan ke Log Amazon CloudWatch

Mengaktifkan mode Standar Database Insights saat memodifikasi cluster DB untuk Aurora PostgreSQL Limitless Database

Aktifkan Database Insights saat memodifikasi database untuk Aurora PostgreSQL Limitless Database.

> **Note**
>
> Untuk mengaktifkan Database Insights, setiap instans DB di cluster DB harus memiliki pengaturan Performance Insights dan Enhanced Monitoring yang sama.

Console

Di konsol, Anda dapat mengaktifkan mode Standar Database Insights saat membuat cluster DB. Pengaturan untuk Database Insights berlaku untuk semua instans DB di cluster DB Anda.

Untuk mengaktifkan mode Standar Database Insights saat memodifikasi cluster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih cluster DB, dan pilih Modify.
4. Di bagian Database Insights, pilih Mode standar. Kemudian, pilih opsi berikut:
   - Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Untuk membuat cluster DB untuk Aurora PostgreSQL Limitless Database, periode retensi harus minimal 31 hari.
   - AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih Langsung terapkan. Jika Anda memilih Terapkan selama jendela pemeliharaan terjadwal berikutnya, database Anda akan mengabaikan pengaturan ini dan segera mengaktifkan mode Standar Database Insights.
7. Pilih Ubah klaster.

AWS CLI

Untuk mengaktifkan mode Standar Database Insights saat memodifikasi cluster DB, panggil modify-db-cluster AWS CLI perintah dan berikan nilai berikut:

- `--db-cluster-identifier`— Pengidentifikasi untuk cluster DB.
- `--database-insights-mode standard`untuk mengaktifkan mode Standar Wawasan Database.
- `--enable-performance-insights`untuk mengaktifkan Performance Insights untuk Database Insights.
- `--performance-insights-retention-period`— Periode retensi untuk data untuk cluster DB Anda. Untuk mengaktifkan mode Standar Wawasan Database, periode retensi harus minimal 31 hari.

Contoh berikut memungkinkan mode Standar Database Insights saat memodifikasi cluster DB.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 31
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 31
```

RDS API

Untuk mengaktifkan mode Standar Database Insights saat Anda memodifikasi kluster DB, tentukan parameter berikut untuk operasi Modify DBCluster Amazon RDS API.

- `DatabaseInsightsMode` untuk `standard`
- `EnablePerformanceInsights` untuk `True`
- `PerformanceInsightsRetentionPeriod`setidaknya `31` berhari-hari

# Memantau Basis Data SQL Tanpa Batas Aurora Postgre dengan Log Amazon CloudWatch

Mengekspor log Postgre ke SQL CloudWatch Log diperlukan sebagai bagian dari mengaktifkan Basis Data Tanpa Batas Aurora Postgre. SQL Anda dapat mengakses dan menganalisis log ini di Wawasan Log, mirip dengan mengakses CloudWatch log Postgre untuk klaster DB Aurora Postgre SQL standar. SQL Untuk informasi selengkapnya, lihat Menganalisis log PostgreSQL menggunakan Wawasan Log CloudWatch .

Nama grup log untuk cluster DB sama seperti di Aurora Postgre: SQL

```
/aws/rds/cluster/DB_cluster_ID/postgresql
```

Nama grup log untuk grup pecahan DB mengambil bentuk berikut:

```
/aws/rds/cluster/DB_cluster_ID/DB_shard_group_ID/postgresql
```

Ada aliran log untuk setiap node (router atau shard). Nama mereka memiliki bentuk berikut:

```
[DistributedTransactionRouter|
DataAccessShard]/node_cluster_serial_ID-node_instance_serial_ID/n
```

Sebagai contoh:

- Router — `DistributedTransactionRouter/6-6.2`
- Pecahan — `DataAccessShard/22-22.0`

> **Note**
>
> Anda tidak dapat melihat file SQL log Postgre untuk grup pecahan DB langsung di RDS konsol AWS CLI, atau RDS API seperti yang Anda bisa untuk cluster DB. Anda harus menggunakan Wawasan CloudWatch Log untuk melihatnya.

# Memantau Basis Data SQL Tanpa Batas Aurora Postgre dengan Pemantauan yang Ditingkatkan

Pemantauan yang Disempurnakan diperlukan sebagai bagian dari mengaktifkan Basis Data Tanpa Batas Aurora SQL Postgre. Anda dapat menggunakannya untuk memantau sistem operasi instans DB Limitless Database Anda secara real time.

Aurora menerbitkan metrik Pemantauan yang Ditingkatkan di Log. CloudWatch Beberapa metrik utama yang tersedia termasuk koneksi database, penggunaan penyimpanan, dan latensi kueri. Ini dapat membantu mengidentifikasi kemacetan kinerja.

Untuk informasi selengkapnya tentang metrik Pemantauan yang Ditingkatkan, lihatMetrik OS untuk Aurora.

# Memantau Database Limitless PostgreSQL Aurora dengan Performance Insights

Gunakan Performance Insights untuk memantau klaster Database Aurora PostgreSQL Limitless Anda. Performance Insights bekerja sama untuk Aurora PostgreSQL Limitless Database seperti halnya untuk cluster DB Aurora standar. Namun, Anda melacak metrik di tingkat grup shard untuk Aurora PostgreSQL Limitless Database.

Dua metrik Performance Insights utama yang harus dilacak adalah sebagai berikut:

- Beban basis data - Mengukur tingkat aktivitas dalam database Anda. Metrik utama dalam Wawasan Performa adalah `DBLoad`, yang dikumpulkan setiap detik.

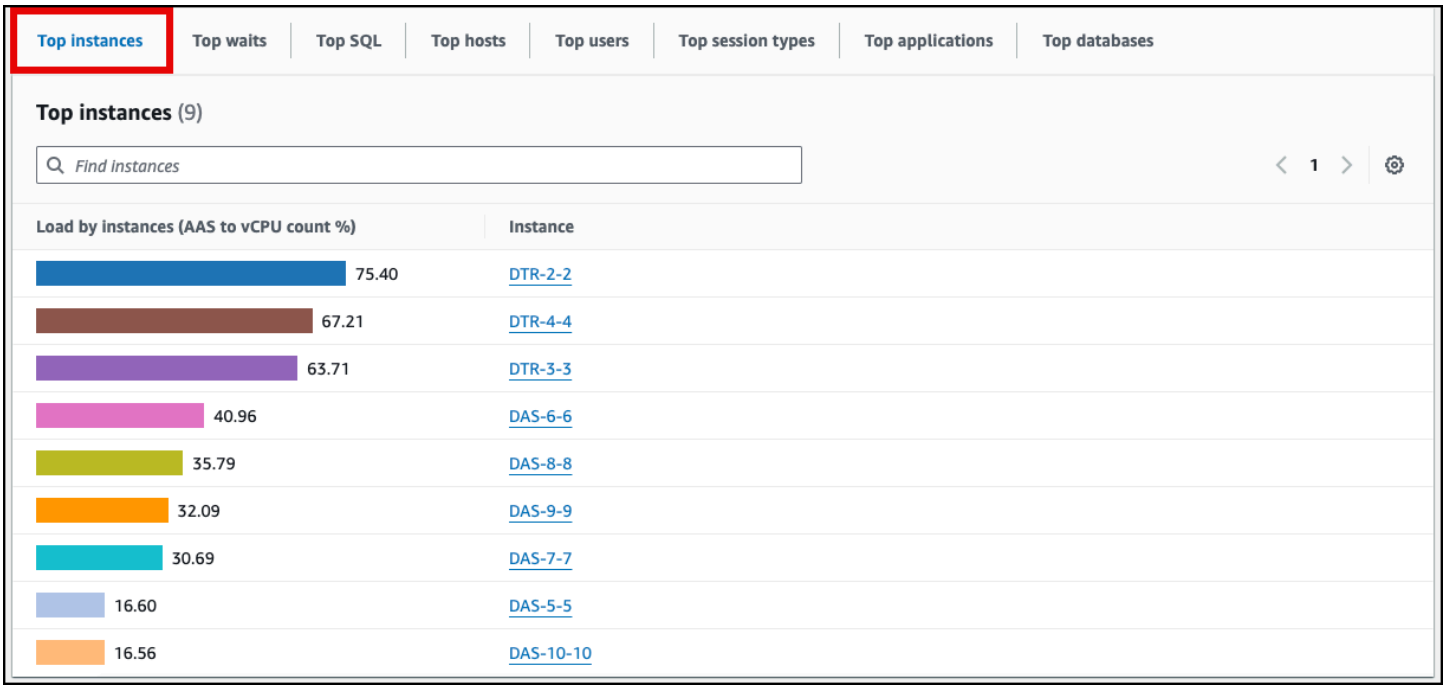
  Unit untuk `DBLoad` metrik dalam Performance Insights adalah sesi aktif rata-rata (AAS). Untuk mendapatkan rata-rata sesi aktif, Performance Insights mengambil sampel jumlah sesi yang menjalankan kueri secara bersamaan. AAS adalah jumlah total sesi dibagi dengan jumlah total sampel untuk periode waktu tertentu. Untuk informasi lebih lanjut tentang `DBLoad` dan AAS, lihatMuatan basis data.
- CPU Maksimum — Daya komputasi maksimum yang tersedia untuk database Anda. Untuk melihat apakah sesi aktif melebihi CPU maksimum, lihat hubungannya dengan `Max vCPU` garis. `Max vCPU`Nilai ditentukan oleh jumlah inti vCPU (CPU virtual) untuk instans DB Anda. Untuk informasi selengkapnya tentang `Max vCPU`, lihat Maksimum CPU.

Selain itu, Anda dapat “mengiris” `DBLoad` metrik ke dalam dimensi, yang merupakan subkategori dari metrik. Dimensi yang paling berguna adalah sebagai berikut:

- Instans teratas - Menunjukkan beban DB relatif untuk instans Anda (pecahan dan router) dalam urutan menurun.
- Tunggu peristiwa - Menyebabkan pernyataan SQL menunggu peristiwa tertentu terjadi sebelum mereka dapat terus berjalan. Acara tunggu menunjukkan di mana pekerjaan terhambat.
- SQL Teratas - Menunjukkan kueri mana yang paling berkontribusi pada pemuatan DB.

Untuk informasi selengkapnya tentang dimensi Performance Insights, lihat. Dimensi

Gambar berikut menunjukkan dimensi contoh Top untuk grup pecahan DB.




Topik

- Menganalisis beban DB untuk Aurora Postgre SQL Limitless Database menggunakan dasbor Performance Insights

## Menganalisis beban DB untuk Aurora Postgre SQL Limitless Database menggunakan dasbor Performance Insights

Dengan Performance Insights, Anda dapat melacak metrik di tingkat grup shard dan di tingkat instans untuk Database Tanpa Batas Aurora Postgre. SQL Saat menganalisis beban DB untuk Database SQL Tanpa Batas Aurora Postgre, Anda mungkin ingin membandingkan beban DB untuk setiap pecahan dan router dengan maksimum v. CPU

> **Note**
>
> Aurora Postgre SQL Limitless Database selalu mengaktifkan Performance Insights dan Enhanced Monitoring. Periode retensi minimum untuk data Performance Insights untuk Limitless Database adalah 31 hari (1 bulan).

Tampilan Absolut menunjukkan jumlah sesi aktif Rata-rata (AAS) dan perkiraan vCPU. Tampilan Relatif menunjukkan rasio AAS terhadap estimasi vCPU.

Topik

- Menganalisis beban DB relatif untuk Aurora Postgre SQL Limitless Database menggunakan dasbor Performance Insights
- Menganalisis beban DB dengan menunggu Database Aurora SQL Postgre Limitless menggunakan dasbor Performance Insights
- Menganalisis distribusi beban untuk Aurora Postgre SQL Limitless Database menggunakan dasbor Performance Insights

### Menganalisis beban DB relatif untuk Aurora Postgre SQL Limitless Database menggunakan dasbor Performance Insights

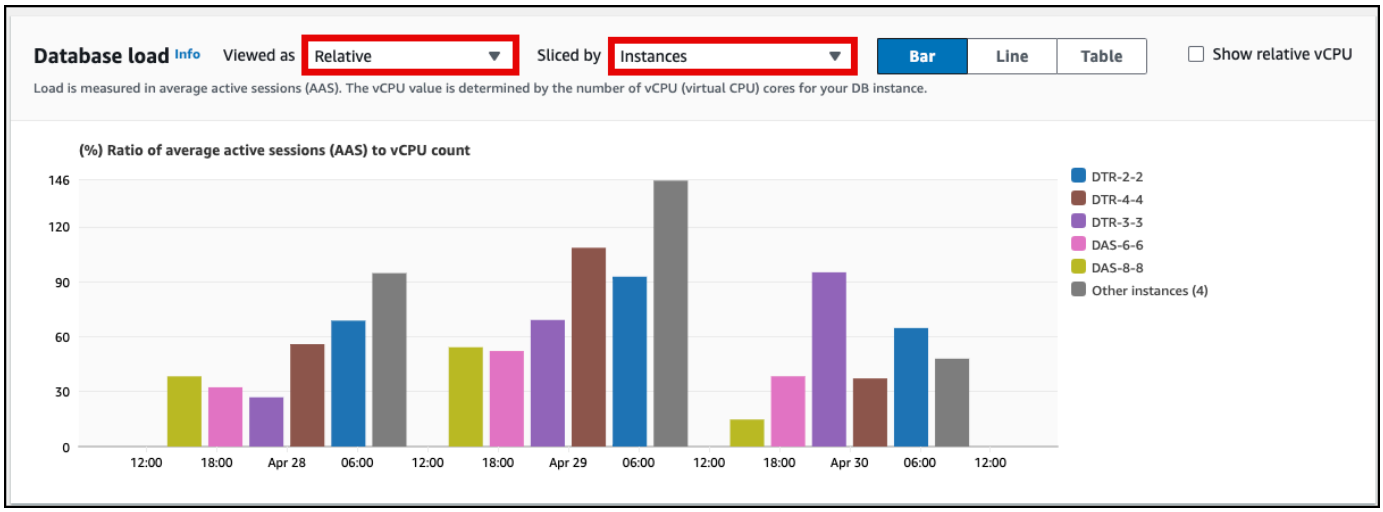
Anda mungkin ingin meningkatkan kinerja Database SQL Tanpa Batas Aurora Postgre Anda dengan melacak beban DB relatif. Untuk menganalisis beban DB relatif dengan contoh untuk Database SQL Tanpa Batas Aurora Postgre Anda, gunakan prosedur berikut.

Untuk menganalisis beban DB relatif untuk Aurora Postgre SQL Limitless Database menggunakan konsol

1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

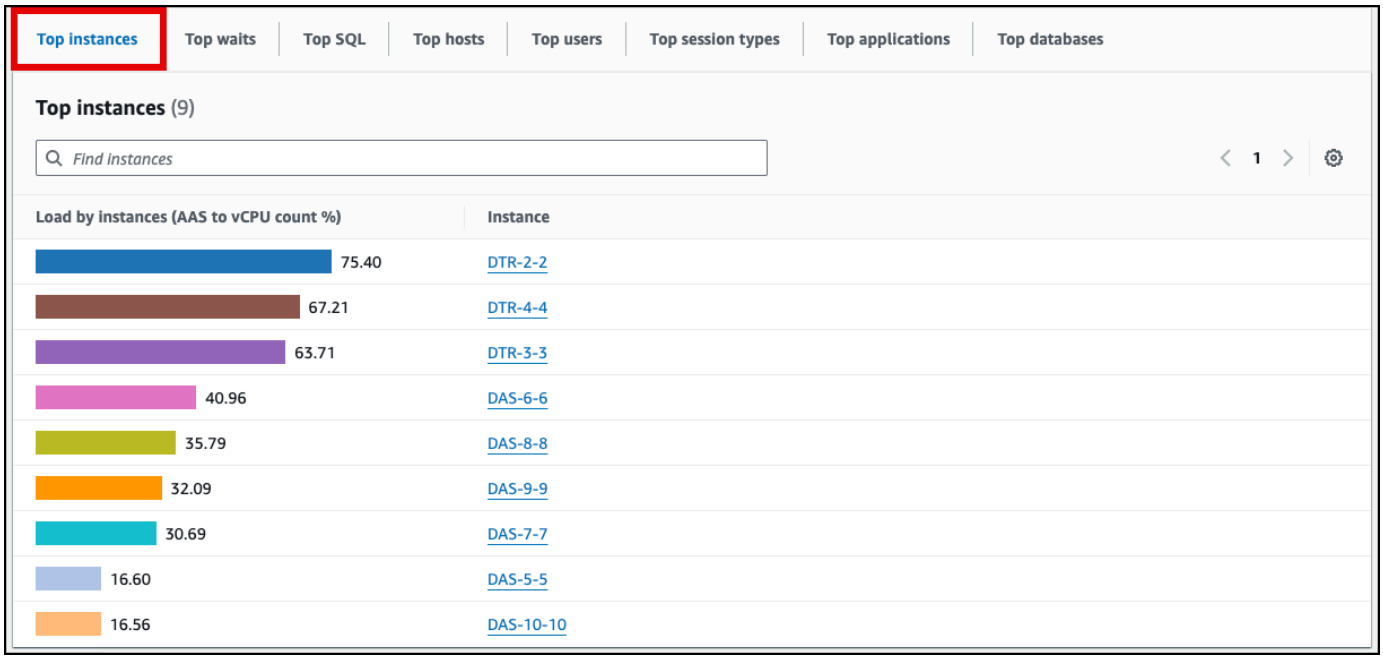
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih Database Tanpa Batas Aurora PostgreSQL. Dasbor Performance Insights ditampilkan untuk Database Aurora Postgre Limitless tersebut. SQL
4. Di bagian Pemuatan basis data (beban DB), pilih Instans untuk Diiris oleh. Untuk melihat rasio Average active session (AAS) ke v CPU core untuk semua instance di Aurora SQL Postgre Limitless Database Anda, pilih Relatif untuk Dilihat sebagai.

   Bagan sesi aktif rata-rata menunjukkan beban DB untuk instans di Database Tanpa Batas Aurora SQL Postgre Anda.




5. Untuk melihat instance teratas, pilih tab Instance teratas.

   Dalam contoh berikut, contoh dengan beban DB tertinggi adalahDTR-2-2.



6. (Opsional) Untuk menganalisis beban DB untuk sebuah instans di Database SQL Tanpa Batas Aurora Postgre Anda, pilih nama instance di kolom Instans. Untuk melihat beban DB`DTR-2-2`, pilih `DTR-2-2` di kolom Instances.

> **Note**
>
> Anda dapat melihat metrik Performance Insights hanya untuk instance di Database Tanpa Batas Aurora Postgre. SQL

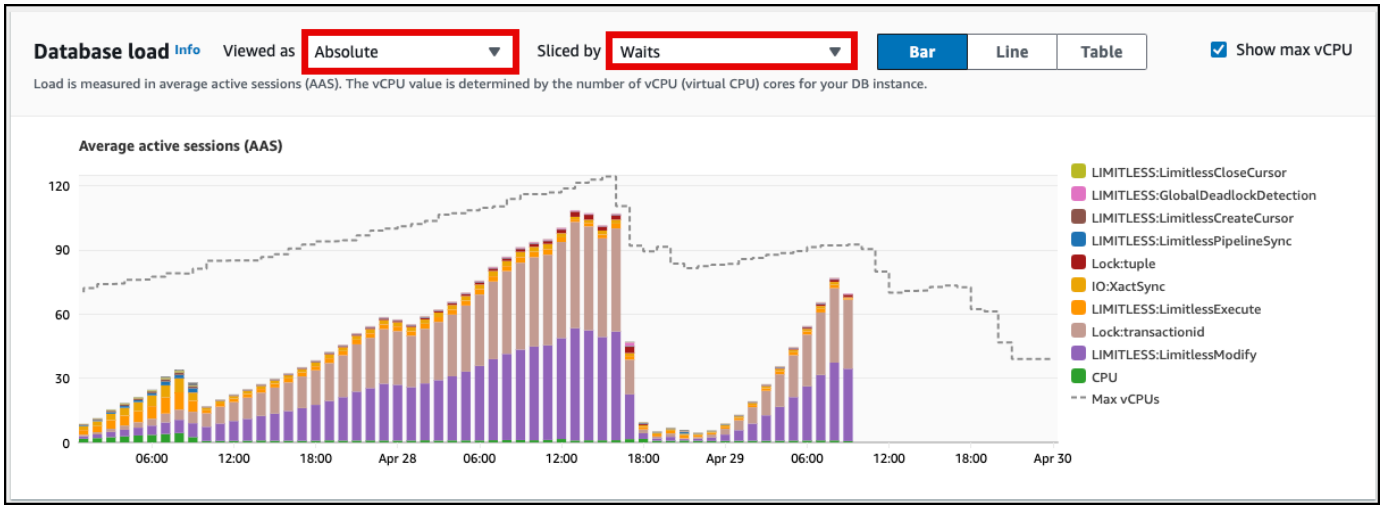
Menganalisis beban DB dengan menunggu Database Aurora SQL Postgre Limitless menggunakan dasbor Performance Insights

Anda mungkin ingin meningkatkan kinerja untuk Basis Data SQL Tanpa Batas Aurora Postgre Anda dengan melacak peristiwa tunggu. Untuk menganalisis beban DB dengan menunggu peristiwa untuk Database Aurora Postgre SQL Limitless Anda, gunakan prosedur berikut.

Untuk menganalisis beban DB dengan menunggu Aurora SQL Postgre Limitless Database menggunakan konsol

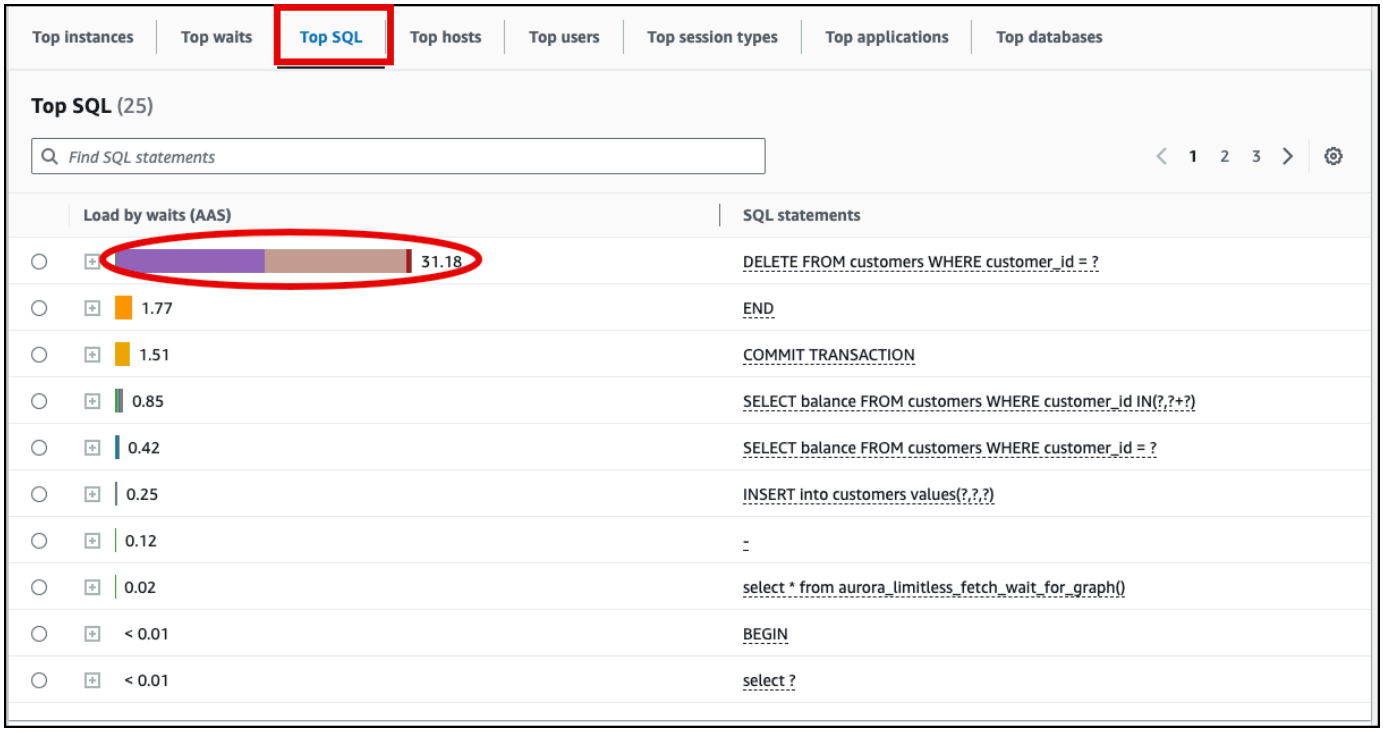
1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih Database Tanpa Batas Aurora PostgreSQL. Dasbor Performance Insights ditampilkan untuk Database Aurora Postgre Limitless tersebut. SQL
4. Di bagian beban basis data (beban DB), pilih Waits for Sliced by. Untuk melihat jumlah AAS dan estimasi vCPU, pilih Absolute untuk Dilihat sebagai.

   Bagan sesi aktif rata-rata menunjukkan beban DB untuk instans di Database Tanpa Batas Aurora SQL Postgre Anda.



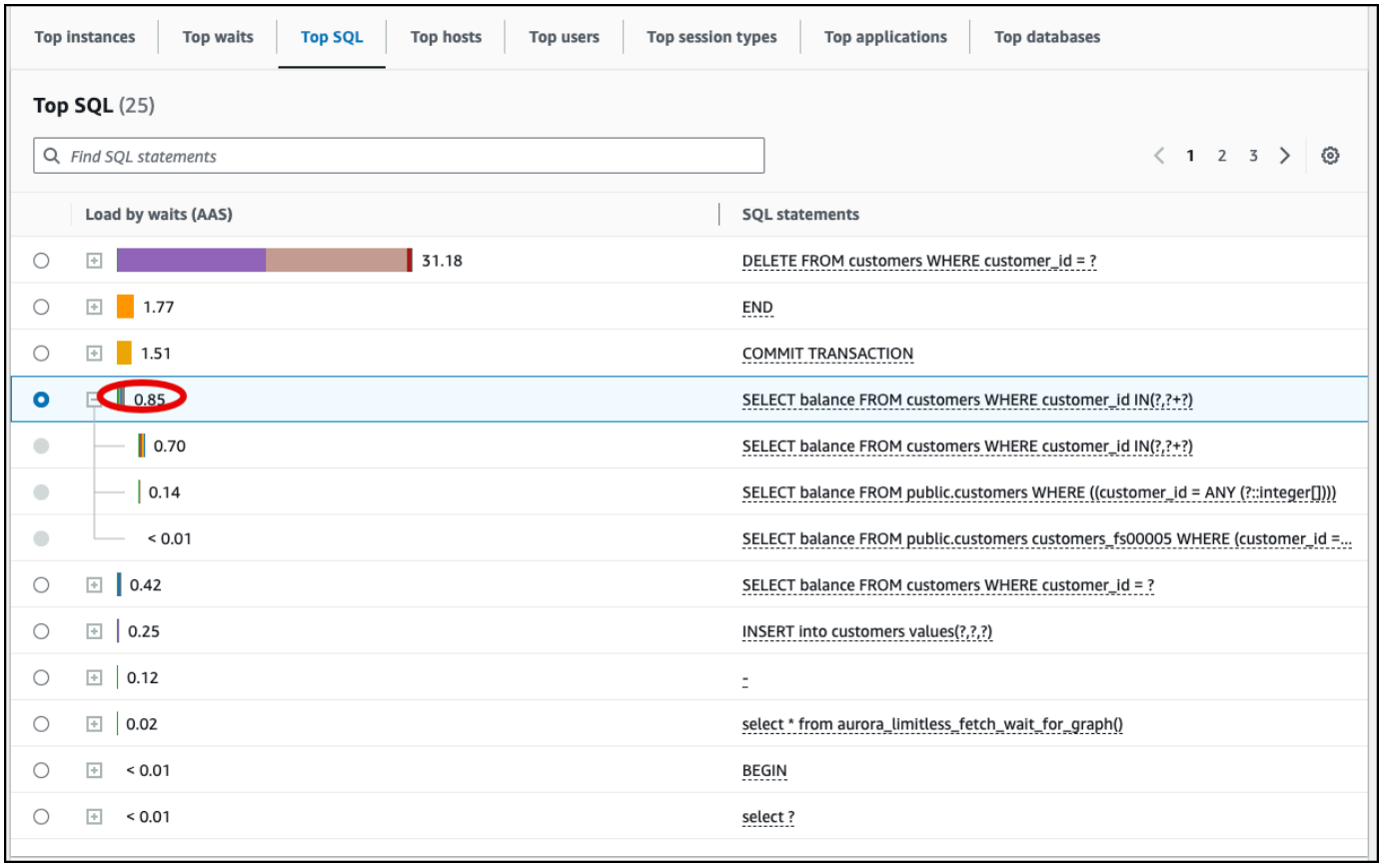

5. Gulir ke bawah ke SQL tab Atas.

   Dalam contoh berikut, SQL pernyataan dengan beban tertinggi dengan menunggu adalah `DELETE` pernyataan.




6. Pilih SQL pernyataan untuk mengembangkannya ke dalam pernyataan komponennya.

   Dalam contoh berikut, `SELECT` pernyataan tersebut memiliki 3 pernyataan komponen.



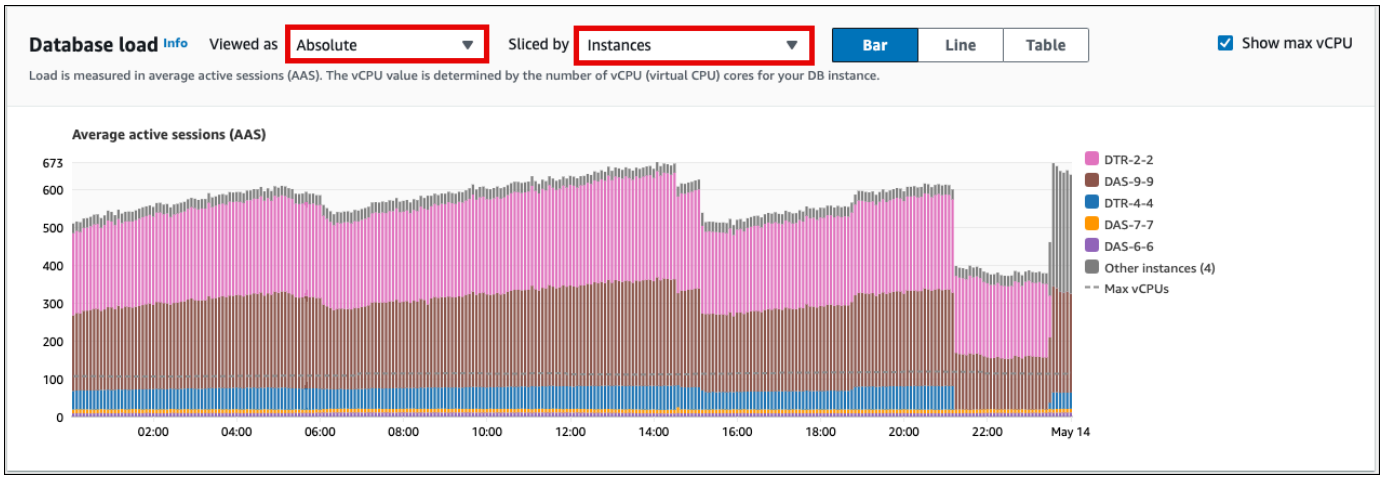

Menganalisis distribusi beban untuk Aurora Postgre SQL Limitless Database menggunakan dasbor Performance Insights

Anda mungkin ingin menyeimbangkan distribusi beban untuk instance di Database Aurora SQL Postgre Limitless Anda. Untuk menganalisis distribusi beban instans pada Database Aurora SQL Postgre Limitless, gunakan prosedur berikut.

Untuk menganalisis distribusi beban instance pada Database Aurora SQL Postgre Limitless menggunakan konsol

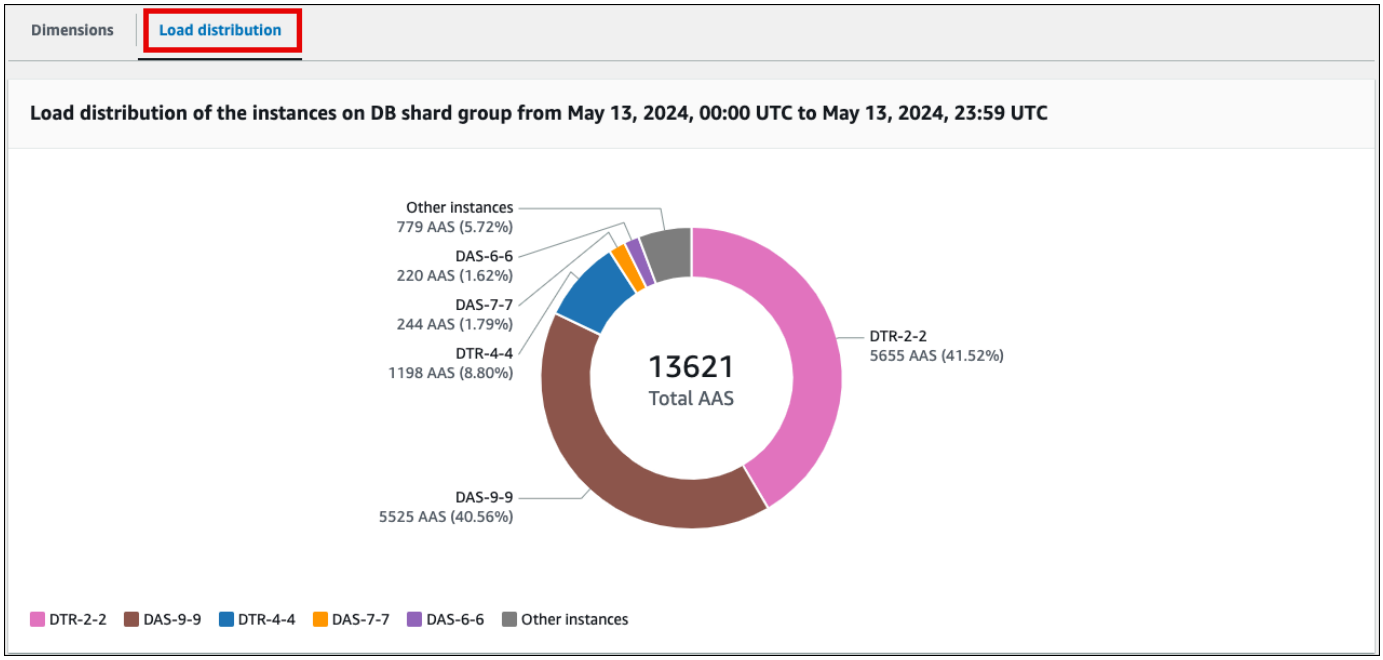
1. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih Database Tanpa Batas Aurora PostgreSQL. Dasbor Performance Insights ditampilkan untuk Database Aurora Postgre Limitless tersebut. SQL
4. Di bagian Pemuatan basis data (beban DB), pilih Instance for Sliced by. Untuk melihat jumlah AAS dan estimasi v CPU untuk semua instans di Basis Data Tanpa Batas Aurora SQL Postgre Anda, pilih Absolute untuk Dilihat sebagai.

Bagan sesi aktif rata-rata menunjukkan beban DB untuk instans di Database Tanpa Batas Aurora SQL Postgre Anda.




5. Untuk melihat bagan distribusi beban instans di Database Aurora SQL Postgre Limitless Anda, pilih tab distribusi Muat.

   Dalam contoh berikut, contoh dengan beban DB tertinggi adalah`DTR-2-2`.



# Memantau Basis Data SQL Tanpa Batas Aurora Postgre dengan Perlindungan Amazon GuardDuty RDS

Amazon GuardDuty adalah layanan deteksi ancaman yang membantu melindungi akun, wadah, beban kerja, dan data di AWS lingkungan Anda. Menggunakan model machine learning (ML), serta kemampuan deteksi anomali dan ancaman, GuardDuty terus memantau berbagai sumber log dan aktivitas runtime untuk mengidentifikasi dan memprioritaskan potensi risiko keamanan dan aktivitas berbahaya di lingkungan Anda.

GuardDuty RDSProtection menganalisis dan memprofilkan peristiwa login untuk potensi ancaman akses ke database Amazon Aurora Anda. Saat Anda mengaktifkan RDS Perlindungan, akan GuardDuty menggunakan peristiwa RDS login dari basis data Aurora Anda. RDS Perlindungan memantau peristiwa ini dan memprofilkannya untuk potensi ancaman orang dalam atau aktor eksternal.

Untuk informasi lebih lanjut tentang GuardDuty RDS Perlindungan di Aurora, lihat. Memantau ancaman dengan Perlindungan Amazon GuardDuty RDS untuk Amazon Aurora

Untuk informasi selengkapnya tentang mengaktifkan GuardDuty RDS Perlindungan, lihat GuardDuty RDSPerlindungan di Panduan GuardDuty Pengguna Amazon.

# Fungsi dan tampilan untuk Aurora SQL Postgre Limitless Database

Aurora Postgre SQL Limitless Database telah menambahkan fungsi dan tampilan. Mereka didasarkan pada fungsi dan tampilan Aurora Postgre SQL yang sesuai.

> **Note**
>
> Beberapa statistik dapat mengembalikan hasil yang tidak konsisten jika ada transaksi yang sedang berlangsung.

Topik


- Fungsi Basis Data Tanpa Batas Aurora Postgre SQL
- Tampilan Database Tanpa Batas Aurora PostgreSQL


## Fungsi Basis Data Tanpa Batas Aurora Postgre SQL

Tabel berikut menunjukkan fungsi baru untuk Aurora SQL Postgre Limitless Database.

> **Note**
>
> Fungsi yang tercantum dalam tabel ini terletak di `rds_aurora` skema. Saat menggunakan fungsi Limitless Database, pastikan untuk menyertakan nama objek yang sepenuhnya memenuhi syarat:`rds_aurora.` *`object_name`*.

| Fungsi Basis Data Tanpa Batas Aurora Postgre SQL | Fungsi Aurora Postgre yang sesuai SQL |
|---|---|
| limitless_backend_dsid | pg_backend_pid |
| limitless_cancel_session | pg_cancel_backend |
| limitless_stat_clear_snapshot | pg_stat_clear_snapshot |
| limitless_stat_database_size | pg_database_size |

| Fungsi Basis Data Tanpa Batas Aurora Postgre SQL | Fungsi Aurora Postgre yang sesuai SQL |
|---|---|
| limitless_stat_get_snapshot_timestamp | pg_stat_get_snapshot_timestamp |
| limitless_stat_prepared_xacts | pg_prepared_xacts |
| limitless_stat_relation_sizes | pg_indexes_size, pg_relation_size, pg_table_size, pg_total_relation_size |
| limitless_stat_reset | pg_stat_reset |
| limitless_stat_statements_reset | pg_stat_statements_reset |
| limitless_stat_system_waits | aurora_stat_system_waits |
| limitless_terminate_session | pg_terminate_backend |
| limitless_wait_report | aurora_wait_report |

Contoh berikut memberikan rincian tentang fungsi Basis Data Tanpa Batas Aurora PostgreSQL. Untuk informasi selengkapnya tentang SQL fungsi Postgre, lihat Fungsi dan operator dalam dokumentasi SQL Postgre.

limitless_backend_dsid

`limitless_backend_dsid`Fungsi mengembalikan ID sesi terdistribusi untuk sesi saat ini. Sesi terdistribusi berjalan pada router dalam grup pecahan DB dan melibatkan proses backend pada satu atau lebih pecahan dalam grup pecahan DB.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_backend_dsid` fungsi.

```
SELECT rds_aurora.limitless_backend_dsid();

limitless_backend_dsid
------------------------
8CACD7B04D0FC2A5
(1 row)
```

limitless_cancel_session

`limitless_cancel_session`Fungsi ini bekerja mirip dengan`pg_cancel_backend`, tetapi mencoba untuk membatalkan semua proses backend yang terkait dengan ID sesi terdistribusi yang disediakan dengan mengirimkan `SIGINT` (sinyal interupsi).

Parameter input adalah sebagai berikut:

- `distributed_session_id`(teks) — ID sesi terdistribusi yang akan dibatalkan.

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `pid`(teks) — ID proses backend.
- `success`(boolean) — Apakah pembatalan berhasil.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_cancel_session` fungsi.

```
SELECT * FROM rds_aurora.limitless_cancel_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_stat_clear_snapshot

`limitless_stat_clear_snapshot`Fungsi membuang snapshot statistik saat ini atau informasi cache pada semua node.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_stat_clear_snapshot` fungsi.

```
SELECT rds_aurora.limitless_stat_clear_snapshot();
```

limitless_stat_database_size

`limitless_stat_database_size`Fungsi mengembalikan ukuran database dalam kelompok shard DB.

Parameter input adalah sebagai berikut:

- dbname(nama) — Database untuk mendapatkan ukuran.

Parameter output adalah sebagai berikut:

- subcluster_id(teks) — ID subcluster tempat proses ini berada.
- subcluster_type(teks) — Jenis subcluster yang menjadi milik proses ini: router ataushard.
- db_size— Ukuran database dalam subcluster ini dalam byte.

Contoh berikut menunjukkan bagaimana menggunakan limitless_stat_database_size fungsi.

```
SELECT * FROM rds_aurora.limitless_stat_database_size('postgres_limitless');

 subcluster_id | subcluster_type | db_size
---------------+-----------------+----------
             1 | router          |  8895919
             2 | router          |  8904111
             3 | shard           | 21929391
             4 | shard           | 21913007
             5 | shard           | 21831087
(5 rows)
```

limitless_stat_get_snapshot_timestamp

limitless_stat_get_snapshot_timestampFungsi mengembalikan stempel waktu snapshot statistik saat ini, atau NULL jika tidak ada snapshot statistik yang diambil. Sebuah snapshot diambil saat pertama kali statistik kumulatif diakses dalam transaksi jika stats_fetch_consistency diatur ke. snapshot Mengembalikan tampilan konsolidasi stempel waktu snapshot dari semua node. subcluster_typeKolom subcluster_id dan menunjukkan node mana datanya berasal.

Contoh berikut menunjukkan bagaimana menggunakan limitless_stat_get_snapshot_timestamp fungsi.

```
SELECT * FROM rds_aurora.limitless_stat_get_snapshot_timestamp();

 subcluster_id | subcluster_type | snapshot_timestamp
---------------+-----------------+--------------------
             1 | router          |
             2 | router          |
```

```
             3 | shard           |
             4 | shard           |
             5 | shard           |
(5 rows)
```

limitless_stat_prepared_xacts

`limitless_stat_prepared_xacts`Fungsi mengembalikan informasi tentang transaksi pada semua node yang saat ini disiapkan untuk komit dua fase. Untuk informasi selengkapnya, lihat pg_prepared_xacts di dokumentasi Postgre. SQL

Contoh berikut menunjukkan bagaimana menggunakan `limitless_stat_prepared_xacts` fungsi.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_prepared_xacts;

 subcluster_id | subcluster_type | transaction_id |             gid              |
         prepared            |  owner_id  |    database_id
---------------+-----------------+----------------+------------------------------
+-------------------------------+------------+--------------------
 8             | shard           |        5815978 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659603+00 | auroraperf | postgres_limitless
 12            | shard           |        4599138 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659637+00 | auroraperf | postgres_limitless
(2 rows)
```

limitless_stat_relation_sizes

`limitless_stat_relation_sizes`Fungsi mengembalikan ukuran yang berbeda dari tabel dalam kelompok shard DB.

Parameter input adalah sebagai berikut:

- `relnspname`(nama) — Nama skema yang berisi tabel.
- `relname`(nama) — Nama tabel.

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster yang menjadi milik proses ini: `router` atau`shard`.
- `main_size`— Ukuran dalam byte dari garpu data utama di node ini.

- `fsm_size`— Ukuran dalam byte peta ruang kosong untuk tabel di simpul ini.
- `vm_size`— Ukuran dalam byte peta visibilitas untuk tabel di node ini.
- `init_size`— Ukuran dalam byte inisialisasi tabel di node ini.
- `toast_size`— Ukuran dalam byte dari meja roti panggang yang terkait dengan tabel di garpu ini.
- `index_size`— Ukuran dalam byte dari semua indeks untuk tabel di node ini.
- `total_size`— Ukuran dalam byte dari semua segmen tabel di node ini.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_stat_relation_sizes` fungsi (beberapa kolom dihilangkan).

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customers');

 subcluster_id | subcluster_type | main_size | fsm_size | vm_size | toast_size |
 table_size | total_size
---------------+-----------------+-----------+----------+---------+------------
+------------+------------
             1 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             2 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             3 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             4 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             5 | shard           |   3981312 |  4227072 | 1409024 |    1409024 |
 11026432 |   11026432
(5 rows)
```

limitless_stat_reset

`limitless_stat_reset`Fungsi me-reset semua penghitung statistik untuk database saat ini ke nol (0). Jika `track_functions` diaktifkan, `stats_reset` kolom di `limitless_stat_database` menunjukkan statistik terakhir kali disetel ulang untuk database. Secara default, hanya `limitless_stat_reset` dapat dijalankan oleh superuser. Pengguna lain dapat diberikan izin dengan menggunakan hak `EXECUTE` istimewa.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_stat_reset` fungsi.

```
SELECT tup_inserted, tup_deleted FROM pg_stat_database
```

```
WHERE datname = 'postgres_limitless';

 tup_inserted | tup_deleted
--------------+-------------
          896 |           0
(1 row)

SELECT rds_aurora.limitless_stat_reset();

limitless_stat_reset
---------------------
(1 row)

SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

tup_inserted | tup_deleted
-------------+-------------
           0 |           0
(1 row)
```

limitless_stat_statements_reset

`limitless_stat_statements_reset`Fungsi membuang statistik yang dikumpulkan sejauh ini dengan `limitless_stat_statements` sesuai dengan yang ditentukan`username`,, `dbname``distributed_query_id`, dan `queryid` parameter. Jika salah satu parameter tidak ditentukan, nilai default "" atau `0` (tidak valid) digunakan untuk masing-masing parameter tersebut, dan statistik yang cocok dengan parameter lain akan disetel ulang. Jika tidak ada parameter yang ditentukan, atau semua parameter yang ditentukan adalah "" atau `0` (tidak valid), fungsi membuang semua statistik. Jika semua statistik dalam `limitless_stat_statements` tampilan dibuang, fungsi juga mengatur ulang statistik dalam tampilan. `limitless_stat_statements_info`

Parameter input adalah sebagai berikut:

- `username`(nama) — Pengguna yang menanyakan pernyataan tersebut.
- `dbname`(nama) — Database tempat kueri dijalankan.
- `distributed_query_id`(bigint) — ID kueri dari kueri induk dari node koordinator. Kolom ini adalah `NULL` jika itu adalah kueri induk. Node koordinator menekan ID kueri terdistribusi ke node peserta. Jadi untuk node peserta, nilai untuk ID kueri terdistribusi dan ID kueri berbeda.
- `queryid`(bigint) — ID kueri pernyataan.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_stat_statements_reset` fungsi untuk mengatur ulang semua statistik yang dikumpulkan oleh`limitless_stat_statements`.

```
SELECT rds_aurora.limitless_stat_statements_reset();
```

limitless_stat_system_waits

`limitless_stat_system_waits`Fungsi mengembalikan tampilan konsolidasi dari data peristiwa tunggu dari`aurora_stat_system_waits`, yang melaporkan aktivitas menunggu luas sistem dalam sebuah instance, dari semua node. `subcluster_type`Kolom `subcluster_id` dan menunjukkan node mana datanya berasal.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_stat_system_waits` fungsi.

```
postgres_limitless=> SELECT *
FROM rds_aurora.limitless_stat_system_waits() lssw, pg_catalog.aurora_stat_wait_event()
 aswe
WHERE lssw.event_id=aswe.event_id and aswe.event_name='LimitlessTaskScheduler';

 subcluster_id | subcluster_type | type_id | event_id  | waits  |  wait_time   |
 event_name
---------------+-----------------+---------+-----------+--------+--------------
+------------------------
             1 | router          |      12 | 201326607 | 677068 | 616942216307 |
 LimitlessTaskScheduler
             2 | router          |      12 | 201326607 | 678586 | 616939897111 |
 LimitlessTaskScheduler
             3 | shard           |      12 | 201326607 | 756640 | 616965545172 |
 LimitlessTaskScheduler
             4 | shard           |      12 | 201326607 | 755184 | 616958057620 |
 LimitlessTaskScheduler
             5 | shard           |      12 | 201326607 | 757522 | 616963183539 |
 LimitlessTaskScheduler
(5 rows)
```

limitless_terminate_session

`limitless_terminate_session`Fungsi ini bekerja mirip denganpg_terminate_backend, tetapi mencoba untuk mengakhiri semua proses backend yang terkait dengan ID sesi terdistribusi yang disediakan dengan mengirimkan `SIGTERM` (sinyal akhir).

Parameter input adalah sebagai berikut:

- `distributed_session_id`(teks) — ID sesi terdistribusi yang akan diakhiri.

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `pid`(teks) — ID proses backend.
- `success`(boolean) - Apakah proses berhasil berakhir.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_terminate_session` fungsi.

```
SELECT * FROM rds_aurora.limitless_terminate_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_wait_report

`limitless_wait_report`Fungsi mengembalikan aktivitas acara tunggu selama periode waktu dari semua node. `subcluster_type`Kolom `subcluster_id` dan menunjukkan node mana datanya berasal.

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster yang menjadi milik proses ini: `router` atau`shard`.

Sisa kolom sama seperti di`aurora_wait_report`.

Contoh berikut menunjukkan bagaimana menggunakan `limitless_wait_report` fungsi.

```
postgres_limitless=> select * from rds_aurora.limitless_wait_report();

 subcluster_id | subcluster_type | type_name | event_name | waits | wait_time |
 ms_per_wait | waits_per_xact | ms_per_xact
---------------+-----------------+-----------+------------+-------+-----------
+-------------+--------------- +-------------
```

```
            1 | router          | Client    | ClientRead |    57 | 741550.14 |
13009.652 |           0.19 |    2505.237
            5 | shard           | Client    | ClientRead |    54 | 738897.68 |
13683.290 |           0.18 |    2496.276
            4 | shard           | Client    | ClientRead |    54 | 738859.53 |
13682.584 |           0.18 |    2496.147
            2 | router          | Client    | ClientRead |    53 | 719223.64 |
13570.257 |           0.18 |    2429.810
            3 | shard           | Client    | ClientRead |    54 | 461720.40 |
8550.378 |           0.18 |    1559.86
```

## Tampilan Database Tanpa Batas Aurora PostgreSQL

Tabel berikut menunjukkan tampilan baru untuk Aurora PostgreSQL Limitless Database.

> **Note**
>
> Tampilan yang tercantum dalam tabel ini terletak di `rds_aurora` skema. Saat menggunakan tampilan Limitless Database, pastikan untuk menyertakan nama objek yang sepenuhnya memenuhi syarat:`rds_aurora.` *`object_name`*.

| Tampilan Database Tanpa Batas Aurora PostgreSQL | Tampilan PostgreSQL Aurora yang sesuai |
|---|---|
| limitless_database | pg_database |
| limitless_locks | pg_locks |
| limitless_stat_activity | pg_stat_activity |
| limitless_stat_all_indexes | pg_stat_all_indexes |
| limitless_stat_all_tables | pg_stat_all_tables |
| limitless_stat_database | pg_stat_database |
| limitless_stat_progress_vacuum | pg_stat_progress_vacuum |
| limitless_stat_statement | pg_stat_statements |
| limitless_stat_subcluster | Tidak ada |
| limitless_stat_statements_info | pg_stat_statements_info |
| limitless_statio_all_indexes | pg_statio_all_indexes |
| limitless_statio_all_tables | pg_statio_all_tables |
| limitless_tables | pg_tables |
| limitless_table_collocations | Tidak ada |

| Tampilan Database Tanpa Batas Aurora PostgreSQL | Tampilan PostgreSQL Aurora yang sesuai |
| --- | --- |
| limitless_table_collocation_distributions | Tidak ada |

Contoh berikut memberikan rincian tentang tampilan Aurora PostgreSQL Limitless Database. Untuk informasi selengkapnya tentang tampilan PostgreSQL, lihat Melihat statistik dalam dokumentasi PostgreSQL.

> **Note**
>
> Beberapa tampilan statistik dapat mengembalikan hasil yang tidak konsisten jika Anda memiliki transaksi yang sedang berlangsung.

limitless_database

Tampilan ini berisi informasi tentang database yang tersedia di grup shard DB. Misalnya:

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, oid, datname, datacl
 FROM rds_aurora.limitless_database;

 subcluster_id | subcluster_type |  oid  |      datname       |
                                 datacl

---------------+-----------------+-------+--------------------
+-------------------------------------------------------------------------------------------
 2             | router          |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 2             | router          |     5 | postgres           |
 2             | router          | 16384 | rdsadmin           |
 {rdsadmin=CTc/rdsadmin,rds_aurora_limitless_metadata_admin=c/
rdsadmin,rds_aurora_limitless_heat_mgmt_admin=c/rdsadmin}
 2             | router          | 16477 | postgres_limitless |
 2             | router          |     1 | template1          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 6             | shard           |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(text) — ID dari subcluster (node)
- `subcluster_type`(teks) — Jenis subcluster (node), router atau shard

Sisa kolom sama seperti di`pg_database`.

limitless_locks

Tampilan ini berisi satu baris per proses per node. Ini menyediakan akses ke informasi tentang kunci yang dipegang oleh proses aktif di server database.

Example membuat kunci dengan dua transaksi

Dalam contoh ini, kami menjalankan dua transaksi secara bersamaan pada dua router.

```
# Transaction 1 (run on router 1)
BEGIN;
SET search_path = public;
SELECT * FROM customers;
INSERT INTO customers VALUES (400,'foo','bar');

# Transaction 2 (run on router 2)
BEGIN;
SET search_path = public;
ALTER TABLE customers ADD COLUMN phone VARCHAR;
```

Transaksi pertama dijalankan. Transaksi selanjutnya harus menunggu hingga transaksi pertama selesai. Oleh karena itu transaksi kedua diblokir dengan kunci. Untuk memeriksa akar penyebabnya, kami menjalankan perintah dengan bergabung `limitless_locks` dengan`limitless_stat_activity`.

```
# Run on router 2
SELECT distributed_session_id, state, usename, query, query_start
FROM rds_aurora.limitless_stat_activity
WHERE distributed_session_id in (
SELECT distributed_session_id
FROM rds_aurora.limitless_locks
WHERE relname = 'customers'
);

 distributed_session_id | state               | usename                 | query
                                  | query_start
------------------------+---------------------+-------------------------
+---------------------------------- -------------+-------------------------------
```

```
 47BDE66E9A5E8477       | idle in transaction | limitless_metadata_admin | INSERT INTO
 customers VALUES (400,'foo','bar'); | 2023-04-13 17:44:45.152244+00
 2AD7F370202D0FA9       | active              | limitless_metadata_admin | ALTER TABLE
 customers ADD COLUMN phone VARCHAR; | 2023-04-13 17:44:55.113388+00
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
(6 rows)
```

Example membuat kunci secara eksplisit

Dalam contoh ini, kita membuat kunci secara eksplisit, kemudian menggunakan `limitless_locks` tampilan untuk melihat kunci (beberapa kolom dihilangkan).

```
BEGIN;
SET search_path = public;
LOCK TABLE customers IN ACCESS SHARE MODE;
SELECT * FROM rds_aurora.limitless_locks WHERE relname = 'customers';

 subcluster_id | subcluster_type | distributed_session_id | locktype |      datname
   | relnspname |  relname  | virtualtransaction |  pid  |      mode
---------------+-----------------+------------------------+----------
+--------------------+------------+ ----------+--------------------+-------
+-----------------
             1 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600787          | 59564 |
 AccessShareLock
             2 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600405          | 67130 |
 AccessShareLock
             3 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 15/473401          | 27735 |
 AccessShareLock
             4 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473524          | 27734 |
 AccessShareLock
```

```
             5 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/472935          | 27737 |
 AccessShareLock
             6 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473015          | 48660 |
 AccessShareLock
(6 rows)
```

limitless_stat_activity

Tampilan ini berisi satu baris per proses per node. Ini dapat digunakan untuk melacak kesehatan sistem secara keseluruhan dan proses triase yang memakan waktu lama. Misalnya:

```
postgres=# SELECT
    subcluster_id,
    subcluster_type,
    distributed_session_id,
    distributed_session_state,
    datname,
    distributed_query_id,
    is_sso_query
FROM
    rds_aurora.limitless_stat_activity
WHERE
    distributed_session_id in ('D2470C97E3D07E06', '5A3CD7B8E5FD13FF')
    order by  distributed_session_id;

 subcluster_id | subcluster_type | distributed_session_id | distributed_session_state |
      datname       | distributed_query_id | is_sso_query
---------------+-----------------+------------------------+---------------------------
+--------------------+----------------------+-------------
 2             | router          | 5A3CD7B8E5FD13FF       | coordinator               |
 postgres_limitless |                      | f
 3             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 4             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 2             | router          | D2470C97E3D07E06       | coordinator               |
 postgres_limitless |                      | t
 3             | shard           | D2470C97E3D07E06       | participant               |
 postgres_limitless |  4058400544464210222 |
(5 rows)
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster tempat proses ini berada: `router` atau`shard`.
- `distributed_session_id`(teks) — ID sesi terdistribusi tempat proses ini berada.
- `distributed_session_state`(teks) — Apakah ini koordinator, peserta, atau standalone/nondistributed proses (ditampilkan sebagai`NULL`).
- `datname`(teks) — Database yang terhubung dengan proses ini.
- `distributed_query_id`(bigint) — ID kueri dari kueri induk dari node koordinator. Kolom ini adalah `NULL` jika itu adalah kueri induk. Node koordinator menekan ID kueri terdistribusi ke node peserta. Jadi untuk node peserta, nilai untuk ID kueri terdistribusi dan ID kueri berbeda.
- `is_sso_query`(teks) - Ini memungkinkan kita tahu apakah kueri adalah pecahan tunggal yang dioptimalkan atau tidak.

Sisa kolom sama seperti di`pg_stat_activity`.

limitless_stat_all_indexes

Tampilan ini berisi statistik penggunaan pada indeks dalam grup pecahan DB. Misalnya:

```
postgres_limitless=> SELECT schemaname, relname, indexrelname, idx_scan
  FROM rds_aurora.limitless_stat_all_indexes
  WHERE relname LIKE 'orders_ts%' ORDER BY indexrelname LIMIT 10;

 schemaname |    relname     |    indexrelname     | idx_scan
------------+----------------+---------------------+----------
 ec_sample  | orders_ts00001 | orders_ts00001_pkey |   196801
 ec_sample  | orders_ts00002 | orders_ts00002_pkey |   196703
 ec_sample  | orders_ts00003 | orders_ts00003_pkey |   196376
 ec_sample  | orders_ts00004 | orders_ts00004_pkey |   197966
 ec_sample  | orders_ts00005 | orders_ts00005_pkey |   195301
 ec_sample  | orders_ts00006 | orders_ts00006_pkey |   195673
 ec_sample  | orders_ts00007 | orders_ts00007_pkey |   194475
 ec_sample  | orders_ts00008 | orders_ts00008_pkey |   191694
 ec_sample  | orders_ts00009 | orders_ts00009_pkey |   193744
 ec_sample  | orders_ts00010 | orders_ts00010_pkey |   195421
(10 rows)
```

limitless_stat_all_tables

Tampilan ini berisi statistik tentang semua tabel dalam database saat ini di grup shard DB. Ini berguna saat melacak operasi vakum dan operasi bahasa Manipulasi Data (DML/Data Manipulation Language). Misalnya:

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, relname,
 n_ins_since_vacuum, n_tup_ins, last_vacuum
  FROM rds_aurora.limitless_stat_all_tables
  WHERE relname LIKE 'orders_ts%' ORDER BY relname LIMIT 10;

 subcluster_id | subcluster_type |    relname     | n_ins_since_vacuum | n_tup_ins |
 last_vacuum
---------------+-----------------+----------------+--------------------+-----------
+-------------
 5             | shard           | orders_ts00001 |              34779 |    196083 |
 5             | shard           | orders_ts00002 |              34632 |    194721 |
 5             | shard           | orders_ts00003 |              34950 |    195965 |
 5             | shard           | orders_ts00004 |              34745 |    197283 |
 5             | shard           | orders_ts00005 |              34879 |    195754 |
 5             | shard           | orders_ts00006 |              34340 |    194605 |
 5             | shard           | orders_ts00007 |              33779 |    192203 |
 5             | shard           | orders_ts00008 |              33826 |    191293 |
 5             | shard           | orders_ts00009 |              34660 |    194117 |
 5             | shard           | orders_ts00010 |              34569 |    195560 |
(10 rows)
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster tempat proses ini berada: `router` atau`shard`.
- `relname`(nama) — Nama tabel.

Sisa kolom sama seperti di`pg_stat_all_tables`.

limitless_stat_database

Tampilan ini berisi statistik tentang semua database dalam grup shard DB. Mengembalikan satu baris per database per node. Misalnya:

```
postgres_limitless=> SELECT
    subcluster_id,
```

```
    subcluster_type,
    datname,
    blks_read,
    blks_hit
FROM
    rds_aurora.limitless_stat_database
WHERE
    datname='postgres_limitless';
 subcluster_id | subcluster_type |      datname       | blks_read | blks_hit
---------------+-----------------+--------------------+-----------+----------
             1 | router          | postgres_limitless |       484 | 34371314
             2 | router          | postgres_limitless |       673 | 33859317
             3 | shard           | postgres_limitless |      1299 | 17749550
             4 | shard           | postgres_limitless |      1094 | 17492849
             5 | shard           | postgres_limitless |      1036 | 17485098
             6 | shard           | postgres_limitless |      1040 | 17437257
(6 rows)
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster tempat proses ini berada: `router` atau`shard`.
- `datname`(nama) — Nama database.

Sisa kolom sama seperti di`pg_stat_database`.

limitless_stat_progress_vacuum

Tampilan ini berisi informasi tentang operasi penyedot debu yang sedang berlangsung. Misalnya:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;

-[ RECORD 1 ]----------+-----------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | A56D96E2A5C9F426
pid                    | 5270
datname                | postgres
nspname                | public
relname                | customer_ts2
phase                  | vacuuming heap
heap_blks_total        | 130500
heap_blks_scanned      | 100036
```

```
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 11184810
num_dead_tuples        | 0

-[ RECORD 2 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | 56DF26A89EC23AB5
pid                    | 6854
datname                | postgres
nspname                | public
relname                | sales_ts1
phase                  | vacuuming heap
heap_blks_total        | 43058
heap_blks_scanned      | 24868
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 8569523
num_dead_tuples        | 0
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster tempat proses ini berada: `router` atau`shard`.
- `distributed_session_id`(teks) — Pengidentifikasi untuk sesi yang memulai operasi penyedot debu.
- `datname`(nama) — Database tempat penyedot debu sedang dilakukan.
- `nspname`(nama) — Nama skema tabel yang sedang disedot. Itu `null` jika tabel yang disedot tidak dalam database yang sama dengan yang terhubung dengan pengguna.
- `relname`(nama) — Nama tabel yang sedang disedot. Itu `null` jika tabel yang disedot tidak dalam database yang sama dengan yang terhubung dengan pengguna.

Sisa kolom sama seperti di`pg_stat_progress_vacuum`.

limitless_stat_statement

Tampilan ini menyediakan sarana untuk melacak perencanaan dan menjalankan statistik dari semua pernyataan SQL yang dijalankan di semua node.

> **Note**
>
> Anda harus menginstal ekstensi pg_stat_statement untuk menggunakan tampilan. `limitless_stat_statements`
>
> ```
> -- CREATE EXTENSION must be run by a superuser
> CREATE EXTENSION pg_stat_statements;
>
> -- Verify that the extension is created on all nodes in the DB shard group
> SELECT distinct node_id
>     FROM rds_aurora.limitless_stat_statements
>     LIMIT 10;
> ```

Contoh berikut menunjukkan penggunaan `limitless_stat_statements` tampilan.

```
postgres_limitless=> SELECT
 subcluster_id,
 subcluster_type,
 distributedqueryid,
 username,
 dbname,
 sso_calls
FROM
 rds_aurora.limitless_stat_statements;

 subcluster_id | subcluster_type |  distributedqueryid  |             username
       |       dbname       | sso_calls
---------------+-----------------+----------------------
+-------------------------------------+---------------------+-----------
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         1
```

```
 3             | shard           | -7975178695405682176 | postgres
       | postgres_limitless |
[...]
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.
- `subcluster_type`(teks) — Jenis subcluster yang menjadi milik proses ini: `router` untuk atau`shard`.
- `distributedqueryid`(bigint) — ID kueri dari kueri induk dari node koordinator. Kolom ini adalah NULL jika itu adalah kueri induk. Node koordinator menekan ID kueri terdistribusi ke node peserta. Jadi untuk node peserta, nilai untuk ID kueri terdistribusi dan ID kueri berbeda.
- `username`(nama) — Pengguna yang menanyakan pernyataan tersebut.
- dbname(nama) — Database tempat kueri dijalankan.
- `sso_calls`(nama) — Berapa kali pernyataan dioptimalkan dengan pecahan tunggal.

Sisa kolom sama seperti di pg_stat_statement.

limitless_stat_statements_info

Tampilan ini berisi statistik untuk `limitless_stat_statements` tampilan. Setiap baris berisi data untuk tampilan pg_stat_statements_info dari setiap node. `subcluster_id`Kolom mengidentifikasi setiap node.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_statements_info;

 subcluster_id | subcluster_type | dealloc |          stats_reset
---------------+-----------------+---------+-------------------------------
             1 | router          |       0 | 2023-06-30 21:22:09.524781+00
             2 | router          |       0 | 2023-06-30 21:21:40.834111+00
             3 | shard           |       0 | 2023-06-30 21:22:10.709942+00
             4 | shard           |       0 | 2023-06-30 21:22:10.740179+00
             5 | shard           |       0 | 2023-06-30 21:22:10.774282+00
             6 | shard           |       0 | 2023-06-30 21:22:10.808267+00
(6 rows)
```

Parameter output adalah sebagai berikut:

- `subcluster_id`(teks) — ID subcluster tempat proses ini berada.

Sisa kolom sama seperti di pg_stat_statements_info.

limitless_stat_subcluster

Tampilan ini berisi statistik jaringan antara router dan node lainnya. Ini berisi baris per pasang router dan node lainnya, misalnya:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_subclusters;

 orig_subcluster | orig_instance_az | dest_subcluster | dest_instance_az |
 latency_us |       latest_collection       | failed_requests | received_bytes |
 sent_bytes | same_az_requests | cross_az_requests |     stat_reset_timestamp
-----------------+------------------+-----------------+------------------
+------------+-------------------------------+-----------------+---------------
+------------+------------------+-------------------+------------------------------
 3               | us-west-2b       | 2               | us-west-2a       |
 847 | 2024-10-07 17:25:39.518617+00 |              0 |       35668633 |   92090171
 |                0 |           302787 | 2024-10-05 12:39:55.239675+00
 3               | us-west-2b       | 4               | us-west-2b       |
 419 | 2024-10-07 17:25:39.546376+00 |              0 |      101190464 |  248795719
 |           883478 |                0 | 2024-10-05 12:39:55.231218+00
 3               | us-west-2b       | 5               | us-west-2c       |
    1396 | 2024-10-07 17:25:39.52122+00  |              0 |       72864849 |
 172086292 |               0 |           557726 | 2024-10-05 12:39:55.196412+00
 3               | us-west-2b       | 6               | us-west-2c       |
 729 | 2024-10-07 17:25:39.54828+00  |              0 |       35668584 |   92090171
 |                0 |           302787 | 2024-10-05 12:39:55.247334+00
 3               | us-west-2b       | 7               | us-west-2a       |
    1702 | 2024-10-07 17:25:39.545307+00 |              0 |       71699576 |
 171634844 |               0 |           556278 | 2024-10-05 12:39:52.715168+00
 2               | us-west-2a       | 3               | us-west-2b       |
 868 | 2024-10-07 17:25:40.293927+00 |              0 |       35659611 |   92011872
 |                0 |           302817 | 2024-10-05 12:39:54.420758+00
 2               | us-west-2a       | 4               | us-west-2b       |
 786 | 2024-10-07 17:25:40.296863+00 |              0 |      102437253 |  251838024
 |                0 |           895060 | 2024-10-05 12:39:54.404081+00
 2               | us-west-2a       | 5               | us-west-2c       |
    1232 | 2024-10-07 17:25:40.292021+00 |              0 |       71990027 |
 168828110 |               0 |           545453 | 2024-10-05 12:39:36.769549+00
```

Parameter output adalah sebagai berikut:

- `orig_subcluster`(teks) — ID router tempat komunikasi berasal
- `orig_subcluster_az`(teks) — Availability Zone (AZ) dari router pencetus
- `dest_subcluster`(teks) — ID dari simpul tujuan

- `dest_subcluster_az`(teks) — AZ terakhir yang dikumpulkan dari simpul tujuan
- `latency_us`(bigint) — Latensi jaringan terakhir yang dikumpulkan antar node, dalam mikrodetik. Nilainya adalah `0` jika node tidak dapat dijangkau.
- `latest_collection`(stempel waktu) - Stempel waktu koleksi terbaru AZ dan latensi untuk node tujuan
- `failed_requests`(bigint) — Jumlah kumulatif permintaan internal yang gagal
- `received_bytes`(bigint) — Perkiraan jumlah kumulatif byte yang diterima dari node ini
- `sent_bytes`(bigint) — Perkiraan jumlah kumulatif byte yang dikirim ke node ini
- `same_az_requests`(bigint) — Jumlah kumulatif permintaan DB internal ke node ini ketika berada di AZ yang sama dengan router pencetus
- `cross_az_requests`(bigint) — Jumlah kumulatif permintaan DB internal ke node ini saat berada di AZ yang berbeda dari router pencetus
- `stat_reset_timestamp`(stempel waktu) — Stempel waktu saat statistik kumulatif untuk tampilan ini terakhir disetel ulang

limitless_statio_all_indexes

Tampilan ini berisi statistik input/output (I/O) untuk semua indeks dalam grup pecahan DB. Misalnya:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_statio_all_indexes WHERE
 relname like'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      |
 indexrelname             | idx_blks_read | idx_blks_hit
--------------+-----------------+------------+-------------------
+-----------------------------------+ --------------+--------------
             3 | shard           | public     | customers_ts00002 |
 customers_ts00002_customer_name_idx |             1 |            0
             3 | shard           | public     | customers_ts00001 |
 customers_ts00001_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00003 |
 customers_ts00003_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00004 |
 customers_ts00004_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00005 |
 customers_ts00005_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00006 |
 customers_ts00006_customer_name_idx |             1 |            0
```

```
            6 | shard           | public     | customers_ts00007 |
 customers_ts00007_customer_name_idx |             1 |           0
            6 | shard           | public     | customers_ts00008 |
 customers_ts00008_customer_name_idx |             1 |           0
(8 rows)
```

limitless_statio_all_tables

Tampilan ini berisi statistik input/output (I/O) untuk semua tabel dalam grup pecahan DB. Misalnya:

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    schemaname,
    relname,
    heap_blks_read,
    heap_blks_hit
FROM
    rds_aurora.limitless_statio_all_tables
WHERE
    relname LIKE 'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      | heap_blks_read |
 heap_blks_hit
---------------+-----------------+------------+-------------------+----------------
+---------------
            3 | shard           | public     | customers_ts00002 |            305 |
      57780
            3 | shard           | public     | customers_ts00001 |            300 |
      56972
            4 | shard           | public     | customers_ts00004 |            302 |
      57291
            4 | shard           | public     | customers_ts00003 |            302 |
      57178
            5 | shard           | public     | customers_ts00006 |            300 |
      56932
            5 | shard           | public     | customers_ts00005 |            302 |
      57386
            6 | shard           | public     | customers_ts00008 |            300 |
      56881
            6 | shard           | public     | customers_ts00007 |            304 |
      57635
```

```
(8 rows)
```

limitless_tables

Tampilan ini berisi informasi tentang tabel di Aurora PostgreSQL Limitless Database.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | placeholder | active       | placeholder |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     | HASH
 (order_id)
         2 |     18579 | public      | customer    | active       | sharded     | HASH
 (cust_id)
(4 rows)
```

limitless_table_collocations

Tampilan ini berisi informasi tentang tabel sharded collocated.

Dalam contoh berikut, `customers` tabel `orders` dan ditempatkan, dan `followers` tabel `users` dan ditempatkan. Tabel collocated memiliki hal yang sama`collocation_id`.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
              2 | public      | orders
              2 | public      | customers
              5 | public      | users
              5 | public      | followers
(4 rows)
```

limitless_table_collocation_distributions

Tampilan ini menunjukkan distribusi kunci untuk setiap kolokasi.

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
              2 |             6 | -9223372036854775808 | -4611686018427387904
              2 |             5 | -4611686018427387904 |                    0
              2 |             4 |                    0 |  4611686018427387904
              2 |             3 |  4611686018427387904 |  9223372036854775807
              5 |             6 | -9223372036854775808 | -4611686018427387904
              5 |             5 | -4611686018427387904 |                    0
              5 |             4 |                    0 |  4611686018427387904
              5 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

# Tunggu acara untuk Aurora SQL Postgre Limitless Database

Acara tunggu di Aurora Postgre SQL menunjukkan sumber daya yang sedang menunggu sesi, sepertiinput/output (I/O) dan kunci. Acara tunggu sangat membantu dalam mencari tahu mengapa sesi menunggu sumber daya, dan mengidentifikasi kemacetan. Untuk informasi selengkapnya, lihat Peristiwa tunggu Aurora PostgreSQL.

Aurora Postgre SQL Limitless Database memiliki acara tunggu sendiri yang terkait dengan router dan pecahan. Banyak dari mereka adalah untuk router yang menunggu pecahan untuk menyelesaikan tugas. Acara tunggu pecahan berisi detail tentang tugas yang sedang dilakukan.

Topik

- Meminta acara tunggu
- Acara tunggu Database Tanpa Batas

## Meminta acara tunggu

Anda dapat menggunakan tampilan limitless_stat_activity untuk menanyakan peristiwa tunggu, seperti yang ditunjukkan pada contoh berikut.

```
SELECT wait_event FROM rds_aurora.limitless_stat_activity WHERE
 wait_event_type='AuroraLimitless';

      wait_event
----------------------
 RemoteStatementSetup
 RemoteStatementSetup
(2 rows)
```

Anda juga dapat menggunakan `aurora_stat_system_waits` fungsi ini untuk mencantumkan jumlah menunggu dan total waktu yang dihabiskan untuk setiap acara tunggu, seperti yang ditunjukkan pada contoh berikut.

```
postgres_limitless=> SELECT type_name,event_name,waits,wait_time
    FROM aurora_stat_system_waits()
    NATURAL JOIN aurora_stat_wait_event()
    NATURAL JOIN aurora_stat_wait_type()
    WHERE type_name='AuroraLimitless'
    ORDER BY wait_time DESC;
```

```
    type_name    |       event_name          |  waits  |  wait_time
-----------------+---------------------------+---------+-------------
 AuroraLimitless | RemoteStatementSetup      |    7518 | 75236507897
 AuroraLimitless | RemoteStatementExecution  |      40 |      132986
 AuroraLimitless | Connect                   |       5 |        1453
(3 rows)
```

## Acara tunggu Database Tanpa Batas

Acara tunggu berikut berlaku untuk Aurora PostgreSQL Limitless Database. Anda dapat memantau peristiwa tunggu ini untuk mengidentifikasi kemacetan dalam pemrosesan Basis Data Aurora PostgreSQL Limitless.

Topik

- IO: TwophaseFilePoolWrite tunggu acara
- IO: TwophaseFilePoolRead tunggu acara
- AuroraLimitless:Hubungkan acara tunggu
- AuroraLimitless: AsyncConnect tunggu acara
- AuroraLimitless: RemoteStatementSetup tunggu acara
- AuroraLimitless:Acara DDLExecution tunggu jarak jauh
- AuroraLimitless: RemoteStatementExecution tunggu acara
- AuroraLimitless: FetchRemoteResults tunggu acara
- AuroraLimitless: AsyncGetInitialResponse tunggu acara
- AuroraLimitless: AsyncGetNextResponse tunggu acara
- AuroraLimitless: AbortedCommandCleanup tunggu acara
- AuroraLimitless: DistributedCommitPrepare tunggu acara
- AuroraLimitless: DistributedCommit tunggu acara
- AuroraLimitless: DistributedCommitPrepareThrottle tunggu acara
- AuroraLimitless: PreparedTransactionResolution tunggu acara
- AuroraLimitless: SendPreparedTransactionOutcome tunggu acara
- AuroraLimitless: CommitClockBarrier tunggu acara
- AuroraLimitless: SnapshotClockBarrier tunggu acara

- AuroraLimitless: ReaderSnapshotClockBarrier tunggu acara
- AuroraLimitless: GatherDistributedDeadlockGraph tunggu acara
- AuroraLimitless: DistributedDeadlockDetection tunggu acara
- AuroraLimitless: DistributedDeadlockAbort tunggu acara
- AuroraLimitless: GatherRemoteStats tunggu acara
- AuroraLimitless: GlobalSequenceRefresh tunggu acara
- AuroraLimitless: GlobalVacuumTimeExchange tunggu acara
- AuroraLimitless: DistributedTransactionMonitorGather tunggu acara
- AuroraLimitlessActivity: AdminTaskSchedulerMain tunggu acara
- AuroraLimitlessActivity: AdminTaskExecutorMain tunggu acara
- AuroraLimitlessActivity: AdminTaskMonitorMain tunggu acara
- AuroraLimitlessActivity: DatabaseCleanupMonitorMain tunggu acara
- AuroraLimitlessActivity: TopologyCleanupMonitorMain tunggu acara
- AuroraLimitlessActivity: ToplogyChangeMonitorMain tunggu acara
- AuroraLimitlessActivity: DistributedTransactionMonitorMain tunggu acara
- AuroraLimitlessActivity: GlobalVacuumMonitorMain tunggu acara

IO: TwophaseFilePoolWrite tunggu acara

Menunggu penulisan file status dua fase dalam kumpulan file status dua fase. Ini adalah acara khusus Aurora.

Penyebab

Proses mengeksekusi `PREPARED TRANSACTION` perintah, termasuk peserta dalam transaksi terdistribusi Limitless Database, harus mempertahankan status transaksi dalam file dua fase. Aurora menggunakan kumpulan file untuk meningkatkan kinerja operasi ini.

Tindakan

Ini adalah operasi I/O tulis sinkron dan oleh karena itu latensi tinggi dalam acara ini memiliki penyebab yang sama `IO:XactSync` dan dapat diselidiki dengan cara yang sama. Jika menggunakan Limitless Database, Anda mungkin perlu mengurangi jumlah transaksi terdistribusi yang dieksekusi.

IO: TwophaseFilePoolRead tunggu acara

Menunggu pembacaan file status dua fase dalam kumpulan file status dua fase.

Penyebab

Proses mengeksekusi `COMMIT PREPARED` perintah terhadap transaksi yang disiapkan sebelumnya, termasuk peserta dalam transaksi terdistribusi Limitless Database, mungkin perlu membaca status transaksi yang bertahan sebelumnya dari file dua fase. Aurora menggunakan kumpulan file untuk meningkatkan kinerja operasi ini.

Tindakan

Ini adalah operasi baca I/O. Oleh karena itu, latensi tinggi dalam peristiwa ini memiliki penyebab yang sama `IO:DataFileRead` dan dapat diselidiki sama. Jika menggunakan Limitless Database, Anda mungkin perlu mengurangi jumlah transaksi terdistribusi yang dieksekusi.

AuroraLimitless:Hubungkan acara tunggu

Proses ini menunggu koneksi ke node lain di cluster yang akan dibuat.

Penyebab

Koneksi dibuat antara proses dan node jarak jauh untuk mengeksekusi kueri, transaksi terdistribusi, dan melakukan DDLs.

Tindakan

Kurangi jumlah koneksi simultan ke cluster atau sesuaikan penggunaan kueri cross-shard.

AuroraLimitless: AsyncConnect tunggu acara

Peristiwa ini mirip dengan`Connect`, tetapi merupakan proses menunggu koneksi paralel ke satu set node yang akan dibuat.

Penyebab

Pembentukan koneksi paralel paling sering dilakukan ketika mengeksekusi pernyataan DDL.

Tindakan

Kurangi jumlah pernyataan DDL atau gabungkan beberapa DDLs di sesi yang sama untuk meningkatkan penggunaan kembali koneksi.

AuroraLimitless: RemoteStatementSetup tunggu acara

Proses ini menunggu penyiapan eksekusi kueri jarak jauh, seperti kursor buka, tutup, atau pembuatan pernyataan yang disiapkan.

Penyebab

Peristiwa tunggu ini meningkat dengan jumlah pemindaian pada tabel sharded di mana pernyataan tidak dapat dioptimalkan dengan pecahan tunggal.

Tindakan

Optimalkan kueri untuk mengurangi jumlah operasi pemindaian atau meningkatkan kelayakan untuk pengoptimalan pecahan tunggal.

AuroraLimitless:Acara DDLExecution tunggu jarak jauh

Prosesnya sedang menunggu perintah DDL jarak jauh selesai.

Penyebab

Ketika Anda mengeluarkan perintah DDL pada grup shard DB, itu harus didistribusikan ke router lain dan node shard sebelum mengonfirmasi operasi. Beberapa operasi DDL dapat berjalan untuk waktu yang lama, karena data harus disesuaikan dengan perubahan skema.

Tindakan

Identifikasi perintah DDL yang berjalan lama sehingga Anda dapat mengoptimalkannya.

AuroraLimitless: RemoteStatementExecution tunggu acara

Sebuah proses sedang menunggu perintah jarak jauh selesai.

Penyebab

Perintah SQL berjalan pada node jarak jauh. Acara ini akan sering muncul untuk komunikasi internal, seperti `auto_analyze` dan pemeriksaan detak jantung.

Tindakan

Identifikasi perintah yang berjalan lama menggunakan tampilan limitless_stat_statement. Dalam banyak kasus ini adalah peristiwa yang diharapkan, terutama untuk pekerja latar belakang atau proses internal dan tidak diperlukan tindakan.

AuroraLimitless: FetchRemoteResults tunggu acara

Sebuah proses sedang menunggu untuk mengambil baris dari node jarak jauh.

Penyebab

Peristiwa tunggu ini dapat meningkat saat mengambil sejumlah besar baris dari tabel jarak jauh, seperti tabel sharded atau referensi.

Tindakan

Identifikasi `SELECT` kueri yang tidak dioptimalkan menggunakan tampilan. `limitless_stat_statements` Optimalkan kueri untuk mengambil hanya data yang diperlukan. Anda juga dapat menyetel `rds_aurora.limitless_maximum_adaptive_fetch_size` parameternya.

AuroraLimitless: AsyncGetInitialResponse tunggu acara

Proses ini menunggu respons awal ketika mode pipeline digunakan dalam eksekusi kueri.

Penyebab

Ini biasanya akan terlihat selama eksekusi router to shard untuk kueri dengan penempatan data shard tunggal dan merupakan bagian yang diharapkan dari eksekusi normal.

Tindakan

Tidak diperlukan tindakan lebih lanjut.

AuroraLimitless: AsyncGetNextResponse tunggu acara

Proses ini menunggu tanggapan tambahan ketika mode pipeline digunakan dalam eksekusi kueri.

Penyebab

Ini biasanya akan terlihat selama eksekusi router to shard untuk kueri dengan penempatan data shard tunggal dan merupakan bagian yang diharapkan dari eksekusi normal.

Tindakan

Tidak diperlukan tindakan lebih lanjut.

AuroraLimitless: AbortedCommandCleanup tunggu acara

Prosesnya menunggu hasil kueri pembersihan jarak jauh. Kueri pembersihan dikeluarkan ke node shard untuk mengembalikannya ke keadaan yang sesuai ketika transaksi terdistribusi berakhir.

Penyebab

Pembersihan transaksi dilakukan ketika transaksi dibatalkan baik karena kesalahan ditemukan atau karena pengguna mengeluarkan perintah ABORT eksplisit atau membatalkan kueri yang sedang berjalan.

Tindakan

Selidiki penyebab transaksi dibatalkan.

AuroraLimitless: DistributedCommitPrepare tunggu acara

Proses ini melakukan transaksi terdistribusi dan menunggu semua peserta untuk mengakui perintah persiapan.

Penyebab

Transaksi yang memodifikasi beberapa node harus melakukan komit terdistribusi. Penantian yang lama `DistributedCommitPrepare` dapat disebabkan oleh menunggu lama di `IO:TwophaseFilePoolWrite` acara pada node yang berpartisipasi.

Tindakan

Kurangi jumlah transaksi yang memodifikasi data pada beberapa node. Selidiki `IO:TwophaseFilePoolWrite` peristiwa di node lain dari cluster.

AuroraLimitless: DistributedCommit tunggu acara

Proses ini melakukan transaksi terdistribusi dan sedang menunggu peserta utama untuk mengakui perintah komit.

Penyebab

Transaksi yang memodifikasi beberapa node harus melakukan komit terdistribusi. Penantian yang lama `DistributedCommit` bisa disebabkan oleh menunggu lama dalam `IO:XactSync` acara pada peserta utama.

Tindakan

Kurangi jumlah transaksi yang memodifikasi data pada beberapa node. Selidiki `IO:XactSync` peristiwa di node lain dari cluster.

AuroraLimitless: DistributedCommitPrepareThrottle tunggu acara

Proses ini mencoba untuk mempersiapkan transaksi terdistribusi dan sedang dibatasi karena transaksi yang sudah disiapkan.

Penyebab

Transaksi yang memodifikasi beberapa node harus melakukan komit terdistribusi. Peserta dalam transaksi ini harus melakukan operasi persiapan sebagai bagian dari protokol komit. Aurora membatasi jumlah persiapan bersamaan, dan jika batas ini terlampaui proses akan menunggu dalam acara tersebut. `DistributedCommitPrepareThrottle`

Tindakan

Kurangi jumlah transaksi yang memodifikasi data pada beberapa node. Selidiki `IO:TwophaseFilePoolWrite` peristiwa karena peningkatan waktu dalam peristiwa tersebut dapat menyebabkan transaksi siap pakai yang ada menumpuk, yang mengakibatkan pelambatan untuk upaya persiapan baru.

AuroraLimitless: PreparedTransactionResolution tunggu acara

Proses ini telah mengalami tuple yang dimodifikasi oleh transaksi terdistribusi yang berada dalam keadaan siap. Proses harus menentukan apakah transaksi yang didistribusikan akan terlihat dalam snapshot-nya.

Penyebab

Transaksi yang memodifikasi beberapa node harus melakukan komit terdistribusi yang mencakup fase persiapan. Tingginya jumlah transaksi terdistribusi atau peningkatan latensi pada komit terdistribusi dapat menyebabkan proses lain menghadapi peristiwa `PreparedTransactionResolution` tunggu.

Tindakan

Kurangi jumlah transaksi yang memodifikasi data pada beberapa node. Selidiki peristiwa terkait komit terdistribusi karena peningkatan waktu dalam peristiwa tersebut dapat meningkatkan latensi di jalur komit transaksi terdistribusi. Anda mungkin juga ingin menyelidiki beban jaringan dan CPU.

AuroraLimitless: SendPreparedTransactionOutcome tunggu acara

Proses ini mengeksekusi pada node yang mengoordinasikan transaksi terdistribusi dan proses lain telah menanyakan keadaan transaksi itu, atau proses telah melakukan transaksi terdistribusi dan mengirimkan hasilnya kepada peserta.

Penyebab

Proses yang menghadapi peristiwa `PreparedTransactionResolution` tunggu akan menanyakan koordinator transaksi. Respons pada koordinator transaksi akan SendPreparedTransactionOutcome ecounter.

Tindakan

Kurangi jumlah transaksi yang memodifikasi data pada beberapa node. Selidiki peristiwa `IO:TwophaseFilePoolWrite` dan `IO:TwophaseFilePoolRead` peristiwa terkait komit terdistribusi karena peristiwa tersebut dapat meningkatkan latensi di jalur komit transaksi terdistribusi. Anda mungkin juga ingin menyelidiki beban jaringan dan CPU.

AuroraLimitless: CommitClockBarrier tunggu acara

Proses ini melakukan transaksi dan harus menunggu untuk memastikan bahwa waktu komit yang ditetapkan dijamin berada di masa lalu untuk semua node di cluster.

Penyebab

Staturasi CPU atau jaringan dapat menyebabkan peningkatan penyimpangan jam, yang mengakibatkan waktu yang dihabiskan dalam acara tunggu ini.

Tindakan

Selidiki CPU atau saturasi jaringan di cluster Anda.

AuroraLimitless: SnapshotClockBarrier tunggu acara

Proses ini telah menerima waktu snapshot dari node lain dengan jam di masa depan dan sedang menunggu jamnya sendiri untuk mencapai waktu itu.

Penyebab

Ini biasanya terjadi setelah proses menerima hasil dari fungsi yang didorong ke pecahan dan ada penyimpangan jam di antara node. Staturasi CPU atau jaringan dapat menyebabkan peningkatan penyimpangan jam, yang mengakibatkan waktu yang dihabiskan dalam acara tunggu ini.

Tindakan

Selidiki staturasi CPU atau jaringan di cluster Anda.

AuroraLimitless: ReaderSnapshotClockBarrier tunggu acara

Peristiwa ini terjadi pada node baca. Proses ini menunggu node baca untuk memutar ulang aliran tulis sehingga semua penulisan yang terjadi sebelum waktu snapshot proses telah diterapkan.

Penyebab

Peningkatan lag replika Aurora dapat menyebabkan peningkatan waktu tunggu dalam acara ini.

Tindakan

Selidiki lag replika Aurora.

AuroraLimitless: GatherDistributedDeadlockGraph tunggu acara

Prosesnya berkomunikasi dengan node lain untuk mengumpulkan grafik kunci sebagai bagian dari deteksi kebuntuan terdistribusi.

Penyebab

Ketika suatu proses menunggu di kunci, ia akan melakukan pemeriksaan kebuntuan terdistribusi setelah menunggu lebih lama dari`rds_aurora.limitless_distributed_deadlock_timeout`.

Tindakan

Selidiki penyebab pertengkaran kunci dalam aplikasi Anda dan pertimbangkan penyetelan`rds_aurora.limitless_distributed_deadlock_timeout`.

AuroraLimitless: DistributedDeadlockDetection tunggu acara

Prosesnya berkomunikasi dengan node lain untuk mendeteksi kebuntuan terdistribusi.

Penyebab

Ketika suatu proses menunggu di kunci, ia akan melakukan pemeriksaan kebuntuan terdistribusi setelah menunggu lebih lama dari`rds_aurora.limitless_distributed_deadlock_timeout`.

Tindakan

Selidiki penyebab pertengkaran kunci dalam aplikasi Anda dan pertimbangkan penyetelan`rds_aurora.limitless_distributed_deadlock_timeout`.

AuroraLimitless: DistributedDeadlockAbort tunggu acara

Prosesnya berkomunikasi dengan node lain untuk membatalkan sesi yang dipilih sebagai korban dalam kebuntuan terdistribusi.

Penyebab

Pola aplikasi menghasilkan kebuntuan terdistribusi.

Tindakan

Selidiki pola aplikasi yang mengakibatkan kebuntuan terdistribusi.

AuroraLimitless: GatherRemoteStats tunggu acara

Prosesnya adalah mengumpulkan statistik dari node lain di cluster.

Penyebab

Monitoring atau activty query dan views, seperti`limitless_stat_activity`, akan mengambil statistik dari node lain.

Tindakan

Tidak diperlukan tindakan lebih lanjut.

AuroraLimitless: GlobalSequenceRefresh tunggu acara

Proses ini menghasilkan nilai urutan baru dan harus meminta potongan baru dari urutan global.

Penyebab

Tingkat pembuatan nilai urutan yang tinggi dapat mengakibatkan penghentian dalam acara ini jika tidak `rds_aurora.limitless_sequence_chunk_size` mencukupi.

Tindakan

Ini adalah kejadian normal. Jika Anda melihat waktu yang berlebihan dalam acara ini pertimbangkan penyetelan. `rds_aurora.limitless_sequence_chunk_size` Lihat dokumentasi tentang urutan di Limitless Database.

AuroraLimitless: GlobalVacuumTimeExchange tunggu acara

Prosesnya bertukar data snapshot untuk mendukung kekosongan.

Penyebab

Node di Limitless Database menukar data waktu snapshot aktif tertua dengan node lain untuk menghitung waktu cutoff yang benar untuk eksekusi vakum.

Tindakan

Tidak diperlukan tindakan lebih lanjut.

AuroraLimitless: DistributedTransactionMonitorGather tunggu acara

Prosesnya adalah mengumpulkan metadata transaksi dari node lain untuk mendukung pembersihan transaksi terdistribusi.

Penyebab

Node dalam metadata transaksi pertukaran Database Limitless dengan node lain untuk menentukan kapan status transaksi terdistribusi dapat dibersihkan.

Tindakan

Tidak diperlukan tindakan lebih lanjut.

AuroraLimitlessActivity: AdminTaskSchedulerMain tunggu acara

Menunggu di loop utama proses penjadwal tugas.

AuroraLimitlessActivity: AdminTaskExecutorMain tunggu acara

Menunggu di loop utama proses pelaksana tugas.

AuroraLimitlessActivity: AdminTaskMonitorMain tunggu acara

Menunggu di loop utama proses monitor tugas.

AuroraLimitlessActivity: DatabaseCleanupMonitorMain tunggu acara

Menunggu di loop utama proses monitor pembersihan basis data.

AuroraLimitlessActivity: TopologyCleanupMonitorMain tunggu acara

Menunggu di loop utama proses monitor pembersihan topologi.

AuroraLimitlessActivity: ToplogyChangeMonitorMain tunggu acara

Menunggu di loop utama proses monitor perubahan topologi.

AuroraLimitlessActivity: DistributedTransactionMonitorMain tunggu acara

Menunggu di loop utama proses monitor transaksi terdistribusi.

AuroraLimitlessActivity: GlobalVacuumMonitorMain tunggu acara

Menunggu di loop utama proses monitor vakum global.

# Mencadangkan dan memulihkan Database Tanpa Batas Aurora Postgre SQL

Anda dapat mencadangkan dan memulihkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database.

Daftar Isi


- Mencadangkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database
  - Membuat snapshot klaster DB
- Memulihkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database
  - Memulihkan cluster DB dari snapshot DB
  - Memulihkan cluster DB menggunakan point-in-time pemulihan
- Postgre SQL backup dan restore utilitas tidak didukung


## Mencadangkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database

Mencadangkan cluster DB dengan Aurora SQL Postgre Limitless Database memiliki persamaan dan perbedaan fungsionalitas dibandingkan dengan mencadangkan cluster Aurora DB standar.

- Saat Anda mengambil snapshot cluster DB manual dari cluster Aurora DB yang menggunakan Limitless Database, snapshot tersebut menyertakan data dari grup pecahan DB.
- Pencadangan berkelanjutan mencakup data dari grup pecahan DB.
- Snapshot harian otomatis menyertakan data dari grup pecahan DB.
- Menyalin snapshot cluster DB didukung. Untuk informasi selengkapnya, lihat Penyalinan snapshot cluster DB.
- Berbagi snapshot cluster DB didukung. Untuk informasi selengkapnya, lihat Berbagi snapshot klaster DB.
- Anda tidak dapat menggunakan `pg_dumpall` utilitas `pg_dump` atau untuk membuat cadangan database di grup pecahan DB.
- Mengambil snapshot akhir saat menghapus cluster DB didukung untuk Aurora Postgre Limitless Database. SQL

- Mempertahankan backup otomatis saat menghapus cluster DB tidak didukung untuk Aurora Postgre Limitless Database. SQL

## Membuat snapshot klaster DB

Anda membuat snapshot cluster DB Aurora Postgre SQL Limitless Database dengan cara yang sama seperti untuk cluster Aurora DB standar, seperti yang ditunjukkan pada contoh berikut: AWS CLI

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier my-db-cluster \
    --db-cluster-snapshot-identifier my-db-cluster-snapshot
```

Untuk informasi lebih lanjut tentang mencadangkan cluster DB, lihat. Gambaran umum pencadangan dan pemulihan klaster DB Aurora

# Memulihkan cluster DB yang menggunakan Aurora SQL Postgre Limitless Database

Memulihkan cluster DB dengan Aurora SQL Postgre Limitless Database memiliki persamaan dan perbedaan fungsionalitas dibandingkan dengan memulihkan cluster Aurora DB standar.

- Anda dapat memulihkan cluster DB Limitless Database hanya dari cluster DB sumber yang menggunakan versi mesin DB yang kompatibel dengan Limitless Database, seperti. `16.4-limitless`
- Saat Anda memulihkan cluster DB dari snapshot manual cluster DB yang menggunakan Limitless Database, seluruh penyimpanan cluster DB dipulihkan. Ini termasuk penyimpanan grup pecahan DB.

  Anda harus membuat grup pecahan DB untuk mengakses penyimpanan untuk Database Limitless Anda.
- Anda dapat memulihkan cluster DB menggunakan point-in-time recovery (PITR) ke titik mana pun dalam periode retensi. Cluster DB yang dipulihkan mencakup penyimpanan grup pecahan DB.

  Anda harus membuat grup pecahan DB untuk mengakses penyimpanan untuk Database Limitless Anda.
- PITRtidak didukung untuk klaster DB Database Aurora Postgre SQL Limitless yang dihapus.
- Saat Anda memulihkan cluster DB dari snapshot harian otomatis, penyimpanan untuk grup shard DB juga dipulihkan.
- Saat memulihkan cluster DB Basis Data Aurora Postgre SQL Limitless, Anda harus mengaktifkan Enhanced Monitoring and Performance Insights. Pastikan untuk menyertakan ID KMS kunci Performance Insights.

Setelah Anda memulihkan cluster DB Basis Data Aurora Postgre SQL Limitless, pastikan untuk memverifikasi fungsinya dengan menjalankan kueri Anda di dalamnya.

## Memulihkan cluster DB dari snapshot DB

AWS CLI Contoh berikut menunjukkan cara mengembalikan cluster DB Database Aurora Postgre SQL Limitless dari snapshot cluster DB.

Anda harus menggunakan versi mesin `16.4-limitless` DB.

Untuk memulihkan cluster DB Limitless Database dari snapshot cluster DB

1. Kembalikan cluster DB:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-db-cluster \
    --snapshot-identifier my-db-cluster-snapshot \
    --engine aurora-postgresql \
    --engine-version 16.4-limitless \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. Buat grup pecahan DB:

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   Untuk informasi selengkapnya, lihat Menambahkan grup pecahan DB ke cluster DB Basis Data Aurora SQL Postgre Limitless yang ada.

Untuk informasi lebih lanjut tentang memulihkan cluster Aurora DB dari snapshot cluster DB, lihat. Memulihkan dari snapshot klaster DB

## Memulihkan cluster DB menggunakan point-in-time pemulihan

AWS CLI Contoh berikut menunjukkan cara mengembalikan cluster DB Database Aurora Postgre SQL Limitless menggunakan recovery (). point-in-time PITR

Untuk memulihkan cluster DB Limitless Database menggunakan PITR

1. Kembalikan cluster DB:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-db-cluster \
    --db-cluster-identifier my-new-db-cluster \
    --use-latest-restorable-time \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. Buat grup pecahan DB:

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   Untuk informasi selengkapnya, lihat Menambahkan grup pecahan DB ke cluster DB Basis Data Aurora SQL Postgre Limitless yang ada.

Untuk informasi selengkapnya tentang PITR, lihat Memulihkan klaster DB ke waktu tertentu.

# Postgre SQL backup dan restore utilitas tidak didukung

SQLUtilitas Postgre berikut tidak didukung untuk cluster DB primer atau grup shard DB:

- `pg_dump`
- `pg_dumpall`
- `pg_restore`

Meskipun Anda mungkin dapat menggunakannya dengan binari open source atau metode alternatif, hal itu dapat menghasilkan hasil yang tidak konsisten.

# Meningkatkan Database Tanpa Batas Amazon Aurora PostgreSQL

Berikut ini berlaku untuk upgrade Aurora PostgreSQL Limitless Database:

- Upgrade versi minor didukung.
- Menambal Aurora PostgreSQL Limitless Database didukung. Patch muncul sebagai pemeliharaan yang tertunda untuk diterapkan selama jendela pemeliharaan Anda.

## Memutakhirkan cluster DB yang menggunakan Amazon Aurora PostgreSQL Limitless Database

Anda meningkatkan cluster DB dengan memodifikasinya dan memilih versi mesin DB baru. Anda dapat menggunakan AWS Management Console atau AWS CLI.

Konsol

Untuk meng-upgrade cluster DB Limitless Database Anda

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Arahkan ke halaman Basis data.
3. Pilih cluster DB Limitless Database yang ingin Anda upgrade.
4. Pilih Ubah.

   Halaman cluster Modify DB ditampilkan.
5. Untuk versi mesin DB, pilih versi mesin DB baru, misalnya Aurora PostgreSQL dengan Limitless Database (Kompatibel dengan PostgreSQL 16.6).
6. Pilih Lanjutkan.
7. Pada halaman ringkasan, pilih apakah akan segera menerapkan perubahan atau selama jendela pemeliharaan terjadwal berikutnya, lalu pilih Ubah cluster.

CLI

Gunakan perintah modify-db-cluster AWS CLI seperti pada contoh berikut.

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-sv2-cluster \
    --engine-version 16.6-limitless \
    --apply-immediately
```

# Referensi Basis Data Tanpa Batas Aurora Postgre SQL

Kami menyediakan topik referensi berikut untuk Aurora SQL Postgre Limitless Database.

Topik


- Perintah Bahasa Definisi Data (DDL) SQL yang didukung dan tidak didukung
- Keterbatasan DDL dan informasi lainnya untuk Aurora PostgreSQL Limitless Database
- Didukung dan tidak didukung Data Manipulation Language (DHTML) dan perintah query processing SQL
- DMLketerbatasan dan informasi lainnya untuk Aurora SQL Postgre Limitless Database
- Variabel dalam Basis Data Tanpa Batas Aurora Postgre SQL
- Parameter cluster DB di Aurora SQL Postgre Limitless Database


## Perintah Bahasa Definisi Data (DDL) SQL yang didukung dan tidak didukung

Tabel berikut mencantumkan DDL perintah yang didukung dan tidak didukung oleh Aurora Postgre SQL Limitless Database, dengan referensi ke batasan atau informasi lebih lanjut.

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| ALTER AGGREGATE | Tidak | Tidak berlaku |
| ALTER COLLATION | Ya | Tidak ada |
| ALTER CONVERSION | Ya | Tidak ada |
| ALTER DATABASE | Tidak | Tidak berlaku |
| ALTER DEFAULT PRIVILEGES | Tidak | Tidak berlaku |
| ALTER DOMAIN | Tidak | Tidak berlaku |
| ALTER EVENT TRIGGER | Tidak | Tidak berlaku |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| ALTER EXTENSION | Ya | Ekstensi |
| ALTER FOREIGN DATA WRAPPER | Tidak | Tidak berlaku |
| ALTER FOREIGN TABLE | Tidak | Tidak berlaku |
| ALTER FUNCTION | Ya | Fungsi |
| ALTER GROUP | Ya | Tidak ada |
| ALTER INDEX | Ya | Tidak ada |
| ALTER LANGUAGE | Tidak | Tidak berlaku |
| ALTER LARGE OBJECT | Tidak | Tidak berlaku |
| ALTER MATERIALIZED VIEW | Tidak | Tidak berlaku |
| ALTER OPERATOR | Ya | Tidak ada |
| ALTER OPERATOR CLASS | Ya | Tidak ada |
| ALTER OPERATOR FAMILY | Ya | Tidak ada |
| ALTER POLICY | Tidak | Tidak berlaku |
| ALTER PROCEDURE | Ya | Tidak ada |
| ALTER PUBLICATION | Tidak | Tidak berlaku |
| ALTER ROLE | Ya | Tidak ada |
| ALTER ROUTINE | Tidak | Tidak berlaku |
| ALTER RULE | Tidak | Tidak berlaku |
| ALTER SCHEMA | Ya | Tidak ada |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
| --- | --- | --- |
| ALTER SEQUENCE | Ya | Urutan |
| ALTER SERVER | Tidak | Tidak berlaku |
| ALTER STATISTICS | Tidak | Tidak berlaku |
| ALTER SUBSCRIPTION | Tidak | Tidak berlaku |
| ALTER SYSTEM | Tidak | Tidak berlaku |
| ALTER TABLE | Ya | ALTER TABLE |
| ALTER TABLESPACE | Tidak | Tidak berlaku |
| ALTER TEXT SEARCH CONFIGURATION | Tidak | Tidak berlaku |
| ALTER TEXT SEARCH DICTIONARY | Tidak | Tidak berlaku |
| ALTER TEXT SEARCH PARSER | Tidak | Tidak berlaku |
| ALTER TEXT SEARCH TEMPLATE | Tidak | Tidak berlaku |
| ALTER TRIGGER | Tidak | Tidak berlaku |
| ALTER TYPE | Ya | Tidak ada |
| ALTER USER | Ya | Tidak ada |
| ALTER USER MAPPING | Tidak | Tidak berlaku |
| ALTER VIEW | Ya | Tidak ada |
| COMMENT | Tidak | Tidak berlaku |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| CREATE ACCESS METHOD | Tidak | Tidak berlaku |
| CREATE AGGREGATE | Tidak | Tidak berlaku |
| CREATE CAST | Ya | Tidak ada |
| CREATE COLLATION | Ya | Tidak ada |
| CREATE CONVERSION | Ya | Tidak ada |
| CREATE DATABASE | Ya | BUAT BASIS DATA |
| CREATE DOMAIN | Tidak | Tidak berlaku |
| CREATE EVENT TRIGGER | Tidak | Tidak berlaku |
| CREATE EXTENSION | Ya | Ekstensi |
| CREATE FOREIGN DATA WRAPPER | Tidak | Tidak berlaku |
| CREATE FOREIGN TABLE | Tidak | Tidak berlaku |
| CREATE FUNCTION | Ya | Fungsi |
| CREATE GROUP | Ya | Tidak ada |
| CREATE INDEX | Ya | CREATE INDEX |
| CREATE LANGUAGE | Tidak | Tidak berlaku |
| CREATE MATERIALIZED VIEW | Tidak | Tidak berlaku |
| CREATE OPERATOR | Ya | Tidak ada |
| CREATE OPERATOR CLASS | Ya | Tidak ada |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| CREATE OPERATOR FAMILY | Ya | Tidak ada |
| CREATE POLICY | Ya | Tidak ada |
| CREATE PROCEDURE | Ya | Tidak ada |
| CREATE PUBLICATION | Tidak | Tidak berlaku |
| CREATE ROLE | Ya | Tidak ada |
| CREATE RULE | Tidak | Tidak berlaku |
| CREATE SCHEMA | Ya | BUAT SKEMA |
| CREATE SEQUENCE | Ya | Urutan |
| CREATE SERVER | Tidak | Tidak berlaku |
| CREATE STATISTICS | Tidak | Tidak berlaku |
| CREATE SUBSCRIPTION | Tidak | Tidak berlaku |
| CREATE TABLE | Ya | CREATE TABLE |
| CREATETABLESEBAGAI | Ya | BUAT TABEL SEBAGAI |
| CREATE TABLESPACE | Tidak | Tidak berlaku |
| CREATE TEMPORARY TABLE | Tidak | Tidak berlaku |
| CREATETEMPORARYTABLESEBAGAI | Tidak | Tidak berlaku |
| CREATE TEXT SEARCH CONFIGURATION | Tidak | Tidak berlaku |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| CREATE TEXT SEARCH DICTIONARY | Tidak | Tidak berlaku |
| CREATE TEXT SEARCH PARSER | Tidak | Tidak berlaku |
| CREATE TEXT SEARCH TEMPLATE | Tidak | Tidak berlaku |
| CREATE TRANSFORM | Tidak | Tidak berlaku |
| CREATE TRIGGER | Tidak | Tidak berlaku |
| CREATE TYPE | Ya | Tidak ada |
| CREATE USER | Ya | Tidak ada |
| CREATE USER MAPPING | Tidak | Tidak berlaku |
| CREATE VIEW | Ya | Tidak ada |
| DROP ACCESS METHOD | Tidak | Tidak berlaku |
| DROP AGGREGATE | Ya | Tidak ada |
| DROP CAST | Ya | Tidak ada |
| DROP COLLATION | Ya | Tidak ada |
| DROP CONVERSION | Ya | Tidak ada |
| DROP DATABASE | Ya | DROP DATABASE |
| DROP DOMAIN | Tidak | Tidak berlaku |
| DROP EVENT TRIGGER | Tidak | Tidak berlaku |
| DROP EXTENSION | Ya | Ekstensi |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| DROP FOREIGN DATA WRAPPER | Tidak | Tidak berlaku |
| DROP FOREIGN TABLE | Tidak | Tidak berlaku |
| DROP FUNCTION | Ya | Fungsi |
| DROP GROUP | Ya | Tidak ada |
| DROP INDEX | Ya | Tidak ada |
| DROP LANGUAGE | Tidak | Tidak berlaku |
| DROP MATERIALIZED VIEW | Tidak | Tidak berlaku |
| DROP OPERATOR | Ya | Tidak ada |
| DROP OPERATOR CLASS | Ya | Tidak ada |
| DROP OPERATOR FAMILY | Ya | Tidak ada |
| DROP OWNED | Tidak | Tidak berlaku |
| DROP POLICY | Tidak | Tidak berlaku |
| DROP PROCEDURE | Ya | Tidak ada |
| DROP PUBLICATION | Tidak | Tidak berlaku |
| DROP ROLE | Ya | Tidak ada |
| DROP ROUTINE | Tidak | Tidak berlaku |
| DROP RULE | Tidak | Tidak berlaku |
| DROP SCHEMA | Ya | Tidak ada |
| DROP SEQUENCE | Ya | Tidak ada |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
| --- | --- | --- |
| DROP SERVER | Tidak | Tidak berlaku |
| DROP STATISTICS | Tidak | Tidak berlaku |
| DROP SUBSCRIPTION | Tidak | Tidak ada |
| DROP TABLE | Ya | Tidak ada |
| DROP TABLESPACE | Tidak | Tidak berlaku |
| DROP TEXT SEARCH CONFIGURATION | Tidak | Tidak berlaku |
| DROP TEXT SEARCH DICTIONARY | Tidak | Tidak berlaku |
| DROP TEXT SEARCH PARSER | Tidak | Tidak berlaku |
| DROP TEXT SEARCH TEMPLATE | Tidak | Tidak berlaku |
| DROP TRANSFORM | Tidak | Tidak berlaku |
| DROP TRIGGER | Tidak | Tidak berlaku |
| DROP TYPE | Ya | Tidak ada |
| DROP USER | Ya | Tidak ada |
| DROP USER MAPPING | Tidak | Tidak berlaku |
| DROP VIEW | Ya | Tidak ada |
| GRANT | Ya | Tidak ada |
| REASSIGN OWNED | Tidak | Tidak berlaku |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| REVOKE | Ya | Tidak ada |
| SECURITY LABEL | Tidak | Tidak berlaku |
| SELECT INTO | Ya | PILIH KE |
| SET | Ya | Tidak ada |
| SET CONSTRAINTS | Tidak | Tidak berlaku |
| SET ROLE | Ya | Tidak ada |
| SET SESSION AUTHORIZATION | Ya | Tidak ada |
| SET TRANSACTION | Ya | Tidak ada |
| TRUNCATE | Ya | Tidak ada |

# Keterbatasan DDL dan informasi lainnya untuk Aurora PostgreSQL Limitless Database

Topik berikut menjelaskan batasan atau memberikan informasi lebih lanjut untuk perintah DDL SQL di Aurora PostgreSQL Limitless Database.

Topik


- ALTER TABLE
- BUAT BASIS DATA
- CREATE INDEX
- BUAT SKEMA
- CREATE TABLE
- BUAT TABEL SEBAGAI
- DROP DATABASE
- PILIH KE
- Batasan
- Nilai default
- Ekstensi
- Kunci asing
- Fungsi
- Urutan

## ALTER TABLE

`ALTER TABLE`Perintah ini umumnya didukung di Aurora PostgreSQL Limitless Database. Untuk informasi selengkapnya, lihat ALTER TABLE dalam dokumentasi PostgreSQL.

### Batasan

`ALTER TABLE`memiliki batasan berikut untuk opsi yang didukung.

Menghapus kolom

- Pada tabel sharded, Anda tidak dapat menghapus kolom yang merupakan bagian dari kunci shard.
- Pada tabel referensi, Anda tidak dapat menghapus kolom kunci utama.

Mengubah tipe data kolom

- `USING`Ekspresi tidak didukung.
- Pada tabel sharded, Anda tidak dapat mengubah jenis kolom yang merupakan bagian dari kunci shard.

Menambahkan atau menghapus kendala

Untuk detail tentang apa yang tidak didukung, lihatBatasan.

Mengubah nilai default kolom

Nilai default didukung. Untuk informasi selengkapnya, lihat Nilai default.

### Opsi yang tidak didukung

Beberapa opsi tidak didukung karena bergantung pada fitur yang tidak didukung, seperti pemicu.

Opsi tingkat tabel berikut untuk `ALTER TABLE` tidak didukung:

- `ALL IN TABLESPACE`
- `ATTACH PARTITION`
- `DETACH PARTITION`
- `ONLY`bendera
- `RENAME CONSTRAINT`

Opsi tingkat kolom berikut untuk `ALTER TABLE` tidak didukung:

- TAMBAHKAN YANG DIHASILKAN
- EKSPRESI DROP [JIKA ADA]
- JATUHKAN IDENTITAS [JIKA ADA]
- ATUR ULANG
- MENGULANG KEMBALI
- SET
- MENGATUR KOMPRESI
- SET STATISTICS

## BUAT BASIS DATA

Di Aurora PostgreSQL Limitless Database, hanya database tak terbatas yang didukung.

Saat `CREATE DATABASE` sedang berjalan, database yang berhasil dibuat dalam satu atau lebih node mungkin gagal di node lain, karena pembuatan database adalah operasi nontransaksional. Dalam hal ini, objek database yang berhasil dibuat secara otomatis dihapus dari semua node dalam jumlah waktu yang telah ditentukan untuk menjaga konsistensi dalam grup shard DB. Selama waktu ini, pembuatan ulang database dengan nama yang sama dapat mengakibatkan kesalahan yang menunjukkan bahwa database sudah ada.

Opsi berikut didukung:

- Pemeriksaan:

```
CREATE DATABASE name WITH
    [LOCALE = locale]
    [LC_COLLATE = lc_collate]
    [LC_CTYPE = lc_ctype]
    [ICU_LOCALE = icu_locale]
    [ICU_RULES = icu_rules]
    [LOCALE_PROVIDER = locale_provider]
    [COLLATION_VERSION = collation_version];
```

- `CREATE DATABASE WITH OWNER`:

```
CREATE DATABASE name WITH OWNER = user_name;
```

Opsi berikut tidak didukung:

- `CREATE DATABASE WITH TABLESPACE`:

```
CREATE DATABASE name WITH TABLESPACE = tablespace_name;
```

- `CREATE DATABASE WITH TEMPLATE`:

```
CREATE DATABASE name WITH TEMPLATE = template;
```

## CREATE INDEX

`CREATE INDEX CONCURRENTLY`didukung untuk tabel sharded:

```
CREATE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

`CREATE UNIQUE INDEX`didukung untuk semua jenis tabel:

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

`CREATE UNIQUE INDEX CONCURRENTLY`tidak didukung:

```
CREATE UNIQUE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

Untuk informasi selengkapnya, lihat UNIK. Untuk informasi umum tentang membuat indeks, lihat CREATE INDEX dalam dokumentasi PostgreSQL.

Menampilkan indeks

Tidak semua indeks terlihat pada router saat Anda menggunakan `\d table_name` atau perintah serupa. Sebagai gantinya, gunakan `pg_catalog.pg_indexes` tampilan untuk mendapatkan indeks, seperti yang ditunjukkan pada contoh berikut.

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items (id int PRIMARY KEY, val int);
CREATE INDEX items_my_index on items (id, val);

postgres_limitless=> SELECT * FROM pg_catalog.pg_indexes WHERE tablename='items';
```

```
 schemaname | tablename |   indexname    | tablespace |
   indexdef
------------+-----------+----------------+------------
+-------------------------------------------------------------------------
 public     | items     | items_my_index |            | CREATE INDEX items_my_index
 ON ONLY public.items USING btree (id, val)
 public     | items     | items_pkey     |            | CREATE UNIQUE INDEX
 items_pkey ON ONLY public.items USING btree (id)
(2 rows)
```

## BUAT SKEMA

`CREATE SCHEMA`dengan elemen skema tidak didukung:

```
CREATE SCHEMA my_schema CREATE TABLE (column_name INT);
```

Ini menghasilkan kesalahan yang mirip dengan yang berikut:

```
ERROR: CREATE SCHEMA with schema elements is not supported
```

## CREATE TABLE

Relasi dalam `CREATE TABLE` pernyataan tidak didukung, misalnya:

```
CREATE TABLE orders (orderid int, customerId int, orderDate date) WITH
 (autovacuum_enabled = false);
```

`IDENTITY`kolom tidak didukung, misalnya:

```
CREATE TABLE orders (orderid INT GENERATED ALWAYS AS IDENTITY);
```

Aurora PostgreSQL Limitless Database mendukung hingga 54 karakter untuk nama tabel sharded.

## BUAT TABEL SEBAGAI

Untuk membuat tabel menggunakan`CREATE TABLE AS`, Anda harus menggunakan `rds_aurora.limitless_create_table_mode` variabel. Untuk tabel sharded, Anda juga harus menggunakan `rds_aurora.limitless_create_table_shard_key` variabel. Untuk informasi selengkapnya, lihat Membuat tabel tanpa batas dengan menggunakan variabel.

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

CREATE TABLE ctas_table AS SELECT 1 a;

-- "source" is the source table whose columns and data types are used to create the new
 "ctas_table2" table.
CREATE TABLE ctas_table2 AS SELECT a,b FROM source;
```

Anda tidak dapat menggunakan `CREATE TABLE AS` untuk membuat tabel referensi, karena mereka memerlukan kendala kunci primer. `CREATE TABLE AS`tidak menyebarkan kunci utama ke tabel baru.

Untuk informasi umum, lihat MEMBUAT TABEL AS dalam dokumentasi PostgreSQL.

## DROP DATABASE

Anda dapat menjatuhkan database yang telah Anda buat.

`DROP DATABASE`Perintah berjalan secara asinkron di latar belakang. Saat sedang berjalan, Anda akan menerima kesalahan jika Anda mencoba membuat database baru dengan nama yang sama.

## PILIH KE

`SELECT INTO`secara fungsional mirip dengan. BUAT TABEL SEBAGAI Anda harus menggunakan `rds_aurora.limitless_create_table_mode` variabel. Untuk tabel sharded, Anda juga harus menggunakan `rds_aurora.limitless_create_table_shard_key` variabel. Untuk informasi selengkapnya, lihat Membuat tabel tanpa batas dengan menggunakan variabel.

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

-- "source" is the source table whose columns and data types are used to create the new
 "destination" table.
SELECT * INTO destination FROM source;
```

Saat ini, `SELECT INTO` operasi dilakukan melalui router, tidak langsung melalui pecahan. Karena itu, kinerjanya bisa lambat.

Untuk informasi umum, lihat SELECT INTO dalam dokumentasi PostgreSQL.

## Batasan

Batasan berikut berlaku untuk kendala di Aurora PostgreSQL Limitless Database.

### MEMERIKSA

Kendala sederhana yang melibatkan operator perbandingan dengan literal didukung. Ekspresi dan batasan yang lebih kompleks yang memerlukan validasi fungsi tidak didukung, seperti yang ditunjukkan pada contoh berikut.

```
CREATE TABLE my_table (
    id  INT CHECK (id > 0)                                        -- supported
  , val INT CHECK (val > 0 AND val < 1000)                        -- supported
  , tag TEXT CHECK (length(tag) > 0)                              -- not supported:
 throws "Expression inside CHECK constraint is not supported"
  , op_date TIMESTAMP WITH TIME ZONE CHECK (op_date <= now())  -- not supported:
 throws "Expression inside CHECK constraint is not supported"
);
```

Anda dapat memberikan batasan nama eksplisit, seperti yang ditunjukkan pada contoh berikut.

```
CREATE TABLE my_table (
    id  INT CONSTRAINT positive_id  CHECK (id > 0)
  , val INT CONSTRAINT val_in_range CHECK (val > 0 AND val < 1000)
);
```

Anda dapat menggunakan sintaks kendala tingkat tabel dengan kendala, seperti yang `CHECK` ditunjukkan pada contoh berikut.

```
CREATE TABLE my_table (
    id INT CONSTRAINT positive_id  CHECK (id > 0)
  , min_val INT CONSTRAINT min_val_in_range CHECK (min_val > 0 AND min_val < 1000)
  , max_val INT
  , CONSTRAINT max_val_in_range CHECK (max_val > 0 AND max_val < 1000 AND max_val >
 min_val)
);
```

### MENGECUALIKAN

Batasan pengecualian tidak didukung di Aurora PostgreSQL Limitless Database.

KUNCI ASING

Untuk informasi selengkapnya, lihat Kunci asing.

TIDAK NULL

`NOT NULL`kendala didukung tanpa batasan.

KUNCI UTAMA

Kunci primer menyiratkan kendala unik dan oleh karena itu pembatasan yang sama pada kendala unik berlaku pada kunci primer. Ini berarti:

- Jika tabel diubah menjadi tabel sharded, kunci shard harus merupakan bagian dari kunci utama. Artinya, kunci utama berisi semua kolom kunci shard.
- Jika tabel diubah menjadi tabel referensi, itu harus memiliki kunci utama.

Contoh berikut menggambarkan penggunaan kunci primer.

```
-- Create a standard table.
CREATE TABLE public.my_table (
    item_id INT
  , location_code INT
  , val INT
  , comment text
);

-- Change the table to a sharded table using the 'item_id' and 'location_code'
 columns as shard keys.
CALL rds_aurora.limitless_alter_table_type_sharded('public.my_table',
 ARRAY['item_id', 'location_code']);
```

Mencoba menambahkan kunci utama yang tidak berisi kunci pecahan:

```
-- Add column 'item_id' as the primary key.
-- Invalid because the primary key doesnt include all columns from the shard key:
-- 'location_code' is part of the shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR

-- add column "val" as primary key
-- Invalid because primary key does not include all columns from shard key:
--  item_id and location_code iare part of shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR
```

Mencoba menambahkan kunci utama yang memang berisi kunci pecahan:

```
-- Add the 'item_id' and 'location_code' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code); -- OK

-- Add the 'item_id', 'location_code', and 'val' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code, val); -- OK
```

Ubah tabel standar menjadi tabel referensi.

```
-- Create a standard table.
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);

-- Convert the table to a reference table.
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcode');
```

Untuk informasi selengkapnya tentang membuat tabel sharded dan referensi, lihatMembuat tabel Basis Data Tanpa Batas Aurora Postgre SQL.

UNIK

Dalam tabel sharded, kunci unik harus berisi kunci shard, yaitu kunci shard harus merupakan bagian dari kunci unik. Ini dicentang saat mengubah jenis tabel menjadi sharded. Dalam tabel referensi tidak ada batasan.

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT UNIQUE
);
```

`UNIQUE`Kendala tingkat tabel didukung, seperti yang ditunjukkan pada contoh berikut.

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT
  , CONSTRAINT zipcode_and_email UNIQUE (zipcode, email)
);
```

Contoh berikut menunjukkan penggunaan kunci utama dan kunci unik bersama-sama. Kedua kunci harus menyertakan kunci shard.

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"p_id"}';
CREATE TABLE t1 (
p_id BIGINT NOT NULL,
c_id BIGINT NOT NULL,
PRIMARY KEY (p_id),
UNIQUE (p_id, c_id)
);
```

Untuk informasi selengkapnya, lihat Batasan dalam dokumentasi PostgreSQL.

## Nilai default

Aurora PostgreSQL Limitless Database mendukung ekspresi dalam nilai default.

Contoh berikut menunjukkan penggunaan nilai default.

```
CREATE TABLE t (
    a INT DEFAULT 5,
    b TEXT DEFAULT 'NAN',
    c NUMERIC
);

CALL rds_aurora.limitless_alter_table_type_sharded('t', ARRAY['a']);
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c
---+-----+---
 5 | NAN |
(1 row)
```

Ekspresi didukung, seperti yang ditunjukkan pada contoh berikut.

```
CREATE TABLE t1 (a NUMERIC DEFAULT random());
```

Contoh berikut menambahkan kolom baru yang `NOT NULL` dan memiliki nilai default.

```
ALTER TABLE t ADD COLUMN d BOOLEAN NOT NULL DEFAULT FALSE;
SELECT * FROM t;

 a |  b  | c | d
---+-----+---+---
 5 | NAN |   | f
(1 row)
```

Contoh berikut mengubah kolom yang ada dengan nilai default.

```
ALTER TABLE t ALTER COLUMN c SET DEFAULT 0.0;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
(2 rows)
```

Contoh berikut menjatuhkan nilai default.

```
ALTER TABLE t ALTER COLUMN a DROP DEFAULT;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
   | NAN | 0.0 | f
(3 rows)
```

Untuk informasi selengkapnya, lihat Nilai default dalam dokumentasi PostgreSQL.

## Ekstensi

Ekstensi PostgreSQL berikut didukung di Aurora PostgreSQL Limitless Database:

- `aurora_limitless_fdw`- Ekstensi ini sudah diinstal sebelumnya. Anda tidak bisa menjatuhkannya.
- `aws_s3`- Ekstensi ini bekerja di Aurora PostgreSQL Limitless Database mirip dengan cara yang dilakukannya di Aurora PostgreSQL.

  Anda dapat mengimpor data dari bucket Amazon S3 ke cluster DB Database PostgreSQL Limitless Aurora, atau mengekspor data dari cluster DB Database PostgreSQL Limitless Aurora ke bucket Amazon S3. Untuk informasi selengkapnya, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL dan Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.
- `btree_gin`
- `citext`
- `ip4r`
- `pg_buffercache`— Ekstensi ini berperilaku berbeda di Aurora PostgreSQL Limitless Database dari PostgreSQL komunitas PostgreSQL. Untuk informasi selengkapnya, lihat perbedaan pg_buffercache di Aurora PostgreSQL Limitless Database.
- `pg_stat_statements`
- `pg_trgm`
- `pgcrypto`
- `pgstattuple`— Ekstensi ini berperilaku berbeda di Aurora PostgreSQL Limitless Database dari PostgreSQL komunitas PostgreSQL. Untuk informasi selengkapnya, lihat perbedaan pgstattuple di Aurora PostgreSQL Limitless Database.
- `pgvector`
- `plpgsql`— Ekstensi ini sudah diinstal sebelumnya, tetapi Anda dapat menjatuhkannya.
- `PostGIS`— Transaksi panjang dan fungsi manajemen tabel tidak didukung. Memodifikasi tabel referensi spasial tidak didukung.
- `unaccent`
- `uuid`

Sebagian besar ekstensi PostgreSQL saat ini tidak didukung di Aurora PostgreSQL Limitless Database. Namun, Anda masih dapat menggunakan pengaturan konfigurasi shared_preload_libraries

(SPL) untuk memuat ekstensi ke dalam cluster DB utama Aurora PostgreSQL. Mereka juga dimuat ke Aurora PostgreSQL Limitless Database, tetapi mungkin tidak berfungsi dengan benar.

Misalnya, Anda dapat memuat `pg_hint_plan` ekstensi, tetapi memuatnya tidak menjamin bahwa petunjuk yang diteruskan dalam komentar kueri digunakan.

> **Note**
>
> Anda tidak dapat memodifikasi objek yang terkait dengan ekstensi pg_stat_statement. Untuk informasi tentang penginstalan`pg_stat_statements`, lihat limitless_stat_statement.

Anda dapat menggunakan `pg_available_extensions` dan `pg_available_extension_versions` fungsi untuk menemukan ekstensi yang didukung di Aurora PostgreSQL Limitless Database.

DDLs Berikut ini didukung untuk ekstensi:

CREATE EXTENSION

Anda dapat membuat ekstensi, seperti di PostgreSQL.

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]
```

Untuk informasi selengkapnya, lihat CREATE EXTENSION di dokumentasi PostgreSQL.

ALTER EXTENSION

Berikut ini DDLs didukung:

```
ALTER EXTENSION name UPDATE [ TO new_version ]

ALTER EXTENSION name SET SCHEMA new_schema
```

Untuk informasi selengkapnya, lihat ALTER EXTENSION dalam dokumentasi PostgreSQL.

DROP EXTENSION

Anda dapat menjatuhkan ekstensi, seperti di PostgreSQL.

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Untuk informasi selengkapnya, lihat DROP EXTENSION di dokumentasi PostgreSQL.

Berikut ini DDLs tidak didukung untuk ekstensi:

ALTER EXTENSION

Anda tidak dapat menambahkan atau menjatuhkan objek anggota dari ekstensi.

```
ALTER EXTENSION name ADD member_object

ALTER EXTENSION name DROP member_object
```

perbedaan pg_buffercache di Aurora PostgreSQL Limitless Database

Di Aurora PostgreSQL Limitless Database, ketika Anda menginstal ekstensi pg_buffercache dan menggunakan `pg_buffercache` tampilan, Anda menerima informasi terkait buffer hanya dari node yang Anda sambungkan saat ini: router. Demikian pula, menggunakan fungsi `pg_buffercache_summary` atau `pg_buffercache_usage_counts` memberikan informasi hanya dari node yang terhubung.

Anda dapat memiliki banyak node dan mungkin perlu mengakses informasi buffer dari node apa pun untuk mendiagnosis masalah secara efektif. Oleh karena itu, Limitless Database menyediakan fungsi-fungsi berikut:

- `rds_aurora.limitless_pg_buffercache(subcluster_id)`
- `rds_aurora.limitless_pg_buffercache_summary(subcluster_id)`
- `rds_aurora.limitless_pg_buffercache_usage_counts(subcluster_id)`

Dengan memasukkan ID subcluster dari setiap node, apakah itu router atau shard, Anda dapat dengan mudah mengakses informasi buffer khusus untuk node itu. Fungsi-fungsi ini langsung tersedia saat Anda menginstal `pg_buffercache` ekstensi di database tanpa batas.

> **Note**
>
> Aurora PostgreSQL Limitless Database mendukung fungsi-fungsi ini untuk versi 1.4 dan lebih tinggi dari ekstensi. `pg_buffercache`

Kolom yang ditampilkan dalam `limitless_pg_buffercache` tampilan sedikit berbeda dari yang ada di `pg_buffercache` tampilan:

- `bufferid`- Tetap tidak berubah dari`pg_buffercache`.
- `relname`— Alih-alih menampilkan nomor node file seperti dalam`pg_buffercache`, `limitless_pg_buffercache` menyajikan yang terkait `relname` jika tersedia dalam database saat ini atau katalog sistem bersama, jika tidak. `NULL`
- `parent_relname`- Kolom baru ini, tidak ada di`pg_buffercache`, menampilkan induk `relname` jika nilai dalam `relname` kolom mewakili tabel yang dipartisi (dalam kasus tabel sharded). Jika tidak, itu akan ditampilkan`NULL`.
- `spcname`— Alih-alih menampilkan pengenal objek tablespace (OID) seperti pada`pg_buffercache`, `limitless_pg_buffercache` menampilkan nama tablespace.
- `datname`— Alih-alih menampilkan database OID seperti dalam`pg_buffercache`, `limitless_pg_buffercache` menampilkan nama database.
- `relforknumber`- Tetap tidak berubah dari`pg_buffercache`.
- `relblocknumber`- Tetap tidak berubah dari`pg_buffercache`.
- `isdirty`- Tetap tidak berubah dari`pg_buffercache`.
- `usagecount`- Tetap tidak berubah dari`pg_buffercache`.
- `pinning_backends`- Tetap tidak berubah dari`pg_buffercache`.

Kolom dalam `limitless_pg_buffercache_summary` dan `limitless_pg_buffercache_usage_counts` tampilan sama dengan yang ada di reguler `pg_buffercache_summary` dan `pg_buffercache_usage_counts` tampilan, masing-masing.

Dengan menggunakan fungsi-fungsi ini, Anda dapat mengakses informasi cache buffer terperinci di semua node di lingkungan Limitless Database Anda, memfasilitasi diagnosis dan pengelolaan sistem database Anda yang lebih efektif.

perbedaan pgstattuple di Aurora PostgreSQL Limitless Database

Di Aurora PostgreSQL, ekstensi pgstattuple saat ini tidak mendukung tabel asing, tabel yang dipartisi, atau indeks yang dipartisi. Namun, di Aurora PostgreSQL Limitless Database, objek yang dibuat pengguna sering di antara jenis yang tidak didukung ini. Meskipun ada tabel dan indeks reguler (misalnya, tabel katalog dan indeksnya), sebagian besar objek berada di node asing, menjadikannya benda asing untuk router.

Kami menyadari pentingnya ekstensi ini untuk mendapatkan statistik tingkat tuple, yang sangat penting untuk tugas-tugas seperti menghilangkan kembung dan mengumpulkan informasi diagnostik. Oleh karena itu, Aurora PostgreSQL Limitless Database menyediakan dukungan untuk ekstensi dalam database tanpa batas. `pgstattuple`

Aurora PostgreSQL Limitless Database mencakup fungsi-fungsi berikut dalam skema: `rds_aurora`

Fungsi statistik tingkat tuple

**`rds_aurora.limitless_pgstattuple(`*`relation_name`*`)`**

- Tujuan: Ekstrak statistik tingkat tuple untuk tabel standar dan indeksnya
- Input: `relation_name` (teks) — Nama relasi
- Output: Kolom konsisten dengan yang dikembalikan oleh `pgstattuple` fungsi di Aurora PostgreSQL

**`rds_aurora.limitless_pgstattuple(`*`relation_name`*`, `*`subcluster_id`*`)`**

- Tujuan: Ekstrak statistik tingkat tupel untuk tabel referensi, tabel sharded, tabel katalog, dan indeksnya
- Masukan:
  - `relation_name`(teks) — Nama relasi
  - `subcluster_id`(text) — ID subcluster dari node tempat statistik akan diekstraksi
- Output:
  - Untuk tabel referensi dan katalog (termasuk indeksnya), kolom konsisten dengan yang ada di Aurora PostgreSQL.
  - Untuk tabel sharded, statistik hanya mewakili partisi tabel sharded yang berada pada subcluster yang ditentukan.

Fungsi statistik indeks

**`rds_aurora.limitless_pgstatindex(`*`relation_name`*`)`**

- Tujuan: Ekstrak statistik untuk indeks B-tree pada tabel standar
- Masukan: `relation_name` (teks) — Nama indeks B-pohon
- Output: Semua kolom `root_block_no` kecuali dikembalikan. Kolom yang dikembalikan konsisten dengan `pgstatindex` fungsi di Aurora PostgreSQL.

**`rds_aurora.limitless_pgstatindex(`*`relation_name`*`, `*`subcluster_id`*`)`**

- Tujuan: Ekstrak statistik untuk indeks B-tree pada tabel referensi, tabel sharded, dan tabel katalog.
- Masukan:
  - `relation_name`(teks) — Nama indeks B-pohon
  - `subcluster_id`(text) — ID subcluster dari node tempat statistik akan diekstraksi
- Output:
  - Untuk indeks tabel referensi dan katalog, semua kolom (kecuali`root_block_no`) dikembalikan. Kolom yang dikembalikan konsisten dengan Aurora PostgreSQL.
  - Untuk tabel sharded, statistik hanya mewakili partisi indeks tabel sharded yang berada pada subcluster yang ditentukan. `tree_level`Kolom menunjukkan rata-rata di semua irisan tabel pada subcluster yang diminta.

**`rds_aurora.limitless_pgstatginindex(`*`relation_name`*`)`**

- Tujuan: Ekstrak statistik untuk Indeks Terbalik Umum (GINs) pada tabel standar
- Input: `relation_name` (teks) — Nama GIN
- Output: Kolom konsisten dengan yang dikembalikan oleh `pgstatginindex` fungsi di Aurora PostgreSQL

**`rds_aurora.limitless_pgstatginindex(`*`relation_name`*`, `*`subcluster_id`*`)`**

- Tujuan: Ekstrak statistik untuk indeks GIN pada tabel referensi, tabel sharded, dan tabel katalog.
- Masukan:
  - `relation_name`(teks) — Nama indeks
  - `subcluster_id`(text) — ID subcluster dari node tempat statistik akan diekstraksi
- Output:
  - Untuk indeks GIN tabel referensi dan katalog, kolom konsisten dengan yang ada di Aurora PostgreSQL.

- Untuk tabel sharded, statistik hanya mewakili partisi indeks tabel sharded yang berada pada subcluster yang ditentukan.

**`rds_aurora.limitless_pgstathashindex(`*`relation_name`*`)`**

- Tujuan: Ekstrak statistik untuk indeks hash pada tabel standar
- Input: `relation_name` (teks) — Nama indeks hash
- Output: Kolom konsisten dengan yang dikembalikan oleh `pgstathashindex` fungsi di Aurora PostgreSQL

**`rds_aurora.limitless_pgstathashindex(`*`relation_name`*`, `*`subcluster_id`*`)`**

- Tujuan: Ekstrak statistik untuk indeks hash pada tabel referensi, tabel sharded, dan tabel katalog.
- Masukan:
  - `relation_name`(teks) — Nama indeks
  - `subcluster_id`(text) — ID subcluster dari node tempat statistik akan diekstraksi
- Output:
  - Untuk indeks hash tabel referensi dan katalog, kolom konsisten dengan Aurora PostgreSQL.
  - Untuk tabel sharded, statistik hanya mewakili partisi indeks tabel sharded yang berada pada subcluster yang ditentukan.

Fungsi penghitungan halaman

**`rds_aurora.limitless_pg_relpages(`*`relation_name`*`)`**

- Tujuan: Ekstrak jumlah halaman untuk tabel standar dan indeksnya
- Input: `relation_name` (teks) — Nama relasi
- Output: Jumlah halaman dari relasi yang ditentukan

**`rds_aurora.limitless_pg_relpages(`*`relation_name`*`, `*`subcluster_id`*`)`**

- Tujuan: Ekstrak jumlah halaman untuk tabel referensi, tabel sharded, dan tabel katalog (termasuk indeksnya)
- Masukan:
  - `relation_name`(teks) — Nama relasi
  - `subcluster_id`(text) — ID subcluster dari node tempat jumlah halaman akan diekstraksi
- Keluaran: Untuk tabel sharded, jumlah halaman adalah jumlah halaman di semua irisan tabel pada subcluster yang ditentukan.

Perkiraan fungsi statistik tingkat tupel

**`rds_aurora.limitless_pgstattuple_approx(`*`relation_name`*`)`**

- Tujuan: Ekstrak perkiraan statistik tingkat tupel untuk tabel standar dan indeksnya
- Input: `relation_name` (teks) — Nama relasi
- Output: Kolom konsisten dengan yang dikembalikan oleh fungsi pgstattuple_approx di Aurora PostgreSQL

**`rds_aurora.limitless_pgstattuple_approx(`*`relation_name`*`, `*`subcluster_id`*`)`**

- Tujuan: Ekstrak perkiraan statistik tingkat tupel untuk tabel referensi, tabel sharded, dan tabel katalog (termasuk indeksnya)
- Masukan:
  - `relation_name`(teks) — Nama relasi
  - `subcluster_id`(text) — ID subcluster dari node tempat statistik akan diekstraksi
- Output:
  - Untuk tabel referensi dan katalog (termasuk indeksnya), kolom konsisten dengan yang ada di Aurora PostgreSQL.
  - Untuk tabel sharded, statistik hanya mewakili partisi tabel sharded yang berada pada subcluster yang ditentukan.

> **Note**
>
> Saat ini, Aurora PostgreSQL Limitless Database tidak mendukung `pgstattuple` ekstensi pada tampilan terwujud, tabel TOAST, atau tabel sementara.
> Di Aurora PostgreSQL Limitless Database, Anda harus memberikan input sebagai teks, meskipun Aurora PostgreSQL mendukung format lain.

## Kunci asing

Kendala kunci asing (`FOREIGN KEY`) didukung dengan beberapa batasan:

- `CREATE TABLE`dengan `FOREIGN KEY` didukung hanya untuk tabel standar. Untuk membuat tabel sharded atau referensi dengan`FOREIGN KEY`, pertama buat tabel tanpa kendala kunci asing. Kemudian memodifikasinya menggunakan pernyataan berikut:

  ```
  ALTER TABLE ADD CONSTRAINT;
  ```

- Mengonversi tabel standar ke tabel sharded atau referensi tidak didukung ketika tabel memiliki kendala kunci asing. Jatuhkan kendala, lalu tambahkan setelah konversi.
- Batasan berikut berlaku untuk jenis tabel untuk kendala kunci asing:
  - Tabel standar dapat memiliki kendala kunci asing ke tabel standar lainnya.
  - Sebuah tabel sharded dapat memiliki kendala kunci asing jika tabel induk dan anak ditempatkan dan kunci asing adalah superset dari kunci shard.
  - Tabel sharded dapat memiliki kendala kunci asing ke tabel referensi.
  - Tabel referensi dapat memiliki kendala kunci asing ke tabel referensi lain.

Topik

- Opsi kunci asing
- Contoh

### Opsi kunci asing

Kunci asing didukung di Aurora PostgreSQL Limitless Database untuk beberapa opsi DDL. Tabel berikut mencantumkan opsi yang didukung dan tidak didukung antara tabel Aurora PostgreSQL Limitless Database.

| Opsi DDL | Referensi untuk referensi | Sharded menjadi sharded (collocated) | Sharded ke referensi | Standar ke standar |
|---|---|---|---|---|
| `DEFERRABLE` | Ya | Ya | Ya | Ya |

| Opsi DDL | Referensi untuk referensi | Sharded menjadi sharded (collocated) | Sharded ke referensi | Standar ke standar |
|---|---|---|---|---|
| `INITIALLY DEFERRED` | Ya | Ya | Ya | Ya |
| `INITIALLY IMMEDIATE` | Ya | Ya | Ya | Ya |
| `MATCH FULL` | Ya | Ya | Ya | Ya |
| `MATCH PARTIAL` | Tidak | Tidak | Tidak | Tidak |
| `MATCH SIMPLE` | Ya | Ya | Ya | Ya |
| `NOT DEFERRABLE` | Ya | Ya | Ya | Ya |
| `NOT VALID` | Ya | Tidak | Tidak | Ya |
| `ON DELETE CASCADE` | Ya | Ya | Ya | Ya |
| `ON DELETE NO ACTION` | Ya | Ya | Ya | Ya |
| `ON DELETE RESTRICT` | Ya | Ya | Ya | Ya |
| `ON DELETE SET DEFAULT` | Tidak | Tidak | Tidak | Tidak |
| `ON DELETE SET NULL` | Ya | Tidak | Tidak | Ya |
| `ON UPDATE CASCADE` | Tidak | Tidak | Tidak | Ya |

| Opsi DDL | Referensi untuk referensi | Sharded menjadi sharded (collocated) | Sharded ke referensi | Standar ke standar |
|---|---|---|---|---|
| `ON UPDATE NO ACTION` | Ya | Ya | Ya | Ya |
| `ON UPDATE RESTRICT` | Ya | Ya | Ya | Ya |
| `ON UPDATE SET DEFAULT` | Tidak | Tidak | Tidak | Tidak |
| `ON UPDATE SET NULL` | Ya | Tidak | Tidak | Ya |

Contoh

- Standar ke standar:

```
set rds_aurora.limitless_create_table_mode='standard';

CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);

SELECT constraint_name, table_name, constraint_type
FROM information_schema.table_constraints WHERE constraint_type='FOREIGN KEY';

 constraint_name         | table_name  | constraint_type
-------------------------+-------------+-----------------
 orders_product_no_fkey  | orders      | FOREIGN KEY
```

```
(1 row)
```

- Sharded ke sharded (collocated):

```
set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"product_no"}';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
set rds_aurora.limitless_create_table_collocate_with='products';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- Sharded ke referensi:

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- Referensi untuk referensi:

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

## Fungsi

Fungsi didukung di Aurora PostgreSQL Limitless Database.

DDLs Berikut ini didukung untuk fungsi:

CREATE FUNCTION

Anda dapat membuat fungsi, seperti di Aurora PostgreSQL, dengan pengecualian mengubah volatilitasnya saat menggantinya.

Untuk informasi selengkapnya, lihat CREATE FUNCTION dalam dokumentasi PostgreSQL.

ALTER FUNCTION

Anda dapat mengubah fungsi, seperti di Aurora PostgreSQL, dengan pengecualian mengubah volatilitasnya.

Untuk informasi selengkapnya, lihat MENGUBAH FUNGSI dalam dokumentasi PostgreSQL.

FUNGSI DROP

Anda dapat menjatuhkan fungsi, seperti di Aurora PostgreSQL.

```
DROP FUNCTION [ IF EXISTS ] name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ]
 [, ...]
    [ CASCADE | RESTRICT ]
```

Untuk informasi selengkapnya, lihat FUNGSI DROP dalam dokumentasi PostgreSQL.

Topik

- Distribusi fungsi
- Volatilitas fungsi

### Distribusi fungsi

Ketika semua pernyataan fungsi ditargetkan ke pecahan tunggal, ada baiknya untuk mendorong seluruh fungsi ke pecahan target. Kemudian hasilnya disebarkan kembali ke router, daripada mengurai fungsi di router itu sendiri. Fungsi dan kemampuan pushdown prosedur tersimpan berguna bagi pelanggan yang ingin menjalankan fungsinya atau prosedur tersimpan lebih dekat ke sumber data, yaitu pecahan.

Untuk mendistribusikan fungsi, pertama buat fungsi, lalu panggil `rds_aurora.limitless_distribute_function` prosedur untuk mendistribusikannya. Fungsi ini menggunakan sintaks berikut:

```
SELECT rds_aurora.limitless_distribute_function('function_prototype',
 ARRAY['shard_key'], 'collocating_table');
```

Fungsi ini mengambil parameter berikut:

- *`function_prototype`*Fungsi yang akan didistribusikan. Sebutkan hanya argumen masukan, dan bukan argumen keluaran apa pun.

  Jika salah satu argumen didefinisikan sebagai OUT parameter, jangan sertakan tipenya dalam argumen`function_prototype`.
- `ARRAY['`*`shard_key`*`']`— Daftar argumen fungsi yang diidentifikasi sebagai kunci shard untuk fungsi tersebut.
- *`collocating_table`*— Tabel sharded yang berisi rentang data pada pecahan target.

Untuk mengidentifikasi pecahan tempat menekan fungsi ini untuk berjalan, sistem mengambil `ARRAY['`*`shard_key`*`']` argumen, melakukan hash, dan menemukan pecahan dari yang menampung rentang *`collocating_table`* yang berisi nilai hash ini.

Pembatasan

Ketika Anda mendistribusikan fungsi atau prosedur, itu hanya berhubungan dengan data yang dibatasi oleh rentang kunci shard di pecahan itu. Dalam kasus di mana fungsi atau prosedur mencoba mengakses data dari pecahan yang berbeda, hasil yang dikembalikan oleh fungsi atau prosedur terdistribusi akan berbeda dibandingkan dengan yang tidak didistribusikan.

Misalnya, Anda membuat fungsi yang berisi kueri yang akan menyentuh beberapa pecahan, tetapi kemudian memanggil `rds_aurora.limitless_distribute_function` prosedur untuk mendistribusikannya. Saat Anda memanggil fungsi ini dengan memberikan argumen untuk kunci shard, kemungkinan hasil menjalankannya akan dibatasi oleh nilai yang ada dalam pecahan itu. Hasil ini berbeda dari yang dihasilkan tanpa mendistribusikan fungsi.

Contoh

Pertimbangkan fungsi berikut `func` di mana kita memiliki tabel sharded `customers` dengan kunci shard. `customer_id`

```
postgres_limitless=> CREATE OR REPLACE FUNCTION func(c_id integer, sc integer)
 RETURNS int
    language SQL
    volatile
    AS $$
    UPDATE customers SET score = sc WHERE customer_id = c_id RETURNING score;
    $$;
```

Sekarang kami mendistribusikan fungsi ini:

```
SELECT rds_aurora.limitless_distribute_function('func(integer, integer)',
 ARRAY['c_id'], 'customers');
```

Berikut ini adalah contoh rencana query.

```
EXPLAIN(costs false, verbose true) SELECT func(27+1,10);

                    QUERY PLAN
 ---------------------------------------------------
  Foreign Scan
    Output: (func((27 + 1), 10))
    Remote SQL:  SELECT func((27 + 1), 10) AS func
  Single Shard Optimized
 (4 rows)
```

```
EXPLAIN(costs false, verbose true)
 SELECT * FROM customers,func(customer_id, score) WHERE customer_id=10 AND score=27;

                          QUERY PLAN
 ------------------------------------------------------------------------
  Foreign Scan
    Output: customer_id, name, score, func
    Remote SQL:  SELECT customers.customer_id,
      customers.name,
      customers.score,
      func.func
     FROM public.customers,
      LATERAL func(customers.customer_id, customers.score) func(func)
    WHERE ((customers.customer_id = 10) AND (customers.score = 27))
  Single Shard Optimized
 (10 rows)
```

Contoh berikut menunjukkan prosedur dengan IN dan OUT parameter sebagai argumen.

```
CREATE OR REPLACE FUNCTION get_data(OUT id INTEGER, IN arg_id INT)
    AS $$
    BEGIN
        SELECT customer_id,
        INTO id
        FROM customer
        WHERE customer_id = arg_id;
    END;
    $$ LANGUAGE plpgsql;
```

Contoh berikut mendistribusikan prosedur hanya menggunakan IN parameter.

```
EXPLAIN(costs false, verbose true) SELECT * FROM get_data(1);

             QUERY PLAN
 ---------------------------------
  Foreign Scan
    Output: id
    Remote SQL:  SELECT customer_id
     FROM get_data(1) get_data(id)
  Single Shard Optimized
 (6 rows)
```

Volatilitas fungsi

Anda dapat menentukan apakah suatu fungsi tidak dapat diubah, stabil, atau tidak stabil dengan memeriksa `provolatile` nilai dalam tampilan pg_proc. `provolatile`Nilai menunjukkan apakah hasil fungsi hanya bergantung pada argumen masukan, atau dipengaruhi oleh faktor luar.

Nilainya adalah salah satu dari berikut ini:

- `i`— Fungsi yang tidak dapat diubah, yang selalu memberikan hasil yang sama untuk input yang sama
- `s`— Fungsi stabil, yang hasilnya (untuk input tetap) tidak berubah dalam pemindaian
- `v`— Fungsi volatil, yang hasilnya dapat berubah sewaktu-waktu. Juga gunakan `v` untuk fungsi dengan efek samping, sehingga panggilan ke mereka tidak dapat dioptimalkan.

Contoh berikut menunjukkan fungsi volatile.

```
SELECT proname, provolatile FROM pg_proc WHERE proname='pg_sleep';

 proname  | provolatile
----------+-------------
 pg_sleep | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='uuid_generate_v4';

     proname      | provolatile
------------------+-------------
 uuid_generate_v4 | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='nextval';

 proname | provolatile
---------+-------------
 nextval | v
(1 row)
```

Mengubah volatilitas fungsi yang ada tidak didukung di Aurora PostgreSQL Limitless Database. Ini berlaku untuk keduanya `ALTER FUNCTION` dan `CREATE OR REPLACE FUNCTION` perintah, seperti yang ditunjukkan pada contoh berikut.

```
-- Create an immutable function
CREATE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$IMMUTABLE;

-- Altering the volatility throws an error
ALTER FUNCTION immutable_func1 STABLE;

-- Replacing the function with altered volatility throws an error
CREATE OR REPLACE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
```

```
$$VOLATILE;
```

Kami sangat menyarankan Anda menetapkan volatilitas yang benar ke fungsi. Misalnya, jika fungsi Anda menggunakan `SELECT` dari beberapa tabel atau referensi objek database, jangan atur sebagai`IMMUTABLE`. Jika isi tabel pernah berubah, kekekalan rusak.

Aurora PostgreSQL memungkinkan fungsi yang tidak dapat diubah `SELECT` di dalam, tetapi hasilnya mungkin salah. Aurora PostgreSQL Limitless Database dapat mengembalikan kesalahan dan hasil yang salah. Untuk informasi selengkapnya tentang volatilitas fungsi, lihat Kategori volatilitas fungsi dalam dokumentasi PostgreSQL.

## Urutan

Urutan bernama adalah objek database yang menghasilkan angka unik dalam urutan naik atau turun. `CREATE SEQUENCE`membuat generator nomor urut baru. Nilai urutan dijamin unik.

Saat Anda membuat urutan bernama di Aurora PostgreSQL Limitless Database, objek urutan terdistribusi dibuat. Kemudian Aurora PostgreSQL Limitless Database mendistribusikan potongan nilai urutan yang tidak tumpang tindih di semua Router Transaksi Terdistribusi (router). Potongan direpresentasikan sebagai objek urutan lokal pada router; oleh karena itu, operasi urutan seperti `nextval` dan dijalankan `currval` secara lokal. Router beroperasi secara independen, dan meminta potongan baru dari urutan terdistribusi bila diperlukan.

Untuk informasi selengkapnya tentang urutan, lihat CREATE SEQUENCE dalam dokumentasi PostgreSQL.

Topik

- Meminta potongan baru
- Batasan
- Opsi yang tidak didukung
- Contoh
- Tampilan urutan
- Memecahkan masalah urutan

### Meminta potongan baru

Anda mengkonfigurasi ukuran potongan yang dialokasikan pada router dengan menggunakan parameter. `rds_aurora.limitless_sequence_chunk_size` Nilai default-nya adalah `250000`. Setiap router awalnya memiliki dua potongan: aktif dan dicadangkan. Potongan aktif digunakan untuk mengonfigurasi objek urutan lokal (pengaturan `minvalue` dan`maxvalue`), dan potongan cadangan disimpan dalam tabel katalog internal. Ketika potongan aktif mencapai nilai minimum atau maksimum, itu diganti dengan potongan cadangan. Untuk melakukan itu, `ALTER SEQUENCE` digunakan secara internal, artinya `AccessExclusiveLock` diperoleh.

Pekerja latar belakang berjalan setiap 10 detik pada node router untuk memindai urutan potongan cadangan yang digunakan. Jika potongan bekas ditemukan, pekerja meminta potongan baru dari urutan terdistribusi. Pastikan untuk mengatur ukuran potongan cukup besar sehingga pekerja latar

belakang memiliki cukup waktu untuk meminta yang baru. Permintaan jarak jauh tidak pernah terjadi dalam konteks sesi pengguna, yang berarti Anda tidak dapat meminta urutan baru secara langsung.

Batasan

Batasan berikut berlaku untuk urutan di Aurora PostgreSQL Limitless Database:

- pg_sequenceKatalog, pg_sequences fungsi, dan SELECT * FROM *sequence_name* pernyataan semuanya hanya menampilkan status urutan lokal, bukan status terdistribusi.
- Nilai urutan dijamin unik, dan dijamin monotonik dalam satu sesi. Tetapi mereka bisa rusak dengan nextval pernyataan yang dijalankan di sesi lain, jika sesi tersebut terhubung ke router lain.
- Pastikan bahwa ukuran urutan (jumlah nilai yang tersedia) cukup besar untuk didistribusikan di semua router. Gunakan rds_aurora.limitless_sequence_chunk_size parameter untuk mengkonfigurasi filechunk_size. (Setiap router memiliki dua potongan.)
- CACHEOpsi ini didukung, tetapi cache harus lebih kecil darichunk_size.

Opsi yang tidak didukung

Opsi berikut tidak didukung untuk urutan di Aurora PostgreSQL Limitless Database.

Fungsi manipulasi urutan

setvalFungsi ini tidak didukung. Untuk informasi selengkapnya, lihat Fungsi Manipulasi Urutan dalam dokumentasi PostgreSQL.

BUAT URUTAN

Opsi berikut tidak didukung.

```
CREATE [{ TEMPORARY | TEMP} | UNLOGGED] SEQUENCE
    [[ NO ] CYCLE]
```

Untuk informasi selengkapnya, lihat CREATE SEQUENCE dalam dokumentasi PostgreSQL.

MENGUBAH URUTAN

Opsi berikut tidak didukung.

```
ALTER SEQUENCE
```

```
    [[ NO ] CYCLE]
```

Untuk informasi selengkapnya, lihat ALTER SEQUENCE dalam dokumentasi PostgreSQL.

ALTER TABLE

`ALTER TABLE` Perintah tidak didukung untuk urutan.

Contoh

MEMBUAT/MENJATUHKAN URUTAN

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
       1
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16960 |       20 |        1 |            1 |  10000 |      1 |        1 | f
(1 row)

% connect to another router
postgres_limitless=> SELECT nextval('s');

 nextval
---------
   10001
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
```

```
----------+----------+----------+--------------+--------+--------+----------
+----------
    16959 |       20 |    10001 |            1 |  20000 |  10001 |        1 | f
(1 row)

postgres_limitless=> DROP SEQUENCE s;
DROP SEQUENCE
```

## MENGUBAH URUTAN

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> ALTER SEQUENCE s RESTART 500;
ALTER SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
     500
(1 row)

postgres_limitless=> SELECT currval('s');

 currval
---------
     500
(1 row)
```

## Fungsi manipulasi urutan

```
postgres=# CREATE TABLE t(a bigint primary key, b bigint);
CREATE TABLE

postgres=# CREATE SEQUENCE s minvalue 0 START 0;
CREATE SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));
```

```
INSERT 0 1

postgres=# SELECT * FROM t;

 a | b
---+---
 0 | 0
 1 | 1
(2 rows)

postgres=# ALTER SEQUENCE s RESTART 10000;
ALTER SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# SELECT * FROM t;

   a   |   b
-------+-------
     0 |     0
     1 |     1
 10000 | 10000
(3 rows)
```

### Tampilan urutan

Aurora PostgreSQL Limitless Database menyediakan tampilan berikut untuk urutan.

#### rds_aurora.limitless_distributed_sequence

Tampilan ini menunjukkan status urutan terdistribusi dan konfigurasi. `cache`Kolom `minvalue` `maxvaluestart`,`inc`,, dan memiliki arti yang sama seperti pada tampilan pg_sequences, dan menampilkan opsi yang dengannya urutan dibuat. `lastval`Kolom menunjukkan nilai alokasi atau cadangan terbaru dalam objek urutan terdistribusi. Ini tidak berarti bahwa nilainya sudah digunakan, karena router menyimpan potongan urutan secara lokal.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_distributed_sequence WHERE
 sequence_name='test_serial_b_seq';

 schema_name |   sequence_name   | lastval | minvalue |  maxvalue  | start | inc |
 cache
-------------+-------------------+---------+----------+------------+-------+-----
+-------
 public      | test_serial_b_seq | 1250000 |        1 | 2147483647 |     1 |   1 |
   1
(1 row)
```

#### rds_aurora.limitless_sequence_metadata

Tampilan ini menunjukkan metadata urutan terdistribusi dan metadata urutan agregat dari node cluster. Ini menggunakan kolom berikut:

- `subcluster_id`— ID node cluster yang memiliki potongan.
- Active chunk — Sebuah potongan dari urutan yang sedang digunakan (`active_minvalue`,`active_maxvalue`).
- Potongan cadangan — Potongan lokal yang akan digunakan next (`reserved_minvalue`,`reserved_maxvalue`).
- `local_last_value`— Nilai terakhir yang diamati dari urutan lokal.
- `chunk_size`— Ukuran potongan, seperti yang dikonfigurasi pada pembuatan.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_sequence_metadata WHERE
 sequence_name='test_serial_b_seq' order by subcluster_id;
```

```
 subcluster_id |   sequence_name   | schema_name | active_minvalue | active_maxvalue
 | reserved_minvalue | reserved_maxvalue | chunk_size | chunk_state |
 local_last_value
---------------+-------------------+-------------+-----------------
+-----------------+-------------------+-------------------+------------
+-------------+------------------
 1             | test_serial_b_seq | public      |          500001 |          750000
 |           1000001 |           1250000 |     250000 |           1 |
 550010
 2             | test_serial_b_seq | public      |          250001 |          500000
 |            750001 |           1000000 |     250000 |           1 |

(2 rows)
```

Memecahkan masalah urutan

Masalah berikut dapat terjadi dengan urutan.

Ukuran potongan tidak cukup besar

Jika ukuran potongan tidak diatur cukup besar dan tingkat transaksi tinggi, pekerja latar belakang mungkin tidak memiliki cukup waktu untuk meminta potongan baru sebelum potongan aktif habis. Hal ini dapat menyebabkan pertengkaran dan peristiwa menunggu sepertiLIMITLESS:AuroraLimitlessSequenceReplace,, LWLock:LockManagerLockrelation, danLWlock:bufferscontent.

Tingkatkan nilai parameter rds_aurora.limitless_sequence_chunk_size.

Cache urutan diatur terlalu tinggi

Di PostgreSQL, caching urutan terjadi pada tingkat sesi. Setiap sesi mengalokasikan nilai urutan berturut-turut selama satu akses ke objek urutan, dan meningkatkan objek urutan yang sesuai. last_value Kemudian, penggunaan berikutnya dari nextval dalam sesi itu hanya mengembalikan nilai yang telah dialokasikan sebelumnya tanpa menyentuh objek urutan.

Setiap angka yang dialokasikan tetapi tidak digunakan dalam sesi akan hilang ketika sesi itu berakhir, menghasilkan “lubang” dalam urutan. Ini dapat menghabiskan sequence_chunk dengan cepat dan menyebabkan pertengkaran dan peristiwa menunggu seperti,,, dan. LIMITLESS:AuroraLimitlessSequenceReplace LWLock:LockManager Lockrelation LWlock:bufferscontent

Kurangi pengaturan cache urutan.

Gambar berikut menunjukkan peristiwa tunggu yang disebabkan oleh masalah urutan.

# Didukung dan tidak didukung Data Manipulation Language (DHTML) dan perintah query processing SQL

Tabel berikut mencantumkan perintah DHTML yang didukung dan tidak didukung oleh Aurora PostgreSQL Limitless Database, dengan referensi ke batasan atau informasi lebih lanjut.

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| MENGGUGURKAN | Ya | Tidak ada |
| MENGANALISA | Ya | ANALYZE |
| MULAI | Ya | Tidak ada |
| PANGGILAN | Ya | Tidak ada |
| CHECKPOINT | Ya | Tidak ada |
| TUTUP | Ya | Tidak ada |
| GUGUS | Ya | CLUSTER |
| COMMIT | Ya | Tidak ada |
| KOMIT DISIAPKAN | Tidak | Tidak berlaku |
| MENYONTEK | Ya | Tidak ada |
| DEALOKASI | Ya | Tidak ada |
| MENYATAKAN | Ya | Tidak ada |
| HAPUS | Ya | Tidak ada |
| MEMBUANG | Ya | Tidak ada |
| BERBUAT | Ya | Tidak ada |
| AKHIR | Ya | Tidak ada |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| EXECUTE | Ya | Tidak ada |
| EXPLAIN | Ya | EXPLAIN |
| AMBIL | Ya | Tidak ada |
| IMPOR SKEMA ASING | Tidak | Tidak berlaku |
| INSERT | Ya | INSERT |
| DENGARKAN | Tidak | Tidak berlaku |
| GEMBOK | Ya | Tidak ada |
| MERGE | Tidak | Tidak berlaku |
| PINDAH | Ya | Tidak ada |
| MEMBERITAHUKAN | Tidak | Tidak berlaku |
| BUKA | Ya | Tidak ada |
| MEMPERSIAPKAN | Ya | Tidak ada |
| MENYIAPKAN TRANSAKSI | Tidak | Tidak berlaku |
| MENYEGARKAN TAMPILAN TERWUJUD | Tidak | Tidak berlaku |
| INDEKS ULANG | Tidak | Tidak berlaku |
| RILIS SAVEPOINT | Ya | Tidak ada |
| ROLLBACK | Ya | Tidak ada |
| ROLLBACK DISIAPKAN | Tidak | Tidak berlaku |
| ROLLBACK KE SAVEPOINT | Ya | Tidak ada |

| Perintah | Didukung? | Keterbatasan atau informasi lebih lanjut |
|---|---|---|
| SAVEPOINT | Ya | Tidak ada |
| SELECT | Ya | Tidak ada |
| PILIH KE DALAM | Ya | Tidak ada |
| MEMPERLIHATKAN | Ya | Tidak ada |
| START TRANSACTION | Ya | Tidak ada |
| TIDAK MENDENGARKAN | Tidak | Tidak ada |
| UPDATE | Ya | UPDATE |
| PERBARUI... DI MANA SAAT INI | Tidak | Tidak berlaku |
| VAKUM | Ya | VACUUM |
| NILAI | Ya | Tidak ada |

# DMLketerbatasan dan informasi lainnya untuk Aurora SQL Postgre Limitless Database

Topik berikut menjelaskan batasan atau memberikan informasi lebih lanjut untuk DML dan SQL perintah pemrosesan kueri di Aurora SQL Postgre Limitless Database.

Topik


- ANALYZE
- CLUSTER
- EXPLAIN
- INSERT
- UPDATE
- VACUUM


## ANALYZE

`ANALYZE`Perintah mengumpulkan statistik tentang isi tabel dalam database. Selanjutnya, perencana kueri menggunakan statistik ini untuk membantu menentukan rencana eksekusi yang paling efisien untuk kueri. Untuk informasi lebih lanjut, lihat ANALYZEdi dokumentasi PostgreSQL.

Dalam Aurora Postgre SQL Limitless Database, `ANALYZE` perintah mengumpulkan statistik tabel di semua router dan pecahan saat dijalankan.

Untuk mencegah perhitungan statistik pada setiap router selama `ANALYZE` berjalan, statistik tabel dihitung pada salah satu router dan kemudian disalin ke router peer.

## CLUSTER

`CLUSTER`Perintah secara fisik menyusun ulang tabel berdasarkan indeks. Indeks harus sudah ditentukan di atas meja. Dalam Aurora Postgre SQL Limitless Database, pengelompokan bersifat lokal ke bagian indeks yang ada pada setiap pecahan.

Untuk informasi lebih lanjut, lihat CLUSTERdi dokumentasi PostgreSQL.

## EXPLAIN

Anda menggunakan parameter berikut untuk mengkonfigurasi output dari `EXPLAIN` perintah:

- `rds_aurora.limitless_explain_options`— Apa yang harus dimasukkan dalam `EXPLAIN` output. Nilai defaultnya adalah`single_shard_optimization`: apakah paket dioptimalkan dengan pecahan tunggal ditampilkan, tetapi paket pecahan tidak disertakan.

Dalam contoh ini, `EXPLAIN` output tidak menampilkan rencana dari pecahan.

```
postgres_limitless=> EXPLAIN SELECT * FROM employees where id =25;

                      QUERY PLAN
------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
 Single Shard Optimized
(2 rows)
```

Sekarang kita mengatur `rds_aurora.limitless_explain_options` untuk memasukkan `shard_plans` dan`single_shard_optimization`. Kita dapat melihat rencana eksekusi pernyataan pada router dan pecahan. Selain itu, kami menonaktifkan `enable_seqscan` parameter untuk menegakkan bahwa pemindaian indeks digunakan pada lapisan pecahan.

```
postgres_limitless=> SET rds_aurora.limitless_explain_options = shard_plans,
 single_shard_optimization;
SET

postgres_limitless=> SET enable_seqscan = OFF;
SET

postgres_limitless=> EXPLAIN SELECT * FROM employees WHERE id = 25;

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Remote Plans from Shard postgres_s4:
         Index Scan using employees_ts00287_id_idx on employees_ts00287
 employees_fs00003  (cost=0.14..8.16 rows=1 width=15)
           Index Cond: (id = 25)
 Single Shard Optimized
(5 rows)
```

Untuk informasi lebih lanjut tentang `EXPLAIN` perintah, lihat EXPLAINdi dokumentasi PostgreSQL.

## INSERT

Sebagian besar `INSERT` perintah didukung di Aurora SQL Postgre Limitless Database.

Postgre SQL tidak memiliki `UPSERT` perintah eksplisit, tetapi mendukung pernyataan. `INSERT ... ON CONFLICT`

`INSERT ... ON CONFLICT`tidak didukung jika tindakan konflik memiliki subquery atau fungsi yang bisa berubah:

```
-- RANDOM is a mutable function.
INSERT INTO sharded_table VALUES (1, 100) ON CONFLICT (id) DO UPDATE SET other_id =
 RANDOM();

ERROR: Aurora Limitless Tables doesn't support pushdown-unsafe functions with DO UPDATE
 clauses.
```

Untuk informasi lebih lanjut tentang `INSERT` perintah, lihat INSERTdi dokumentasi PostgreSQL.

## UPDATE

Memperbarui kunci shard tidak didukung. Misalnya, Anda memiliki tabel sharded yang disebut`customers`, dengan kunci pecahan. `customer_id` DMLPernyataan berikut menyebabkan kesalahan:

```
postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_id =1;
ERROR:  Shard key column update is not supported

postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_name='abc';
ERROR:  Shard key column update is not supported
```

Untuk memperbarui kunci shard, pertama Anda harus `DELETE` baris dengan kunci shard, lalu baris baru `INSERT` dengan nilai kunci shard yang diperbarui.

Untuk informasi selengkapnya tentang `UPDATE` perintah, lihat Memperbarui data dalam dokumentasi PostgreSQL.

## VACUUM

Anda dapat melakukan penyedot debu pada tabel sharded dan referensi. `VACUUM`Fungsi-fungsi berikut didukung penuh di Aurora SQL Postgre Limitless Database:

- VACUUM
- ANALYZE
- DISABLE_PAGE_SKIPPING
- FREEZE
- FULL
- INDEX_CLEANUP
- PARALLEL
- PROCESS_TOAST
- TRUNCATE
- VERBOSE

`VACUUM`pada Aurora Postgre SQL Limitless Database memiliki batasan sebagai berikut:

- Ekstensi pg_visibility_map tidak didukung.
- Memeriksa indeks yang tidak digunakan dengan tampilan pg_stat_all_indexes tidak didukung.
- Tampilan konsolidasi untuk pg_stat_user_indexes, pg_class, dan pg_stats tidak diimplementasikan.

Untuk informasi lebih lanjut tentang `VACUUM` perintah, lihat VACUUMdi dokumentasi PostgreSQL. Untuk informasi lebih lanjut tentang cara kerja penyedot debu di Aurora SQL Postgre Limitless Database, lihat. Merebut kembali ruang penyimpanan dengan menyedot debu

# Variabel dalam Basis Data Tanpa Batas Aurora Postgre SQL

Anda dapat menggunakan variabel berikut untuk mengkonfigurasi Aurora SQL Postgre Limitless Database.

rds_aurora.limitless_active_shard_key

Menetapkan kunci shard tunggal saat query database, menyebabkan semua `SELECT` dan DML query ditambahkan dengan kunci shard sebagai predikat konstan. Untuk informasi selengkapnya, lihat Menyetel kunci pecahan aktif.

rds_aurora.limitless_create_table_collocate_with

Tetapkan variabel ini ke nama tabel tertentu untuk membentuk tabel yang baru dibuat dengan tabel itu. Untuk informasi selengkapnya, lihat Membuat tabel tanpa batas dengan menggunakan variabel.

rds_aurora.limitless_create_table_mode

Mengatur mode pembuatan tabel. Untuk informasi selengkapnya, lihat Membuat tabel tanpa batas dengan menggunakan variabel.

rds_aurora.limitless_create_table_shard_key

Atur variabel ini ke array nama kolom untuk digunakan sebagai kunci shard. Untuk informasi selengkapnya, lihat Membuat tabel tanpa batas dengan menggunakan variabel.

rds_aurora.limitless_explain_options

Apa yang harus dimasukkan dalam `EXPLAIN` output. Untuk informasi selengkapnya, lihat EXPLAIN.

# Parameter cluster DB di Aurora SQL Postgre Limitless Database

Anda dapat menggunakan parameter cluster DB berikut untuk mengkonfigurasi Aurora SQL Postgre Limitless Database.

rds_aurora.limitless_adaptive_fetch_size

Meningkatkan batch prefetching. Saat disetel ke`true`, parameter ini memungkinkan ukuran pengambilan yang dapat disesuaikan sendiri (adaptif) untuk prefetching. Ketika diatur ke`false`, ukuran fetch konstan.

rds_aurora.limitless_auto_scale_options

Menetapkan opsi yang tersedia untuk menambahkan router atau memisahkan pecahan dalam grup pecahan DB. Nilainya bisa`add_router`,`split_shard`, atau keduanya.

Untuk informasi selengkapnya, silakan lihat Menambahkan router ke grup pecahan DB dan Memisahkan pecahan dalam grup pecahan DB.

rds_aurora.limitless_distributed_deadlock_timeout

Jumlah waktu untuk menunggu di kunci sebelum memeriksa apakah ada kondisi kebuntuan yang didistribusikan, dalam milidetik. Defaultnya adalah `1000` (1 detik).

Untuk informasi selengkapnya, lihat Kebuntuan yang didistribusikan di Aurora SQL Postgre Limitless Database.

rds_aurora.limitless_enable_auto_scale

Memungkinkan penambahan router dan pemisahan pecahan dalam grup pecahan DB.

Untuk informasi selengkapnya, silakan lihat Menambahkan router ke grup pecahan DB dan Memisahkan pecahan dalam grup pecahan DB.

rds_aurora.limitless_finalize_split_shard_mode

Menentukan bagaimana pemisahan pecahan yang diprakarsai sistem diselesaikan. Untuk informasi selengkapnya, lihat Memisahkan pecahan dalam grup pecahan DB.

rds_aurora.limitless_maximum_adaptive_fetch_size

Menetapkan batas atas untuk ukuran pengambilan adaptif. Kisarannya adalah 1 -`INT_MAX`. Default-nya adalah `1000`.

# Menggunakan Database Global Amazon Aurora

Dengan fitur Amazon Aurora Global Database, Anda menyiapkan beberapa cluster Aurora DB yang menjangkau beberapa. Wilayah AWS Aurora secara otomatis menyinkronkan semua perubahan yang dibuat di cluster DB primer ke satu atau lebih cluster sekunder. Database global Aurora memiliki cluster DB primer di satu Wilayah, dan hingga 10 cluster DB sekunder di Wilayah yang berbeda. Konfigurasi Multi-wilayah ini memberikan pemulihan cepat dari pemadaman langka yang mungkin memengaruhi keseluruhan. Wilayah AWS Memiliki salinan lengkap semua data Anda di beberapa lokasi geografis juga memungkinkan operasi baca latensi rendah untuk aplikasi yang terhubung dari lokasi yang terpisah luas di seluruh dunia.

Topik


- Ikhtisar Database Global Amazon Aurora
- Keuntungan dari Amazon Aurora Global Database
- Ketersediaan Wilayah dan versi
- Keterbatasan Basis Data Global Amazon Aurora
- Memulai dengan Amazon Aurora Global Database
- Mengelola basis data global Amazon Aurora
- Menghubungkan ke Amazon Aurora Global Database
- Menggunakan penerusan menulis dalam basis data global Amazon Aurora
- Menggunakan switchover atau failover di Amazon Aurora Global Database
- Memantau basis data global Amazon Aurora
- Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya
- Meningkatkan basis data global Amazon Aurora

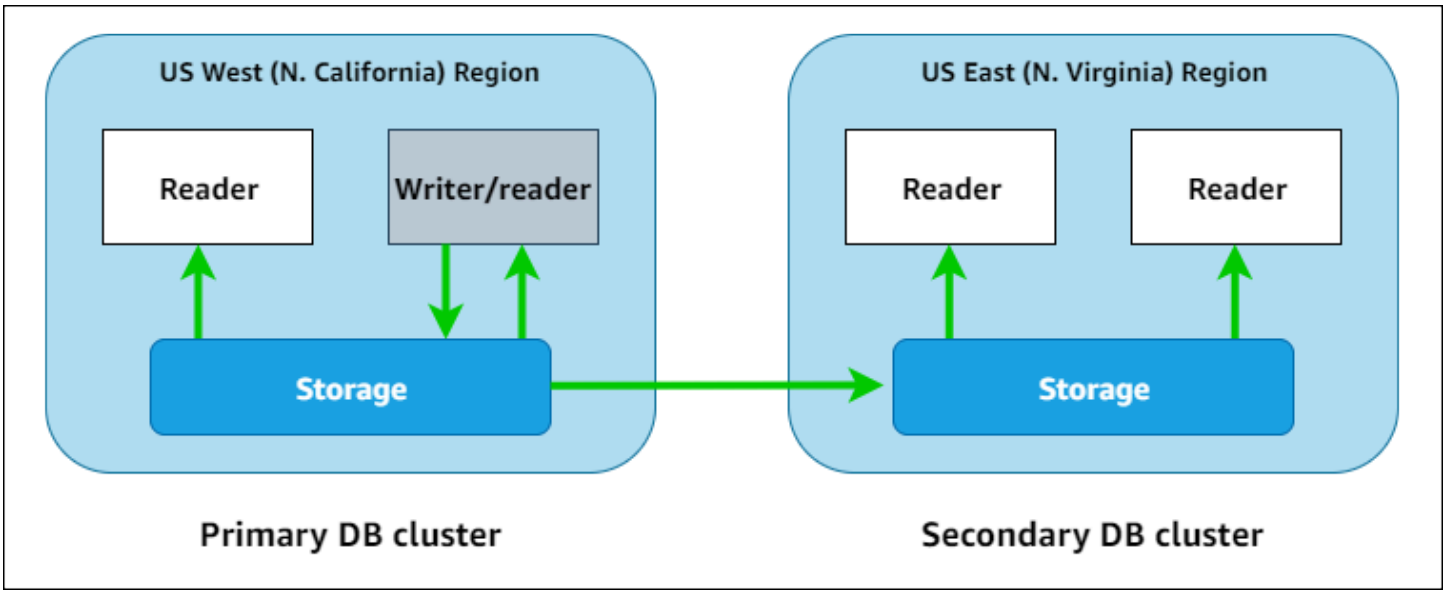

## Ikhtisar Database Global Amazon Aurora

Dengan menggunakan fitur Amazon Aurora Global Database, Anda dapat menjalankan aplikasi terdistribusi secara global menggunakan satu database Aurora yang mencakup beberapa. Wilayah AWS

Basis data global Aurora terdiri dari satu primer Wilayah AWS tempat data Anda ditulis, dan hingga 10 sekunder hanya-baca. Wilayah AWS Anda mengeluarkan operasi tulis ke cluster DB primer di primer Wilayah AWS. Cara paling mudah untuk melakukannya adalah terhubung ke titik akhir penulis

Aurora Global Database, yang selalu menunjuk ke cluster DB utama, bahkan setelah peralihan atau failover ke yang berbeda. Wilayah AWS Setelah operasi tulis apa pun, Aurora mereplikasi data ke sekunder Wilayah AWS menggunakan infrastruktur khusus, dengan latensi biasanya di bawah satu detik.

Dalam diagram berikut, Anda dapat menemukan contoh database global Aurora yang mencakup dua. Wilayah AWS




Anda dapat meningkatkan skala setiap cluster sekunder secara independen, dengan menambahkan satu atau lebih instance pembaca Aurora untuk menyajikan beban kerja hanya-baca. Anda dapat menggunakan instance pembaca Aurora Serverless v2 untuk penskalaan yang lebih granular dan fleksibel.

Hanya klaster primer yang melakukan operasi penulisan. Klien yang melakukan operasi tulis terhubung ke titik akhir penulis Aurora Global Database, yang selalu menunjuk ke instance DB penulis dari cluster utama. Seperti yang ditunjukkan pada diagram, Aurora menggunakan volume penyimpanan cluster dan bukan mesin database untuk replikasi overhead yang cepat dan rendah. Untuk mempelajari selengkapnya, lihat Gambaran umum penyimpanan Amazon Aurora.

Aurora Global Database dirancang untuk aplikasi dengan jejak di seluruh dunia. Cluster DB sekunder hanya-baca dalam beberapa Wilayah AWS membantu mengoptimalkan operasi baca lebih dekat dengan pengguna aplikasi. Dengan menggunakan fitur penerusan tulis, Anda juga dapat mengonfigurasi database global Anda sehingga cluster sekunder mengirim permintaan tulis ke primer. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Aurora Global Database mendukung dua operasi berbeda untuk mengubah Wilayah cluster DB utama Anda, tergantung pada skenario: Aurora Global Database switchover dan Aurora Global Database failover.

- Untuk prosedur operasional yang direncanakan seperti rotasi Regional, gunakan mekanisme switchover (sebelumnya disebut “managed planned failover”). Dengan fitur ini, Anda dapat memindahkan cluster utama dari Aurora Global Database yang sehat ke salah satu Wilayah sekundernya tanpa kehilangan data. Untuk mempelajari selengkapnya, lihat Melakukan switchover untuk basis data global Amazon Aurora.
- Untuk memulihkan Database Global Aurora Anda setelah pemadaman di Wilayah utama, gunakan mekanisme failover. Dengan fitur ini, Anda melakukan failover dari cluster DB utama Anda ke Wilayah lain (Failover lintas wilayah). Untuk mempelajari selengkapnya, lihat Melakukan failover terkelola untuk basis data global Aurora.

## Keuntungan dari Amazon Aurora Global Database

Dengan menggunakan Aurora Global Database, Anda bisa mendapatkan keuntungan sebagai berikut:

- Bacaan global dengan latensi lokal — Jika Anda memiliki kantor di seluruh dunia, Anda dapat menggunakan Aurora Global Database untuk menjaga sumber informasi utama Anda diperbarui di primer. Wilayah AWS Kantor yang berada di Wilayah lain dapat mengakses informasi di Wilayah masing-masing, dengan latensi lokal.
- Cluster DB Aurora sekunder yang dapat diskalakan — Anda dapat menskalakan klaster sekunder Anda dengan menambahkan lebih banyak instance hanya-baca ke sekunder. Wilayah AWS Cluster sekunder hanya baca, sehingga dapat mendukung hingga 16 instans DB hanya-baca daripada batas biasa 15 untuk satu cluster Aurora.
- Replikasi cepat dari cluster DB Aurora primer ke sekunder — Replikasi yang dilakukan oleh Aurora Global Database memiliki dampak kinerja yang kecil pada cluster DB primer. Sumber daya instans DB dikhususkan sepenuhnya untuk melayani beban kerja baca dan tulis aplikasi.
- Pemulihan dari pemadaman di seluruh wilayah — Cluster sekunder memungkinkan Anda membuat Database Global Aurora tersedia di primer baru dengan Wilayah AWS lebih cepat (RTO lebih rendah) dan dengan kehilangan data yang lebih sedikit (RPO lebih rendah) daripada solusi replikasi tradisional.

# Ketersediaan Wilayah dan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah dengan Aurora Global Database, lihat. Daerah yang Didukung dan mesin DB untuk database global Aurora

# Keterbatasan Basis Data Global Amazon Aurora

Batasan berikut saat ini berlaku untuk Aurora Global Database:

- Aurora Global Database tersedia dalam versi Aurora MySQL Wilayah AWS dan Aurora PostgreSQL tertentu dan khusus. Untuk informasi selengkapnya, lihat Daerah yang Didukung dan mesin DB untuk database global Aurora.
- Aurora Global Database memiliki persyaratan konfigurasi khusus untuk kelas instans Aurora DB yang didukung, jumlah Wilayah AWS maksimum, dan sebagainya. Untuk informasi selengkapnya, lihat Persyaratan konfigurasi basis data global Amazon Aurora.
- Untuk Aurora MySQL dengan kompatibilitas MySQL 5.7, switchover Aurora Global Database memerlukan versi 2.09.1 atau versi minor yang lebih tinggi.
- Anda dapat melakukan switchover atau failover lintas wilayah terkelola dengan Aurora Global Database hanya jika cluster DB primer dan sekunder memiliki versi mesin mayor dan minor yang sama. Tergantung pada versi mesin dan mesin, level patch mungkin harus identik atau level patch bisa berbeda. Untuk daftar mesin dan versi mesin yang memungkinkan operasi ini antara cluster primer dan sekunder dengan tingkat patch yang berbeda, lihatKompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola. Jika versi mesin Anda memerlukan level patch yang identik, Anda dapat melakukan failover secara manual dengan mengikuti langkah-langkahnya. Melakukan failover manual untuk basis data global Aurora
- Aurora Global Database saat ini tidak mendukung fitur Aurora berikut:
  - Aurora Serverless v1
  - Backtracking di Aurora
- Untuk batasan penggunaan fitur RDS Proxy dengan Aurora Global Database, lihat. Batasan untuk Proksi RDS dengan basis data global
- Upgrade versi minor otomatis tidak berlaku untuk klaster Aurora MySQL dan Aurora PostgreSQL yang merupakan bagian dari database global. Perlu diketahui bahwa Anda dapat menentukan

pengaturan ini untuk instans DB yang merupakan bagian dari klaster basis data global, tetapi pengaturan tersebut tidak akan berpengaruh.

- Aurora Global Database saat ini tidak mendukung Auto Scaling Aurora untuk cluster DB sekunder.
- Untuk menggunakan Database Activity Streams (DAS) pada Aurora Global Database yang menjalankan Aurora MySQL 5.7, versi mesin harus versi 2.08 atau lebih tinggi. Untuk informasi tentang DAS, lihatMemantau Amazon Aurora dengan Aliran Aktivitas Basis Data.
- Batasan berikut saat ini berlaku untuk meningkatkan Aurora Global Database:
  - Anda tidak dapat menerapkan grup parameter kustom ke klaster basis data global saat melakukan peningkatan versi utama dari basis data global Aurora tersebut. Anda membuat grup parameter kustom di setiap Wilayah klaster global dan menerapkannya secara manual ke klaster Regional setelah melakukan peningkatan.
  - Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan in-place dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan. Untuk informasi selengkapnya tentang metode yang dapat Anda gunakan, lihat Peningkatan versi utama.
  - Dengan Aurora Global Database, Anda tidak dapat melakukan upgrade versi utama dari mesin Aurora PostgreSQL DB jika fitur tujuan titik pemulihan (RPO) diaktifkan. Untuk informasi tentang fitur RPO, lihat Mengelola RPOs database global berbasis Aurora PostgreSQL.
  - Dengan Aurora Global Database, Anda tidak dapat melakukan upgrade versi minor dari Aurora MySQL versi 3.01 atau 3.02 ke 3.03 atau lebih tinggi dengan menggunakan proses standar. Untuk detail tentang proses yang akan digunakan, lihat Meningkatkan Aurora MySQL dengan mengubah versi mesin.

  Untuk informasi tentang memutakhirkan Basis Data Global Aurora, lihat. Meningkatkan basis data global Amazon Aurora
- Anda tidak dapat menghentikan atau memulai cluster Aurora DB di database global Anda satu per satu. Untuk mempelajari selengkapnya, lihat Menghentikan dan memulai klaster DB Amazon Aurora.
- Instans DB pembaca Aurora yang dilampirkan ke cluster Aurora DB sekunder dapat dimulai ulang dalam keadaan tertentu. Jika instans DB Wilayah AWS penulis primer mengalami restart atau failover, instance DB pembaca di Wilayah sekunder juga dimulai ulang. Cluster sekunder kemudian tidak tersedia sampai semua instance DB pembaca kembali sinkron dengan instance penulis cluster DB primer. Perilaku cluster primer saat me-reboot atau selama failover sama dengan cluster DB nonglobal tunggal. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

Pastikan Anda memahami dampaknya terhadap basis data global Anda sebelum membuat perubahan pada cluster DB utama Anda. Untuk mempelajari selengkapnya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

- Aurora Global Database saat ini tidak mendukung `inaccessible-encryption-credentials-recoverable` status ketika Amazon Aurora kehilangan akses ke kunci untuk cluster AWS KMS DB. Dalam kasus ini, klaster DB yang terenkripsi akan langsung berstatus `inaccessible-encryption-credentials` terminal. Untuk informasi selengkapnya tentang status ini, lihat Melihat status klaster DB.
- Secrets Manager tidak mendukung Aurora Global Database. Saat menambahkan Region ke database global, Anda harus menonaktifkan integrasi Secrets Manager untuk instans DB terlebih dahulu.
- Cluster DB berbasis Aurora PostgreSQL yang menggunakan Aurora Global Database memiliki batasan sebagai berikut:
  - Manajemen cache cluster tidak didukung untuk klaster DB sekunder Aurora PostgreSQL yang merupakan bagian dari database global Aurora.
  - Jika klaster DB utama database global Anda didasarkan pada replika instans Amazon RDS PostgreSQL, Anda tidak dapat membuat klaster sekunder. Jangan mencoba membuat sekunder dari cluster itu menggunakan operasi AWS Management Console, the AWS CLI, atau `CreateDBCluster` API. Upaya tersebut akan kehabisan waktu, dan klaster sekunder tidak akan dibuat.

Kami menyarankan Anda membuat cluster DB sekunder untuk database global Anda dengan menggunakan versi mesin Aurora DB yang sama dengan yang utama. Lihat informasi yang lebih lengkap di Membuat basis data global Amazon Aurora.

## Memulai dengan Amazon Aurora Global Database

Untuk memulai dengan Aurora Global Database, pertama-tama tentukan mesin Aurora DB mana yang ingin Anda gunakan dan di mana. Wilayah AWS Hanya versi spesifik dari SQL mesin basis data Aurora My dan SQL Aurora Postgre yang mendukung Aurora Global Database tertentu. Wilayah AWS Untuk daftar lengkapnya, lihat Daerah yang Didukung dan mesin DB untuk database global Aurora.

Anda dapat membuat Database Global Aurora dengan salah satu cara berikut:

- Buat Database Global Aurora baru dengan cluster Aurora DB baru dan instans Aurora DB — Anda dapat melakukannya dengan mengikuti langkah-langkah di. Membuat basis data global Amazon Aurora Setelah Anda membuat cluster Aurora DB primer, Anda kemudian menambahkan sekunder Wilayah AWS dengan mengikuti langkah-langkah di. Menambahkan Wilayah AWS ke database global Amazon Aurora
- Gunakan cluster Aurora DB yang ada yang mendukung fitur Aurora Global Database dan tambahkan Wilayah AWS ke dalamnya — Anda dapat melakukan ini hanya jika cluster Aurora DB Anda yang ada menggunakan versi mesin DB yang kompatibel dengan global.

  Periksa apakah Anda dapat memilih Tambahkan wilayah untuk Tindakan pada AWS Management Console saat klaster Aurora DB Anda dipilih. Jika Anda dapat memilihnya, Anda dapat menggunakan klaster DB Aurora tersebut untuk klaster global Aurora Anda. Untuk informasi selengkapnya, lihat Menambahkan Wilayah AWS ke database global Amazon Aurora.

Sebelum membuat Database Global Aurora, kami sarankan Anda memahami semua persyaratan konfigurasi.

Topik

- Persyaratan konfigurasi basis data global Amazon Aurora
- Membuat basis data global Amazon Aurora
- Menambahkan Wilayah AWS ke database global Amazon Aurora
- Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder
- Membuat database global Amazon Aurora dari snapshot Aurora atau Amazon RDS

## Persyaratan konfigurasi basis data global Amazon Aurora

Sebelum Anda mulai bekerja dengan database global Anda, rencanakan nama cluster dan instance, AWS Regions, jumlah instance dan kelas instance yang ingin Anda gunakan. Pastikan pilihan Anda sesuai dengan persyaratan konfigurasi berikut.

Database global Aurora mencakup setidaknya dua. Wilayah AWS Primer Wilayah AWS mendukung cluster Aurora DB yang memiliki satu penulis Aurora DB instance. Sekunder Wilayah AWS menjalankan cluster Aurora DB read-only yang seluruhnya terdiri dari Aurora Replicas. Setidaknya satu sekunder Wilayah AWS diperlukan, tetapi database global Aurora dapat memiliki hingga 10 sekunder. Wilayah AWS Tabel ini mencantumkan jumlah maksimum klaster DB Aurora, instans DB Aurora, dan Replika Aurora yang diperbolehkan dalam basis data global Aurora.

| Deskripsi | Primer Wilayah AWS | Sekunder Wilayah AWS |
|---|---|---|
| Klaster DB Aurora | 1 | 10 (maksimum) |
| Instans penulis | 1 | 0 |
| Instans hanya-baca (replika Aurora), per klaster DB Aurora | 15 (maks) | 16 (total) |
| Instans hanya-baca (maks. yang diperbolehkan, dengan jumlah aktual dari Wilayah sekunder) | 15 - s | s = jumlah total sekunder Wilayah AWS |

Klaster DB Aurora yang membentuk basis data global Aurora memiliki persyaratan khusus berikut:

- Persyaratan kelas instans DB — Sebuah basis data global Aurora membutuhkan kelas instans DB yang dioptimalkan untuk aplikasi sarat memori. Untuk informasi tentang kelas instans DB yang dioptimalkan memori, lihatJenis kelas instans DB. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi.
- Wilayah AWS persyaratan — Database global Aurora membutuhkan cluster Aurora DB primer dalam satu Wilayah AWS, dan setidaknya satu cluster Aurora DB sekunder di Wilayah yang berbeda. Anda dapat membuat hingga 10 cluster Aurora DB sekunder (hanya-baca), dan masing-masing harus berada di Wilayah yang berbeda. Dengan kata lain, tidak ada dua klaster DB Aurora dalam basis data global Aurora dapat berada di Wilayah AWS yang sama.

  Untuk informasi tentang kombinasi mesin Aurora DB, versi mesin, dan yang dapat Wilayah AWS Anda gunakan dengan Aurora Global Database, lihat. Daerah yang Didukung dan mesin DB untuk database global Aurora
- Persyaratan penamaan — Nama yang Anda pilih untuk setiap cluster Aurora DB Anda harus unik, di semua. Wilayah AWS Anda tidak dapat menggunakan nama yang sama untuk klaster DB Aurora yang berbeda meskipun berada di Wilayah yang berbeda.
- Persyaratan kapasitas untuk Aurora Serverless v2 — Untuk database global denganAurora Serverless v2, kapasitas minimum yang direkomendasikan untuk cluster DB di primer Wilayah AWS adalah 8 ACUs.

Sebelum Anda dapat mengikuti prosedur di bagian ini, Anda memerlukan Akun AWS. Lengkapi tugas penyiapan untuk menggunakan Amazon Aurora. Untuk informasi selengkapnya, lihat Menyiapkan lingkungan Anda untuk Amazon Aurora. Anda juga perlu menyelesaikan langkah awal lainnya untuk membuat klaster DB Aurora. Untuk mempelajari selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Ketika Anda siap untuk mengatur database global Anda, lihat prosedur Membuat basis data global Amazon Aurora untuk membuat semua sumber daya yang diperlukan. Anda juga dapat mengikuti prosedur Menambahkan Wilayah AWS ke database global Amazon Aurora untuk membuat database global menggunakan cluster Aurora yang ada sebagai cluster utama.

## Membuat basis data global Amazon Aurora

Untuk membuat database global Aurora dan sumber daya yang terkait dengan menggunakan AWS Management Console, AWS CLI, atau RDSAPI, gunakan langkah-langkah berikut.

> **Note**
>
> Jika Anda memiliki cluster Aurora DB yang menjalankan mesin database Aurora yang kompatibel dengan global, Anda dapat menggunakan bentuk yang lebih pendek dari prosedur ini. Dalam hal ini, Anda dapat menambahkan yang lain Wilayah AWS ke cluster DB yang ada untuk membuat database global Aurora Anda. Untuk melakukannya, lihat Menambahkan Wilayah AWS ke database global Amazon Aurora.

Konsol

Langkah-langkah untuk membuat database global Aurora dimulai dengan masuk ke Wilayah AWS yang mendukung fitur database global Aurora. Untuk daftar lengkap, lihat Daerah yang Didukung dan mesin DB untuk database global Aurora.

Salah satu langkah berikut adalah memilih virtual private cloud (VPC) berbasis Amazon VPC untuk cluster Aurora DB Anda. Untuk menggunakan milik Anda sendiriVPC, kami sarankan Anda membuatnya terlebih dahulu sehingga tersedia untuk Anda pilih. Pada saat yang sama, buat subnet terkait apa pun, serta grup subnet dan grup keamanan sesuai keperluan. Untuk mempelajari caranya, lihat Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .

Untuk informasi umum tentang cara membuat klaster DB Aurora klaster, lihat Membuat klaster DB Amazon Aurora.

Membuat basis data global Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Buat basis data. Di halaman Buat basis data, lakukan hal berikut:
   - Untuk metode pembuatan basis data, pilih Pembuatan Standar. (Jangan pilih Pembuatan mudah.)
   - Untuk `Engine type` di bagian Opsi mesin, pilih jenis mesin yang berlaku, Aurora (SQLKompatibel Saya) atau Aurora (Kompatibel dengan Postgre). SQL
3. Lanjutkan membuat basis data global Aurora Anda dengan menggunakan langkah-langkah dari prosedur berikut.

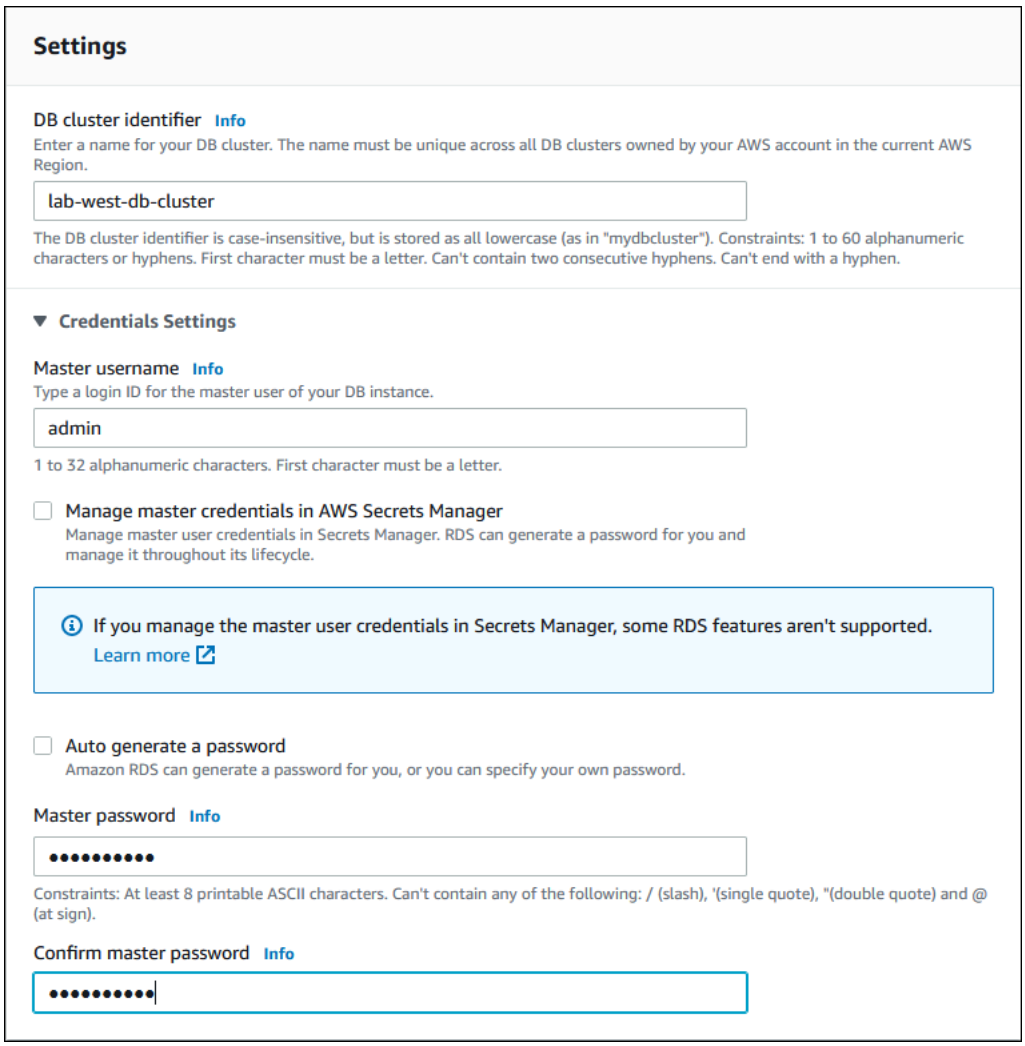
Membuat database global menggunakan Aurora My SQL

Langkah-langkah berikut berlaku untuk semua versi Aurora My. SQL

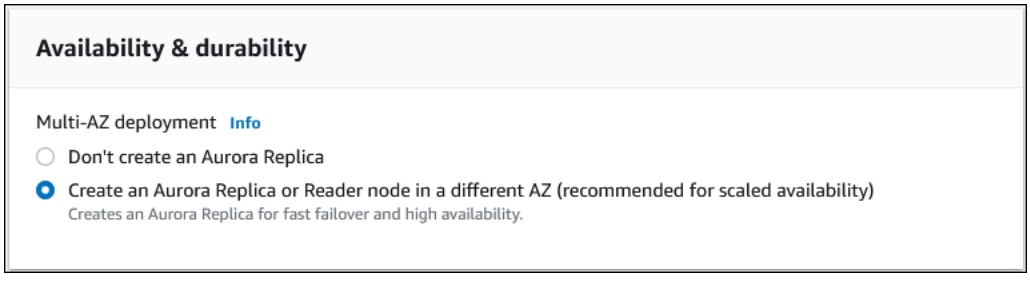
Untuk membuat database global Aurora menggunakan Aurora My SQL

Lengkapi halaman Buat basis data.

1. Untuk Opsi mesin, pilih salah satu hal berikut:
   - Untuk versi Engine, pilih versi Aurora My SQL yang ingin Anda gunakan untuk database global Aurora Anda.
2. Untuk Templat, pilih Produksi. Atau Anda dapat memilih Dev/Test if appropriate for your use case. Don't use Dev/Test di lingkungan produksi.
3. Untuk Pengaturan, lakukan hal berikut:
   a. Masukkan nama yang bermakna untuk pengidentifikasi klaster DB. Ketika Anda selesai membuat basis data global Aurora, nama ini mengidentifikasi klaster DB primer.
   b. Masukkan kata sandi Anda sendiri untuk akun pengguna `admin` untuk instans DB tersebut, atau minta Aurora untuk membuatkannya untuk Anda. Jika Anda memilih untuk membuat kata sandi secara otomatis, Anda akan mendapatkan opsi untuk menyalin kata sandi.




4. Untuk Kelas instans DB, pilih `db.r5.large` atau kelas instans DB memori yang dioptimalkan lainnya. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi.
5. Untuk Ketersediaan & daya tahan, kami sarankan Anda memilih untuk meminta Aurora membuatkan Replika Aurora di Zona Ketersediaan (AZ) yang berbeda untuk Anda. Jika Anda tidak membuat Replika Aurora sekarang, Anda harus melakukannya nanti.




6. Untuk Konektivitas, pilih virtual private cloud (VPC) berbasis Amazon VPC yang mendefinisikan lingkungan jaringan virtual untuk instans DB ini. Anda dapat memilih default untuk menyederhanakan tugas ini.

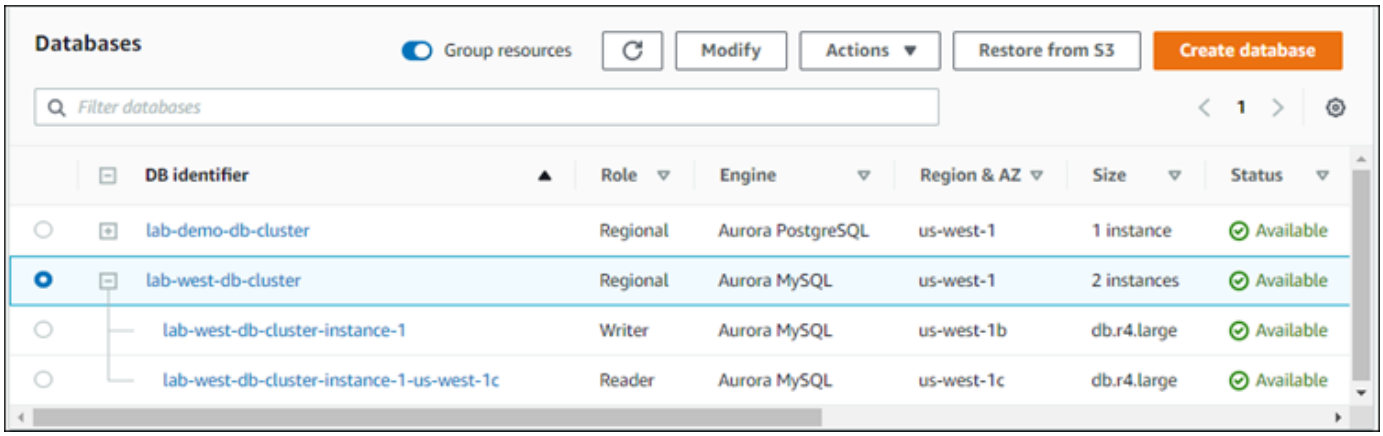
7. Lengkapi pengaturan Autentikasi basis data. Untuk menyederhanakan proses, Anda dapat memilih otentikasi kata sandi sekarang dan mengatur AWS Identity and Access Management (IAM) nanti.
8. Pada Konfigurasi tambahan, lakukan hal berikut:

   a. Masukkan nama untuk Nama basis data awal untuk membuat instans DB Aurora primer untuk klaster ini. Ini adalah simpul penulis untuk klaster DB Aurora primer.

      Tetap pilih default untuk grup parameter klaster DB dan grup parameter DB, kecuali Anda memiliki grup parameter kustom Anda sendiri yang ingin Anda gunakan.

   b. Kosongkan kotak centang Aktifkan lacak mundur jika dipilih. Basis data global Aurora tidak mendukung pelacakan mundur. Jika tidak, terima pengaturan default lainnya untuk Konfigurasi tambahan.
9. Pilih Buat basis data.

   Mungkin diperlukan beberapa menit bagi Aurora untuk menyelesaikan proses pembuatan instans DB Aurora, Replika Aurora, dan klaster DB Aurora. Anda akan tahu ketika klaster DB Aurora siap untuk digunakan sebagai klaster DB primer dalam basis data global Aurora berdasarkan statusnya. Ketika sudah siap, statusnya dan status simpul penulis serta replika adalah Tersedia, seperti yang ditunjukkan berikut ini.



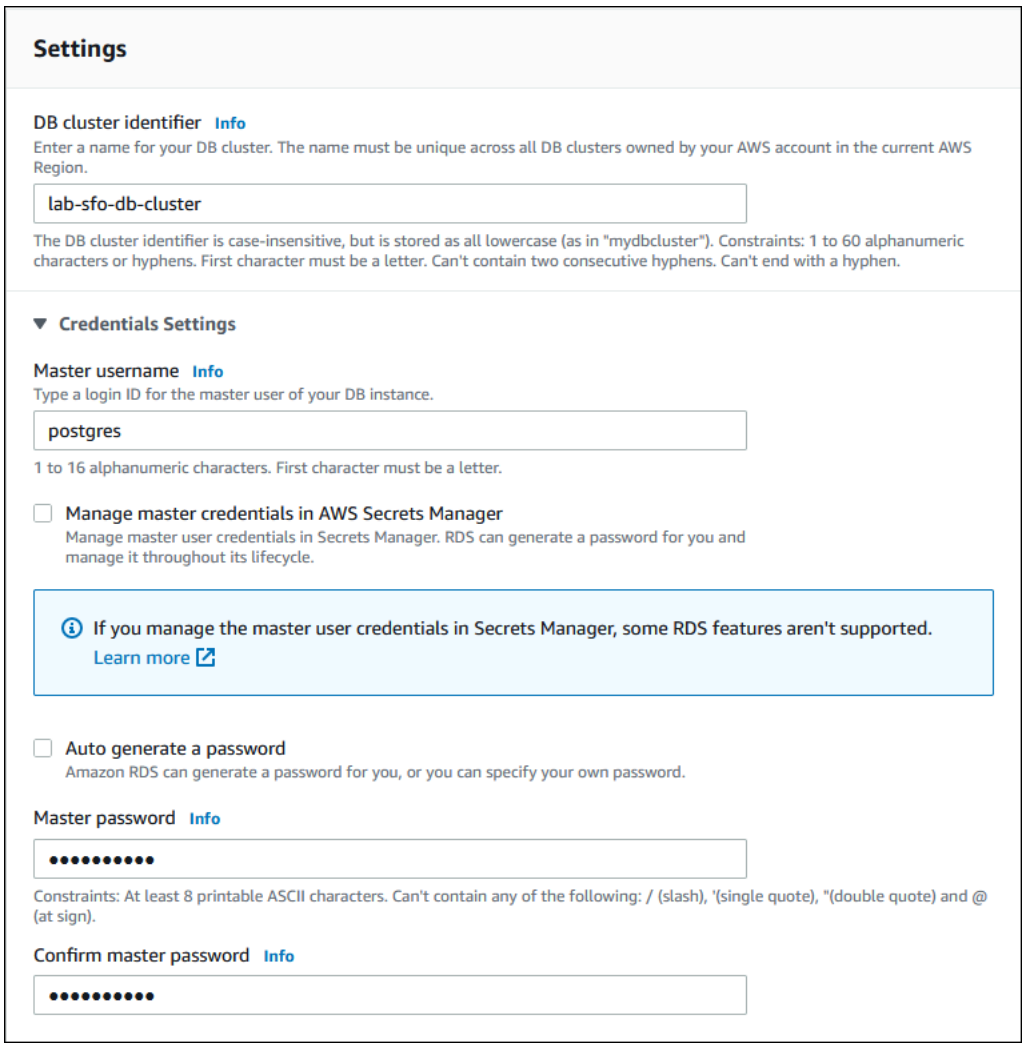

Ketika klaster DB primer Anda tersedia, buat basis data global Aurora dengan menambahkan klaster sekunder untuk itu. Untuk melakukannya, ikuti langkah-langkah di Menambahkan Wilayah AWS ke database global Amazon Aurora.

### Membuat database global menggunakan Aurora Postgre SQL

Untuk membuat database global Aurora menggunakan Aurora Postgre SQL

Lengkapi halaman Buat basis data.

1. Untuk Opsi mesin, pilih salah satu hal berikut:
   - Untuk versi Engine, pilih versi Aurora Postgre SQL yang ingin Anda gunakan untuk database global Aurora Anda.
2. Untuk Templat, pilih Produksi. Atau Anda dapat memilih Dev/Test if appropriate. Don't use Dev/Test di lingkungan produksi.
3. Untuk Pengaturan, lakukan hal berikut:
   a. Masukkan nama yang bermakna untuk pengidentifikasi klaster DB. Ketika Anda selesai membuat basis data global Aurora, nama ini mengidentifikasi klaster DB primer.
   b. Masukkan kata sandi Anda sendiri untuk akun admin default untuk klaster DB, atau minta Aurora membuatkannya untuk Anda. Jika Anda memilih untuk membuat kata sandi secara otomatis, Anda akan mendapatkan opsi untuk menyalin kata sandi.




4. Untuk Kelas instans DB, pilih `db.r5.large` atau kelas instans DB memori yang dioptimalkan lainnya. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi.
5. Untuk Ketersediaan & daya tahan, kami sarankan Anda memilih untuk meminta Aurora membuatkan Replika Aurora di AZ yang berbeda untuk Anda. Jika Anda tidak membuat Replika Aurora sekarang, Anda harus melakukannya nanti.
6. Untuk Konektivitas, pilih virtual private cloud (VPC) berbasis Amazon VPC yang mendefinisikan lingkungan jaringan virtual untuk instans DB ini. Anda dapat memilih default untuk menyederhanakan tugas ini.
7. (Opsional) Lengkapi pengaturan Autentikasi basis data. Autentikasi kata sandi selalu diaktifkan. Untuk menyederhanakan proses, Anda dapat melewati bagian ini dan mengatur IAM atau kata sandi dan otentikasi Kerberos nanti.
8. Pada Konfigurasi tambahan, lakukan hal berikut:

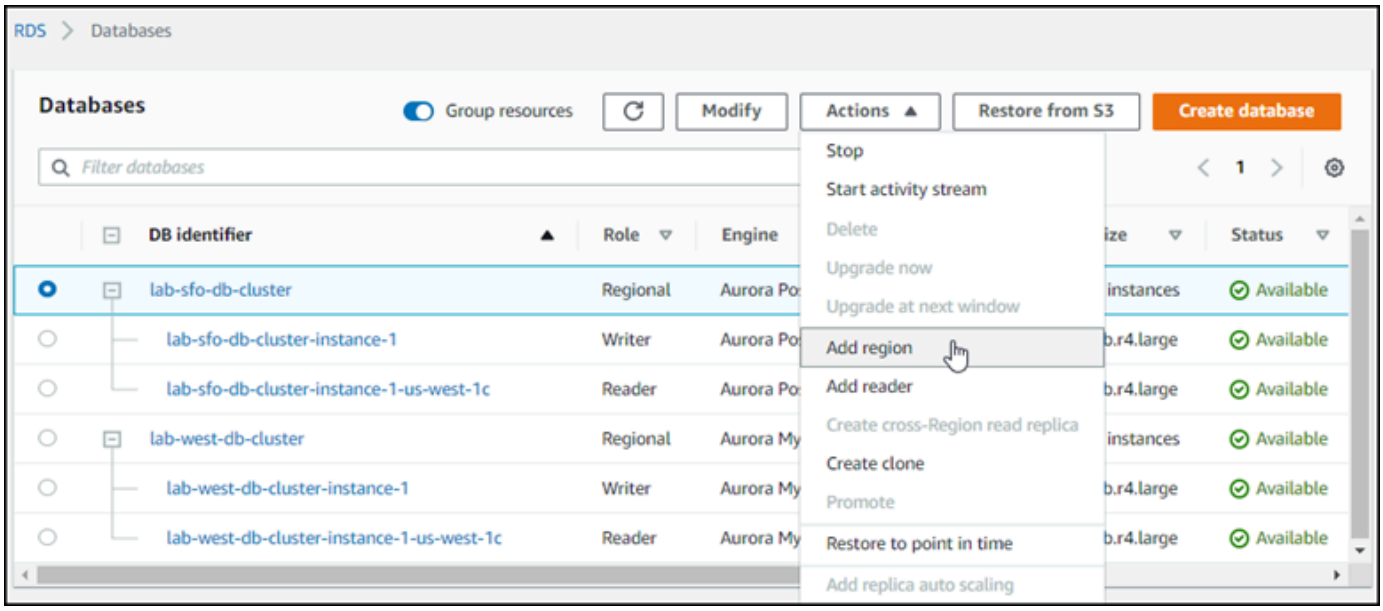
a. Masukkan nama untuk Nama basis data awal untuk membuat instans DB Aurora primer untuk klaster ini. Ini adalah simpul penulis untuk klaster DB Aurora primer.

   Tetap pilih default untuk grup parameter klaster DB dan grup parameter DB, kecuali Anda memiliki grup parameter kustom Anda sendiri yang ingin Anda gunakan.

b. Terima semua pengaturan default lainnya untuk Konfigurasi tambahan, seperti Enkripsi, Ekspor log, dan sebagainya.

9. Pilih Buat basis data.

   Mungkin diperlukan beberapa menit bagi Aurora untuk menyelesaikan proses pembuatan instans DB Aurora, Replika Aurora, dan klaster DB Aurora. Ketika klaster siap untuk digunakan, klaster DB Aurora beserta simpul penulis dan replikanya akan menampilkan status Tersedia. Ini menjadi klaster DB Aurora primer dari basis data global Aurora Anda, setelah Anda menambahkan klaster sekunder.




Saat klaster DB primer Anda tersedia, buat satu atau beberapa klaster sekunder dengan mengikuti langkah-langkah dalam Menambahkan Wilayah AWS ke database global Amazon Aurora.

AWS CLI

AWS CLI Perintah dalam prosedur berikut menyelesaikan tugas-tugas berikut:

1. Buat basis data global Aurora, dengan memberikan nama dan menentukan jenis mesin basis data Aurora yang ingin Anda gunakan.

2. Buat klaster DB Aurora primer untuk basis data global Aurora.
3. Buat instans DB Aurora untuk klaster. Ini adalah klaster DB Aurora primer untuk basis data global.
4. Buat instans DB Aurora kedua untuk klaster DB Aurora. Ini adalah pembaca untuk melengkapi klaster DB Aurora.
5. Buat klaster DB Aurora kedua di Wilayah lain kemudian tambahkan ke basis data global Aurora Anda, dengan mengikuti langkah-langkah di Menambahkan Wilayah AWS ke database global Amazon Aurora.

Ikuti prosedur untuk mesin basis data Aurora Anda.

Membuat database global menggunakan Aurora My SQL

Untuk membuat database global Aurora menggunakan Aurora My SQL

1. Gunakan `create-global-cluster` CLI perintah, melewati nama Wilayah AWS, mesin database Aurora, dan versi.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-mysql \
       --engine-version version # optional
   ```

   Untuk Windows:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-mysql ^
       --engine-version version # optional
   ```

   Tindakan ini menciptakan basis data global Aurora “kosong”, hanya dengan nama (pengidentifikasi) dan mesin basis data Aurora. Perlu waktu beberapa menit agar basis data global Aurora tersedia. Sebelum pergi ke langkah berikutnya, gunakan `describe-global-clusters` CLI perintah untuk melihat apakah itu tersedia.

   ```
   aws rds describe-global-clusters --region primary_region --global-cluster-
   identifier global_database_id
   ```

Ketika basis data global Aurora tersedia, Anda dapat membuat klaster DB Aurora primer.

2. Untuk membuat cluster Aurora DB primer, gunakan perintah. `create-db-cluster` CLI Sertakan nama basis data global Aurora Anda dengan menggunakan parameter `--global-cluster-identifier`.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-mysql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   Untuk Windows:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-mysql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   Gunakan `describe-db-clusters` AWS CLI perintah untuk mengonfirmasi bahwa cluster Aurora DB sudah siap. Untuk memilih sebuah klaster DB Aurora tertentu, gunakan parameter `--db-cluster-identifier`. Atau Anda dapat meninggalkan nama klaster DB Aurora dalam perintah untuk mendapatkan detail tentang semua klaster DB Aurora Anda di Wilayah tertentu.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

   Ketika respons menunjukkan `"Status": "available"` untuk klaster, berarti klaster sudah siap untuk digunakan.

3. Buat instans DB untuk klaster DB Aurora primer Anda. Untuk melakukannya, gunakan `create-db-instance` CLI perintah. Berikan nama klaster DB Aurora Anda dalam perintah, dan tentukan detail konfigurasi untuk instans. Anda tidak perlu memasukkan parameter `--master-username` dan `--master-user-password` dalam perintah, karena parameter tersebut diperoleh dari klaster DB Aurora.

   Untuk `--db-instance-class`, Anda hanya dapat menggunakan dari kelas dengan memori yang dioptimalkan, seperti `db.r5.large`. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi. Untuk informasi tentang kelas-kelas ini, lihatJenis kelas instans DB.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance \
     --db-cluster-identifier primary_db_cluster_id \
     --db-instance-class instance_class \
     --db-instance-identifier db_instance_id \
     --engine aurora-mysql \
     --engine-version version \
     --region primary_region
   ```

   Untuk Windows:

   ```
   aws rds create-db-instance ^
     --db-cluster-identifier primary_db_cluster_id ^
     --db-instance-class instance_class ^
     --db-instance-identifier db_instance_id ^
     --engine aurora-mysql ^
     --engine-version version ^
     --region primary_region
   ```

   Operasi `create-db-instance` mungkin membutuhkan beberapa waktu untuk diselesaikan. Periksa status untuk melihat apakah instans Aurora DB tersedia sebelum melanjutkan.

   ```
   aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
   ```

   Ketika perintah mengembalikan status`available`, Anda dapat membuat instance Aurora DB lain untuk cluster DB utama Anda. Ini adalah instans pembaca (Aurora Replica) untuk klaster DB Aurora.

4. Untuk membuat instance Aurora DB lain untuk cluster, gunakan perintah. `create-db-instance` CLI

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-mysql
```

   Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-mysql
```

Ketika instans DB tersedia, replikasi dimulai dari simpul penulis ke replika. Sebelum melanjutkan, periksa apakah instans DB tersedia dengan `describe-db-instances` CLI perintah.

Pada titik ini, Anda memiliki basis data global Aurora dengan klaster DB Aurora primer yang berisi instans DB penulis dan Replika Aurora. Anda sekarang dapat menambahkan klaster DB Aurora hanya-baca di Wilayah yang berbeda untuk menyelesaikan basis data global Aurora Anda. Untuk melakukannya, ikuti langkah yang ada di Menambahkan Wilayah AWS ke database global Amazon Aurora.

Membuat database global menggunakan Aurora Postgre SQL

Ketika Anda membuat objek Aurora untuk basis data global Aurora dengan menggunakan perintah berikut, mungkin diperlukan beberapa menit sampai setiap objek tersedia. Kami merekomendasikan bahwa setelah menyelesaikan perintah tertentu, Anda memeriksa status objek Aurora tertentu untuk memastikan statusnya tersedia.

Untuk melakukannya, gunakan `describe-global-clusters` CLI perintah.

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

Untuk membuat database global Aurora menggunakan Aurora Postgre SQL

1. Gunakan perintah `create-global-cluster` CLI.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-postgresql \
       --engine-version version # optional
   ```

   Untuk Windows:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-postgresql ^
       --engine-version version # optional
   ```

   Ketika basis data global Aurora tersedia, Anda dapat membuat klaster DB Aurora primer.

2. Untuk membuat cluster Aurora DB primer, gunakan perintah. `create-db-cluster` CLI Sertakan nama basis data global Aurora Anda dengan menggunakan parameter `--global-cluster-identifier`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster \
     --region primary_region \
     --db-cluster-identifier primary_db_cluster_id \
     --master-username userid \
     --master-user-password password \
     --engine aurora-postgresql \
     --engine-version version \
     --global-cluster-identifier global_database_id
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster ^
     --region primary_region ^
     --db-cluster-identifier primary_db_cluster_id ^
     --master-username userid ^
   ```

```
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

Periksa apakah klaster DB Aurora sudah siap. Ketika respons dari perintah berikut menunjukkan `"Status": "available"` untuk klaster DB Aurora, Anda dapat melanjutkan.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. Buat instans DB untuk klaster DB Aurora primer Anda. Untuk melakukannya, gunakan `create-db-instance` CLI perintah.

   Loloskan nama klaster DB Aurora Anda dengan parameter `--db-cluster-identifier`.

   Anda tidak perlu memasukkan parameter `--master-username` dan `--master-user-password` dalam perintah, karena parameter tersebut diperoleh dari klaster DB Aurora.

   Untuk `--db-instance-class`, Anda hanya dapat menggunakan dari kelas dengan memori yang dioptimalkan, seperti `db.r5.large`. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi. Untuk informasi tentang kelas-kelas ini, lihatJenis kelas instans DB.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

   Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-postgresql ^
  --engine-version version ^
```

```
  --region primary_region
```

4. Periksa status instans Aurora DB sebelum melanjutkan.

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

Jika respons menunjukkan bahwa status instans Aurora DB adalah`available`, Anda dapat membuat instans Aurora DB lain untuk cluster DB utama Anda.

5. Untuk membuat Aurora Replica untuk cluster Aurora DB, gunakan perintah. `create-db-instance` CLI

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

Ketika instans DB tersedia, replikasi dimulai dari simpul penulis ke replika. Sebelum melanjutkan, periksa apakah instans DB tersedia dengan `describe-db-instances` CLI perintah.

Basis data global Aurora Anda ada, tetapi hanya memiliki Wilayah primernya dengan klaster DB Aurora yang terdiri dari instans DB penulis dan Replika Aurora. Anda sekarang dapat menambahkan klaster DB Aurora hanya-baca di Wilayah yang berbeda untuk menyelesaikan basis data global Aurora Anda. Untuk melakukannya, ikuti langkah yang ada di Menambahkan Wilayah AWS ke database global Amazon Aurora.

RDS API

Untuk membuat database global Aurora dengan RDSAPI, jalankan operasi. CreateGlobalCluster

# Menambahkan Wilayah AWS ke database global Amazon Aurora

Anda dapat menggunakan prosedur berikut untuk menambahkan cluster sekunder tambahan ke database global yang ada. Anda juga dapat membuat database global dari cluster Aurora DB mandiri dengan menggunakan prosedur ini untuk menambahkan Wilayah sekunder pertama AWS .

Database global Aurora membutuhkan setidaknya satu cluster Aurora DB sekunder yang berbeda Wilayah AWS dari cluster DB Aurora primer. Anda dapat melampirkan hingga 10 cluster DB sekunder ke database global Aurora Anda. Ulangi prosedur berikut untuk setiap cluster DB sekunder baru. Untuk setiap klaster DB sekunder yang Anda tambahkan ke basis data global Aurora Anda, kurangi jumlah Replika Aurora yang diperbolehkan untuk klaster DB primer sebanyak satu buah.

Misalnya, jika database global Aurora Anda memiliki 10 Wilayah sekunder, cluster DB primer Anda hanya dapat memiliki 5 (bukan 15) Replika Aurora. Untuk informasi selengkapnya, lihat Persyaratan konfigurasi basis data global Amazon Aurora.

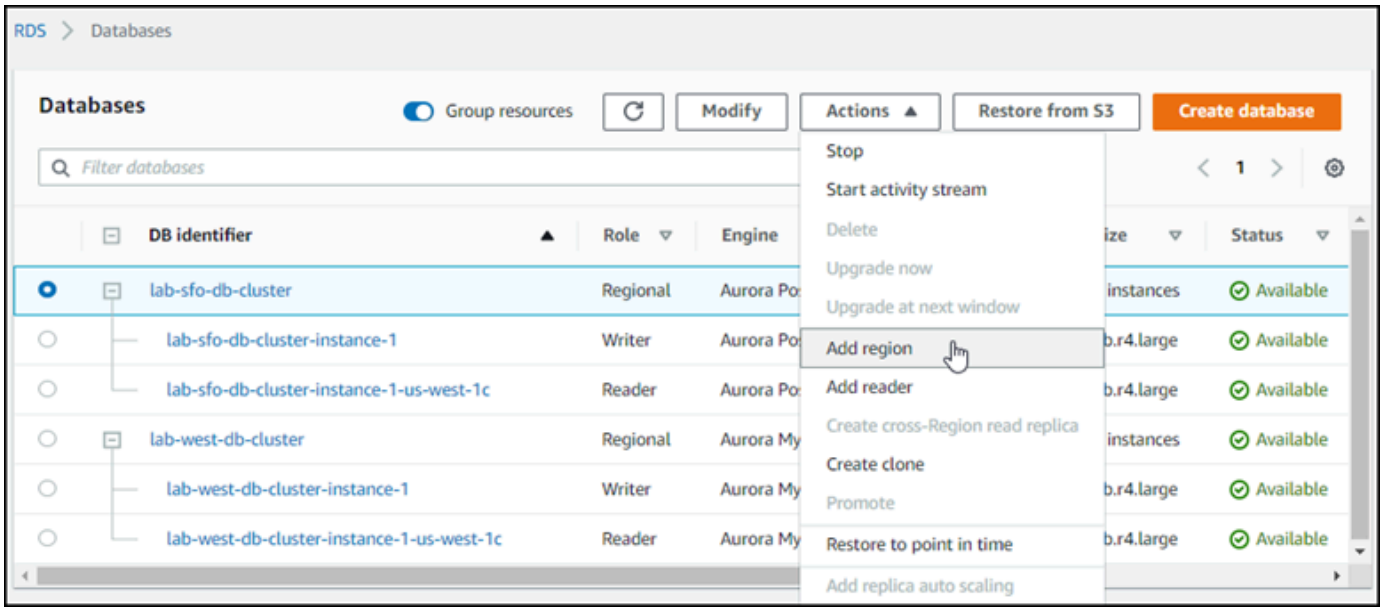
Jumlah Replika Aurora (instans pembaca) di klaster DB primer menentukan jumlah klaster DB sekunder yang dapat Anda tambahkan. Jumlah total instans pembaca dalam klaster DB primer ditambah jumlah klaster sekunder tidak boleh lebih dari 15. Misalnya, jika Anda memiliki 14 instans pembaca di klaster DB primer dan 1 klaster sekunder, Anda tidak dapat menambahkan klaster sekunder lainnya ke basis data global.

> **Note**
>
> Untuk Aurora MySQL versi 3, saat Anda membuat klaster sekunder, pastikan nilai `lower_case_table_names` cocok dengan nilai di klaster primer. Pengaturan ini adalah parameter basis data yang memengaruhi cara server menangani sensitivitas huruf besar/kecil pengidentifikasi. Untuk informasi selengkapnya tentang parameter basis data, lihat .
> Kami menyarankan ketika Anda membuat klaster sekunder, gunakan versi mesin DB yang sama untuk primer dan sekunder. Jika perlu, tingkatkan klaster primer menjadi versi yang sama dengan versi sekunder. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola.

Konsol

Untuk menambahkan Wilayah AWS ke database global Aurora

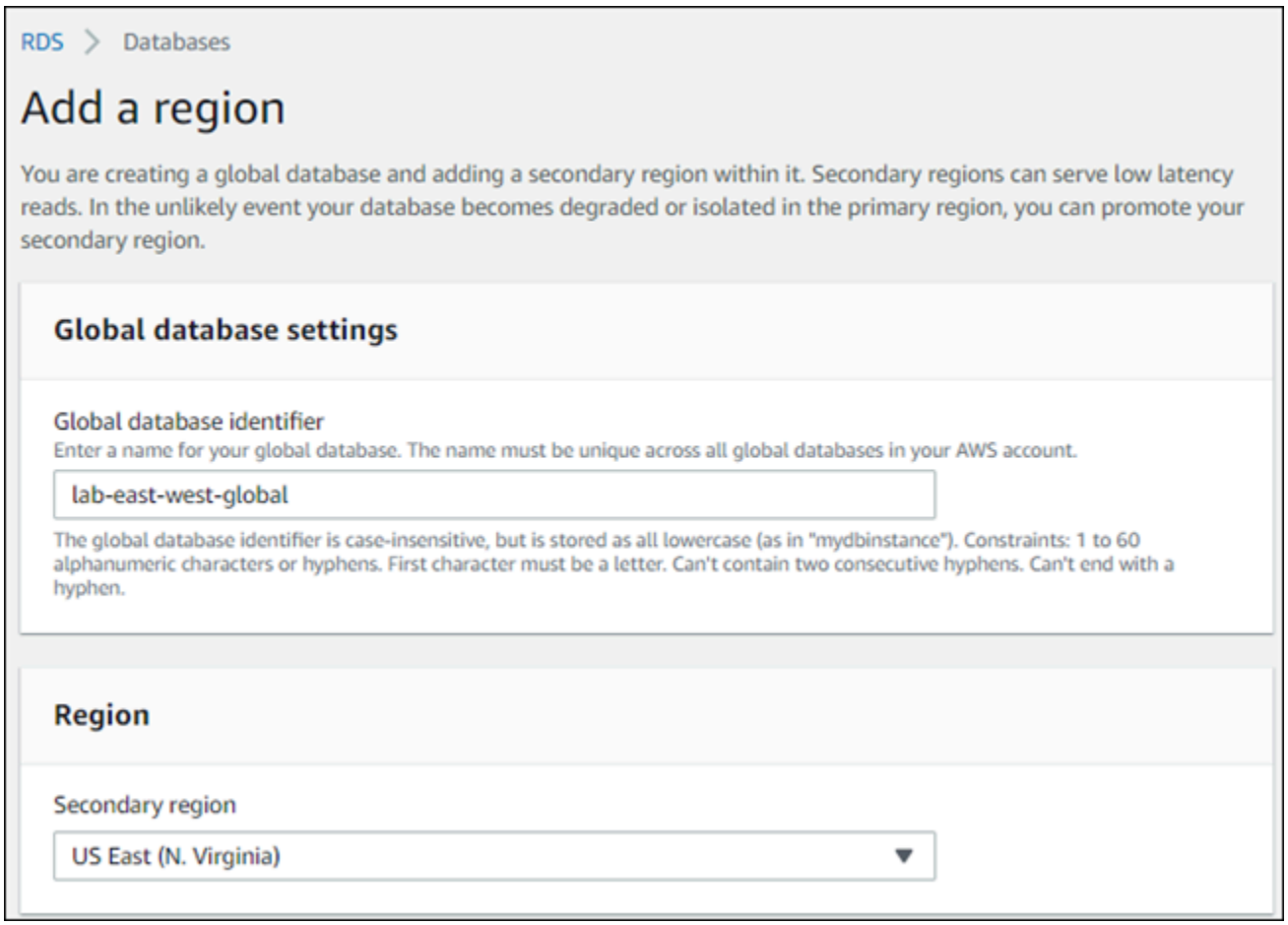
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi AWS Management Console, pilih Database.
3. Pilih basis data global Aurora yang membutuhkan klaster DB Aurora sekunder. Pastikan klaster DB Aurora primer berstatus `Available`.
4. Untuk Tindakan, pilih Tambah AWS Wilayah.




5. Pada halaman Tambahkan wilayah, pilih yang kedua Wilayah AWS.

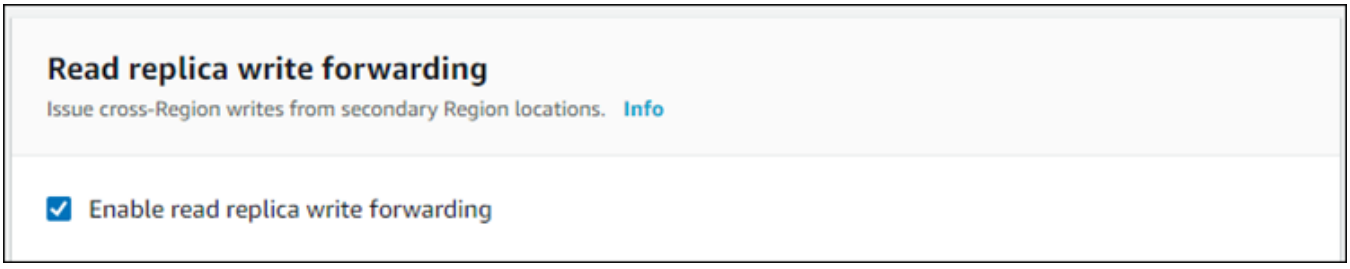
   Anda tidak dapat memilih Wilayah AWS yang sudah memiliki cluster Aurora DB sekunder untuk database global Aurora yang sama. Selain itu, Wilayah tersebut tidak boleh sama dengan Wilayah klaster DB Aurora primer.

> **Note**
>
> Basis data global Babelfish untuk Aurora PostgreSQL bekerja di wilayah sekunder hanya jika parameter yang mengontrol preferensi Babelfish diaktifkan di wilayah tersebut. Untuk informasi selengkapnya, lihat Pengaturan grup parameter klaster DB untuk Babelfish



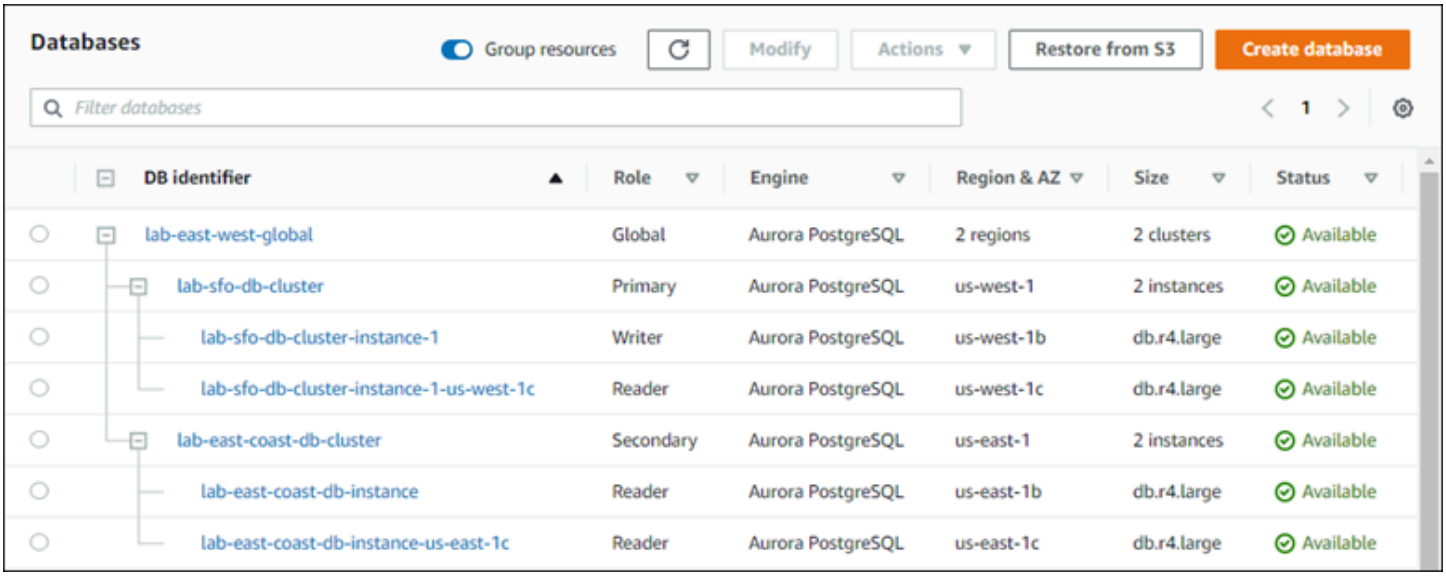

6. Lengkapi bidang yang tersisa untuk klaster Aurora sekunder di Wilayah AWS baru. Opsi konfigurasi ini sama seperti opsi konfigurasi untuk setiap instans klaster DB Aurora, kecuali opsi berikut yang ditujukan khusus untuk basis data global berbasis Aurora MySQL:

   - Aktifkan penerusan tulis replika baca - Pengaturan opsional ini memungkinkan klaster DB sekunder milik basis data global Aurora Anda meneruskan operasi tulis ke klaster primer. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.




7. Pilih Tambah AWS Wilayah.

Setelah Anda selesai menambahkan Wilayah ke database global Aurora Anda, Anda dapat melihatnya di daftar Database AWS Management Console seperti yang ditunjukkan pada tangkapan layar.




AWS CLI

Untuk menambahkan sekunder Wilayah AWS ke database global Aurora

Untuk menambahkan cluster sekunder ke database global Anda menggunakan CLI, Anda harus sudah memiliki objek kontainer cluster global. Jika Anda belum menjalankan `create-global-cluster` perintah, lihat prosedur CLI di. Membuat basis data global Amazon Aurora

1. Gunakan perintah CLI `create-db-cluster` dengan nama (`--global-cluster-identifier`) basis data global Aurora Anda. Untuk parameter lainnya, lakukan hal berikut:
2. Untuk`--region`, pilih yang berbeda Wilayah AWS dari Wilayah utama Aurora Anda.
3. Pilih nilai spesifik untuk parameter `--engine` dan `--engine-version`. Nilai-nilai tersebut sama dengan nilai untuk klaster DB Aurora primer dalam basis data global Aurora Anda.
4. Untuk klaster terenkripsi, tentukan primer Anda Wilayah AWS sebagai enkripsi `--source-region` untuk.

Contoh berikut ini membuat klaster DB Aurora baru dan melampirkannya ke basis data global Aurora sebagai klaster DB Aurora sekunder hanya-baca. Pada langkah terakhir, instans DB Aurora ditambahkan ke klaster DB Aurora baru.

Untuk Linux, macOS, atau Unix:

```
aws rds --region secondary_region \
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql | aurora-postgresql \
    --engine-version version

aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql | aurora-postgresql
```

Untuk Windows:

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql | aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql | aurora-postgresql
```

API RDS

Untuk menambahkan yang baru Wilayah AWS ke database global Aurora dengan RDS API, jalankan operasi Create. DBCluster Tentukan pengidentifikasi basis data global yang sudah ada dengan menggunakan parameter `GlobalClusterIdentifier`.

## Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder

Meskipun database global Aurora memerlukan setidaknya satu cluster Aurora DB sekunder yang berbeda Wilayah AWS dari yang utama, Anda dapat menggunakan konfigurasi tanpa kepala untuk cluster sekunder. Klaster DB Aurora sekunder tanpa kepala adalah klaster DB Aurora sekunder

yang tidak memiliki instans DB. Jenis konfigurasi ini dapat menurunkan biaya untuk basis data global Aurora. Dalam klaster DB Aurora, komputasi dan penyimpanan dipisahkan. Tanpa instans DB, Anda tidak dikenakan biaya untuk komputasi, hanya untuk penyimpanan. Jika diatur dengan benar, volume penyimpanan sekunder tanpa kepala tetap sinkron dengan klaster DB Aurora primer.

Anda menambahkan klaster sekunder seperti yang biasanya Anda lakukan saat membuat basis data global Aurora. Jika Anda membuat semua cluster di database global, ikuti prosedur diMembuat basis data global Amazon Aurora. Jika Anda sudah memiliki cluster DB untuk digunakan sebagai cluster utama, ikuti prosedur diMenambahkan Wilayah AWS ke database global Amazon Aurora.

Setelah cluster Aurora DB primer memulai replikasi ke sekunder, Anda menghapus instans DB hanya-baca Aurora dari cluster Aurora DB sekunder. Klaster sekunder ini sekarang dianggap “tanpa kepala” karena sudah tidak memiliki instans DB. Bahkan tanpa instans DB di cluster sekunder, Aurora menjaga volume penyimpanan tetap sinkron dengan cluster Aurora DB utama.

> **Warning**
>
> Dengan Aurora PostgreSQL, untuk membuat cluster tanpa kepala di sekunder Wilayah AWS, gunakan AWS CLI or RDS API untuk menambahkan sekunder. Wilayah AWS Lewati langkah ini untuk membuat instans DB pembaca untuk klaster sekunder. Saat ini, membuat klaster tanpa kepala tidak didukung di RDS Konsol. Untuk prosedur CLI dan API prosedur yang digunakan, lihatMenambahkan Wilayah AWS ke database global Amazon Aurora.
> Jika database global Anda menggunakan versi SQL mesin Aurora Postgre yang lebih rendah dari 13.4, 12.8, atau 11.13, membuat instans DB pembaca di Wilayah sekunder dan kemudian menghapusnya dapat menyebabkan masalah vakum Aurora SQL Postgre pada instans DB penulis Wilayah utama. Jika Anda mengalami masalah ini, mulai ulang instans DB penulis Wilayah primer setelah Anda menghapus instans DB pembaca sekunder Wilayah tersebut.

Menambahkan klaster DB Aurora sekunder tanpa kepala ke basis data global Aurora Anda

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi AWS Management Console, pilih Database.
3. Pilih basis data global Aurora yang membutuhkan klaster DB Aurora sekunder. Pastikan klaster DB Aurora primer berstatus `Available`.
4. Untuk Tindakan, pilih Tambah AWS Wilayah.

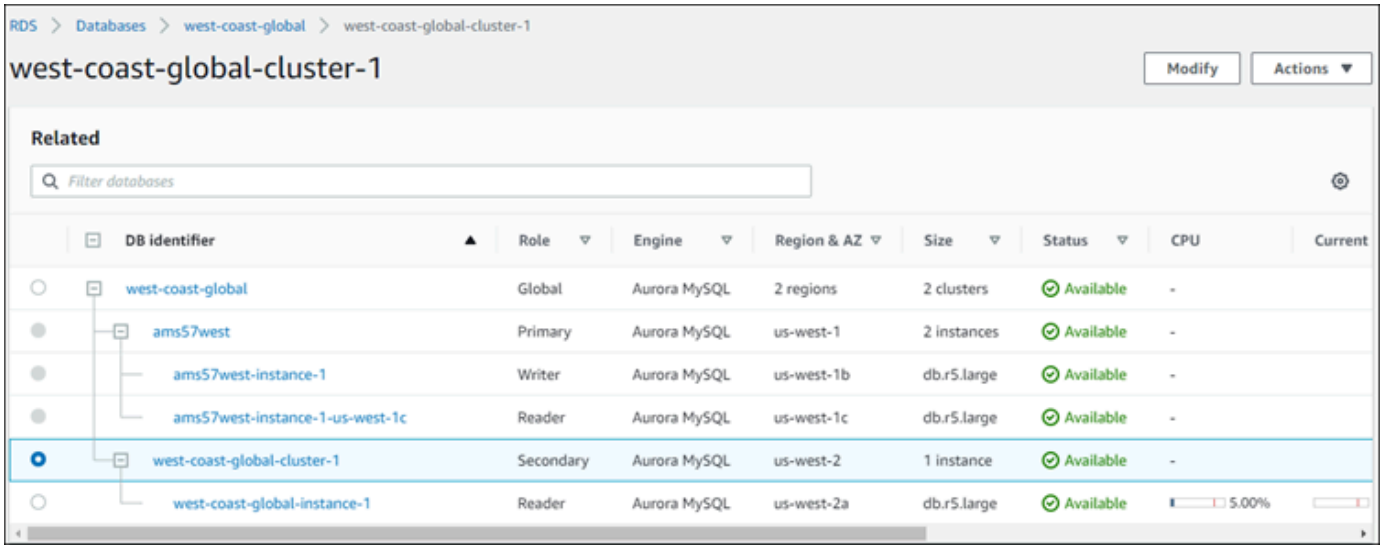
5. Pada halaman Tambahkan wilayah, pilih yang kedua Wilayah AWS.

   Anda tidak dapat memilih Wilayah AWS yang sudah memiliki cluster Aurora DB sekunder untuk database global Aurora yang sama. Selain itu, Wilayah tersebut tidak boleh sama dengan Wilayah klaster DB Aurora primer.

6. Lengkapi bidang yang tersisa untuk cluster Aurora sekunder di yang baru. Wilayah AWS Opsi konfigurasi ini sama dengan opsi konfigurasi untuk setiap instans klaster DB Aurora.

   Untuk basis data global Aurora Aurora My SQL berbasis Aurora, abaikan opsi Aktifkan penerusan tulis replika baca. Opsi ini tidak memiliki fungsi setelah Anda menghapus instans pembaca.

7. Pilih Tambah AWS Wilayah. Setelah Anda selesai menambahkan Wilayah ke database global Aurora Anda, Anda dapat melihatnya di daftar Database AWS Management Console seperti yang ditunjukkan pada tangkapan layar.

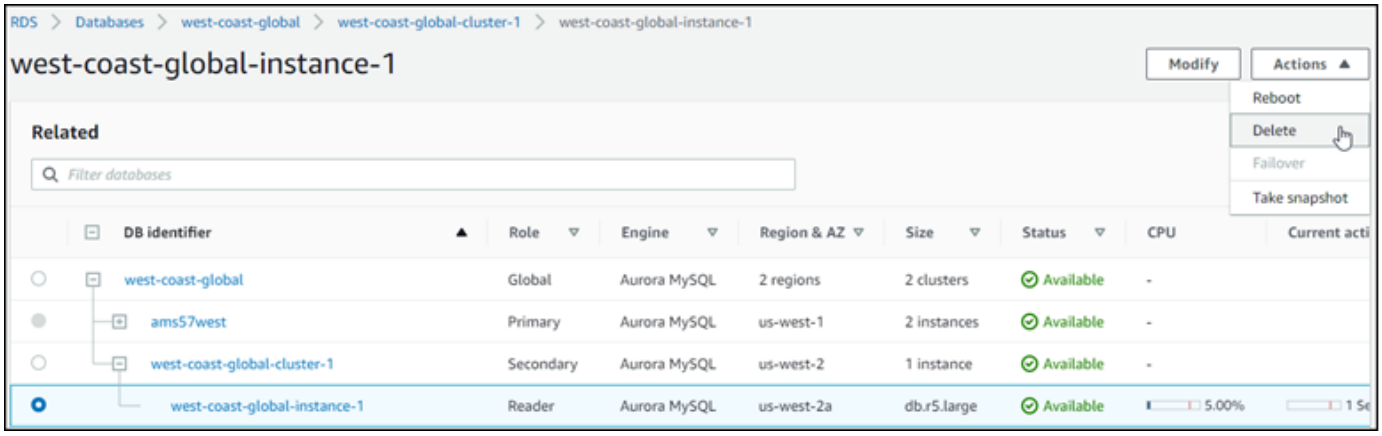



8. Periksa status cluster Aurora DB sekunder dan instance pembacanya sebelum melanjutkan, dengan menggunakan AWS Management Console atau. AWS CLI Sebagai contoh:

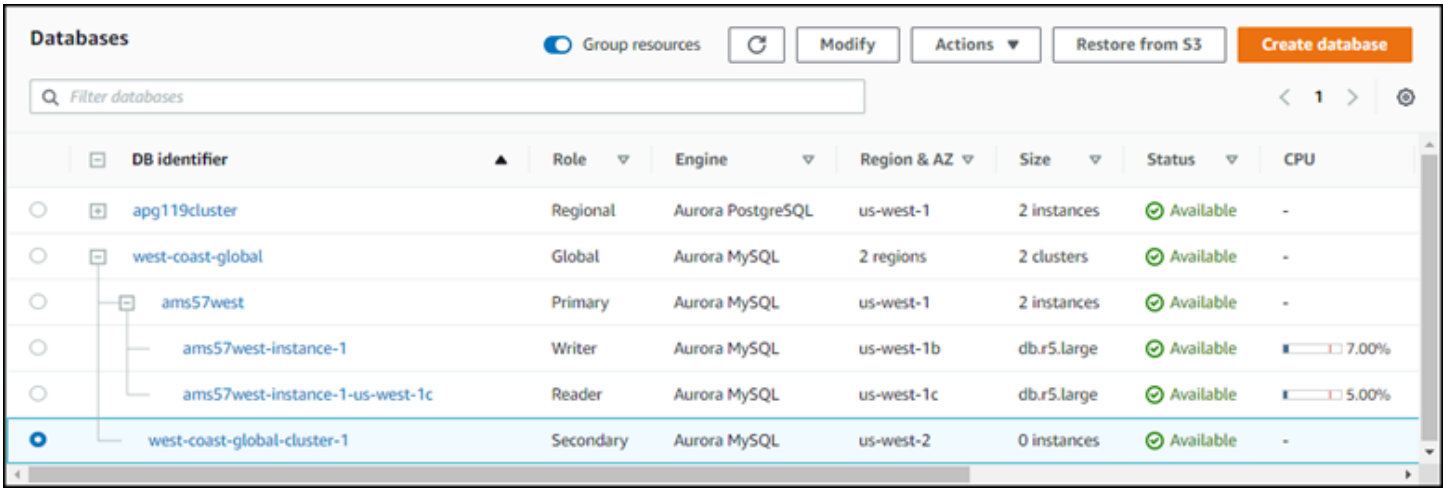
```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
 '*[].[Status]' --output text
```

   Mungkin diperlukan waktu beberapa menit hingga status klaster DB Aurora sekunder yang baru ditambahkan berubah dari `creating` menjadi `available`. Ketika klaster DB Aurora tersedia, Anda dapat menghapus instans pembaca.

9. Pilih instans pembaca di klaster DB Aurora sekunder, kemudian pilih Hapus.




Setelah menghapus instans pembaca, klaster sekunder tetap menjadi bagian basis data global Aurora. Klaster tersebut tidak memiliki instans yang terkait dengannya, seperti yang ditunjukkan berikut ini.




Anda dapat menggunakan klaster DB Aurora sekunder tanpa kepala ini untuk secara manual memulihkan basis data global Amazon Aurora Anda dari pemadaman yang tidak direncanakan di Wilayah AWS primer jika terjadi pemadaman seperti itu.

## Membuat database global Amazon Aurora dari snapshot Aurora atau Amazon RDS

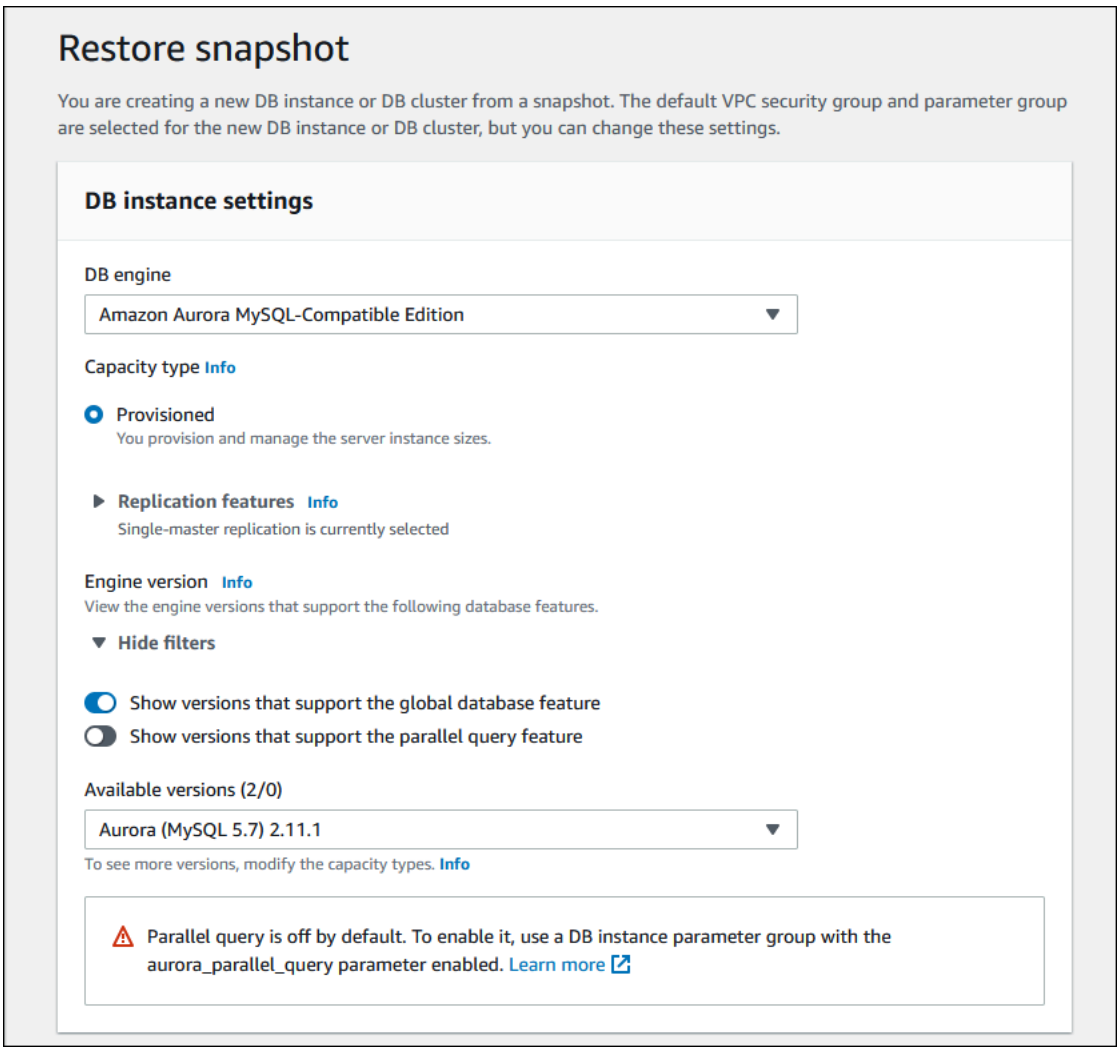
Anda dapat memulihkan snapshot cluster Aurora DB atau dari instans RDS Amazon DB untuk digunakan sebagai titik awal untuk database global Aurora Anda. Anda memulihkan snapshot dan membuat klaster DB baru yang disediakan Aurora pada waktu yang sama. Anda kemudian menambahkan yang lain Wilayah AWS ke cluster DB yang dipulihkan, sehingga mengubahnya

menjadi database global Aurora. Klaster DB Aurora apa pun yang Anda buat menggunakan snapshot dengan cara ini akan menjadi klaster primer basis data global Aurora Anda.

Snapshot yang Anda gunakan dapat berasal dari klaster DB Aurora `provisioned` atau `serverless`.

Selama proses pemulihan, pilih jenis mesin DB yang sama dengan snapshot. Misalnya, anggaplah Anda ingin mengembalikan snapshot yang dibuat dari sebuah Aurora Serverless Cluster DB menjalankan Aurora Postgre. SQL Dalam hal ini, Anda membuat cluster Aurora Postgre SQL DB menggunakan mesin dan versi Aurora DB yang sama.

Cluster DB yang dipulihkan mengasumsikan peran cluster utama untuk database global Aurora saat Anda menambahkan ke dalamnya Wilayah AWS . Semua data yang terkandung dalam klaster primer ini direplikasi ke setiap klaster sekunder yang Anda tambahkan ke basis data global Aurora Anda.



# Mengelola basis data global Amazon Aurora

Anda melakukan sebagian besar operasi manajemen pada klaster individu yang membentuk basis data global Aurora. Saat Anda memilih Sumber daya terkait grup di halaman Basis data dalam konsol, Anda melihat klaster primer dan klaster sekunder yang dikelompokkan di bawah basis data global yang dikaitkan. Untuk menemukan di Wilayah AWS mana cluster DB database global berjalan, mesin dan versi Aurora DB-nya, dan pengenalnya, gunakan tab Konfigurasinya.

Proses failover basis data lintas wilayah tersedia untuk basis data global Aurora saja, dan tidak tersedia untuk klaster DB Aurora tunggal. Untuk mempelajari selengkapnya, lihat Menggunakan switchover atau failover di Amazon Aurora Global Database.

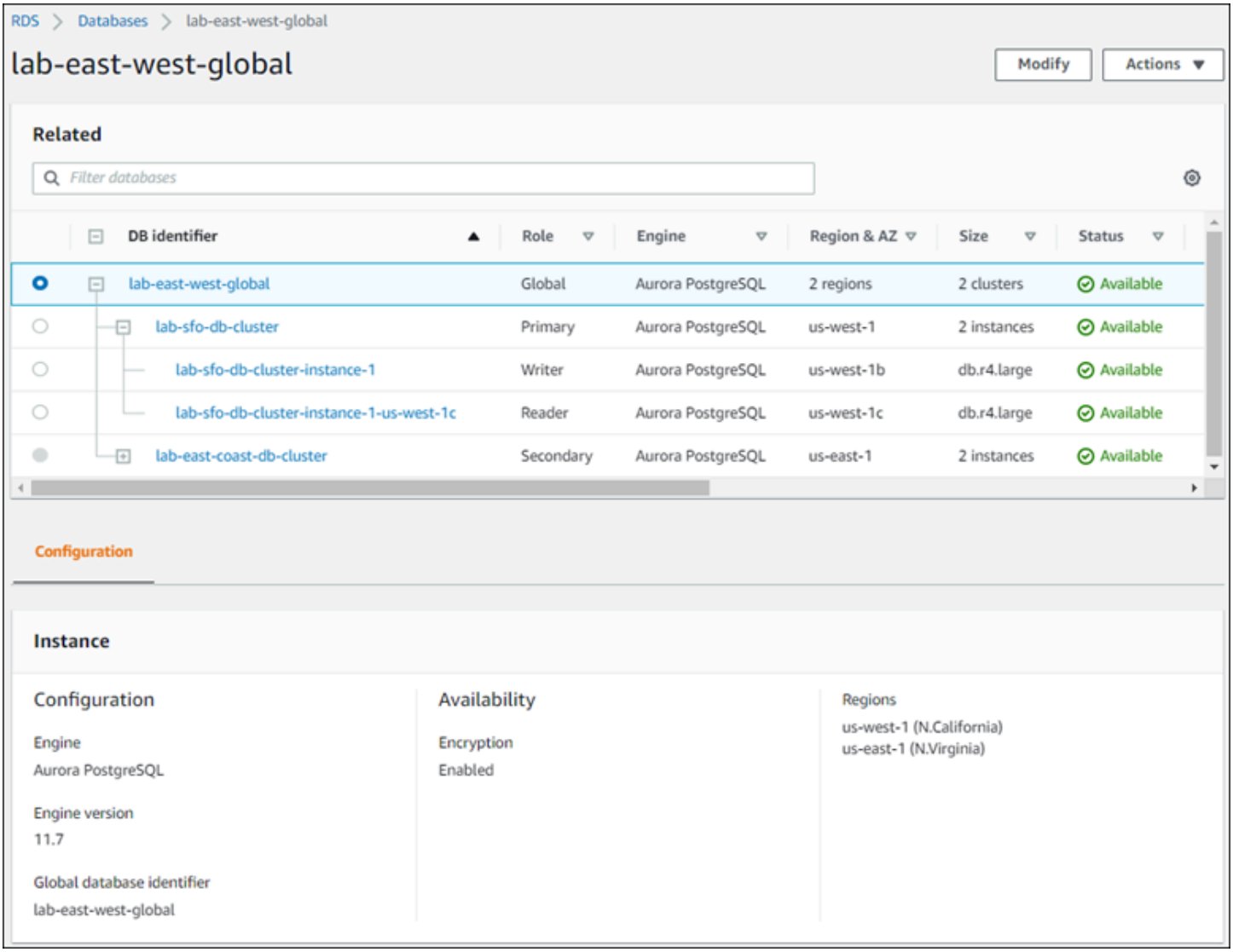
Untuk memulihkan basis data global Aurora dari pemadaman yang tidak direncanakan di Wilayah primernya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

Topik

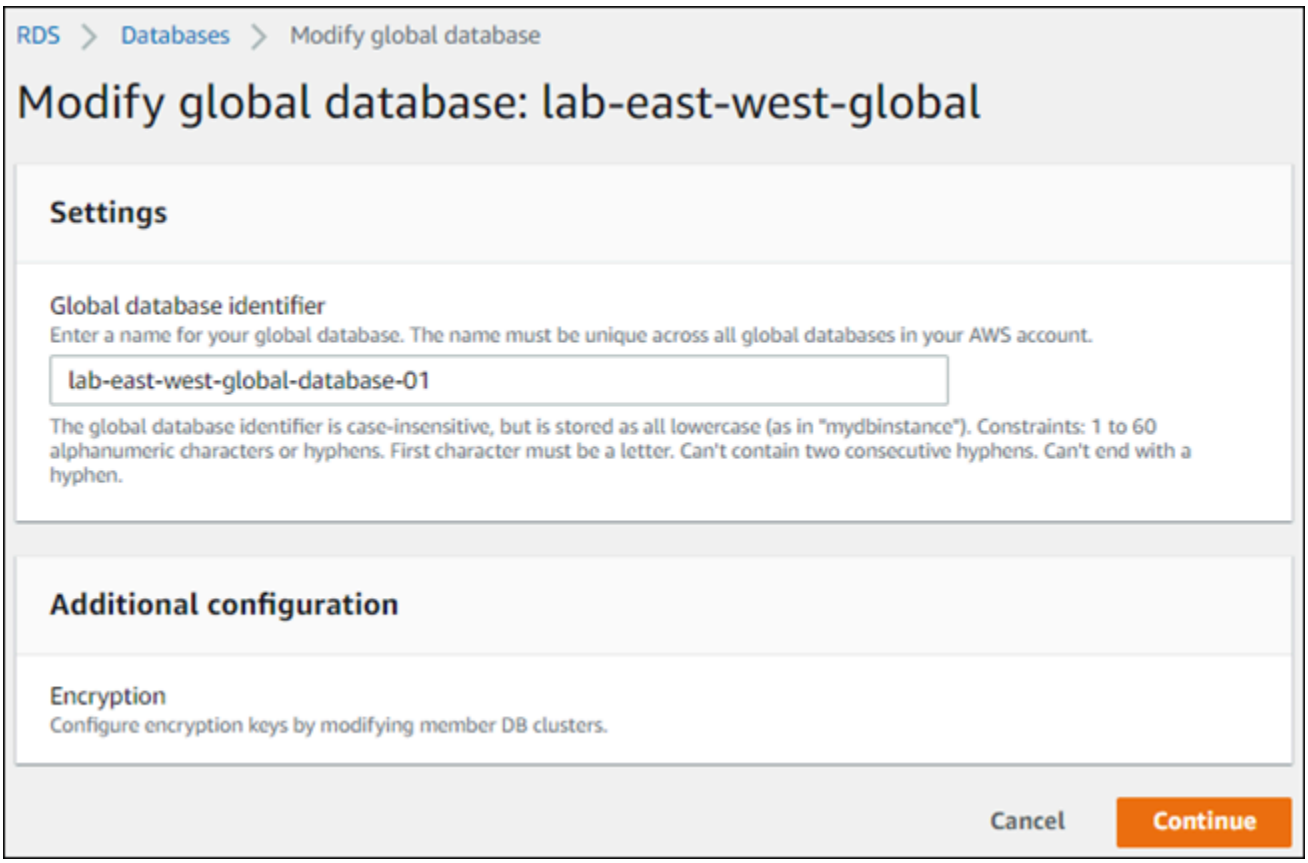

- Memodifikasi basis data global Amazon Aurora
- Memodifikasi parameter basis data global Aurora
- Menghapus klaster dari basis data global Amazon Aurora
- Menghapus basis data global Amazon Aurora
- Menandai sumber daya Basis Data Global Amazon Aurora


## Memodifikasi basis data global Amazon Aurora

Halaman Database dalam AWS Management Console daftar semua database global Aurora Anda, menunjukkan cluster primer dan cluster sekunder untuk masing-masing database. Basis data global Aurora mempunyai pengaturan konfigurasinya sendiri. Secara khusus, ini telah Wilayah AWS dikaitkan dengan cluster primer dan sekundernya, seperti yang ditunjukkan pada tangkapan layar berikut.




Ketika Anda membuat perubahan pada basis data global Aurora, Anda memiliki kesempatan untuk membatalkan perubahan, seperti yang ditunjukkan pada tangkapan layar berikut.




Ketika Anda memilih Lanjutkan, Anda mengonfirmasi perubahannya.

## Memodifikasi parameter basis data global Aurora

Anda dapat mengonfigurasi grup parameter klaster DB Aurora secara independen untuk setiap klaster Aurora yang ada di dalam basis data global Aurora. Cara kerja sebagian besar parameter sama dengan jenis klaster Aurora lainnya. Kami menyarankan Anda menjaga pengaturan tetap konsisten di antara semua klaster di dalam basis data global. Tindakan tersebut membantu mencegah perubahan perilaku yang tidak terduga jika Anda menaikkan klaster sekunder menjadi klaster primer.

Misalnya, gunakan pengaturan zona waktu dan rangkaian karakter yang sama untuk mencegah perilaku yang tidak konsisten jika klaster yang berbeda mengambil alih peran sebagai klaster primer.

Pengaturan konfigurasi `aurora_enable_repl_bin_log_filtering` dan `aurora_enable_replica_log_compression` tidak berpengaruh.

# Menghapus klaster dari basis data global Amazon Aurora

Anda dapat menghapus klaster DB Aurora dari basis data global Aurora Anda untuk beberapa alasan yang berbeda. Misalnya, Anda mungkin ingin menghapus klaster Aurora dari global basis data jika klaster primer tersebut terdegradasi atau terisolasi. Klaster tersebut kemudian menjadi klaster DB Aurora yang disediakan yang berdiri sendiri dan dapat digunakan untuk membuat basis data global Aurora baru. Untuk mempelajari selengkapnya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

Anda juga mungkin ingin menghapus klaster DB Aurora karena Anda ingin menghapus basis data global Aurora yang sudah tidak Anda perlukan. Anda tidak dapat menghapus basis data global Aurora sampai Anda menghapus (melepaskan) semua klaster DB Aurora terkait, dan menyisakan klaster primer. Untuk informasi selengkapnya, lihat Menghapus basis data global Amazon Aurora.

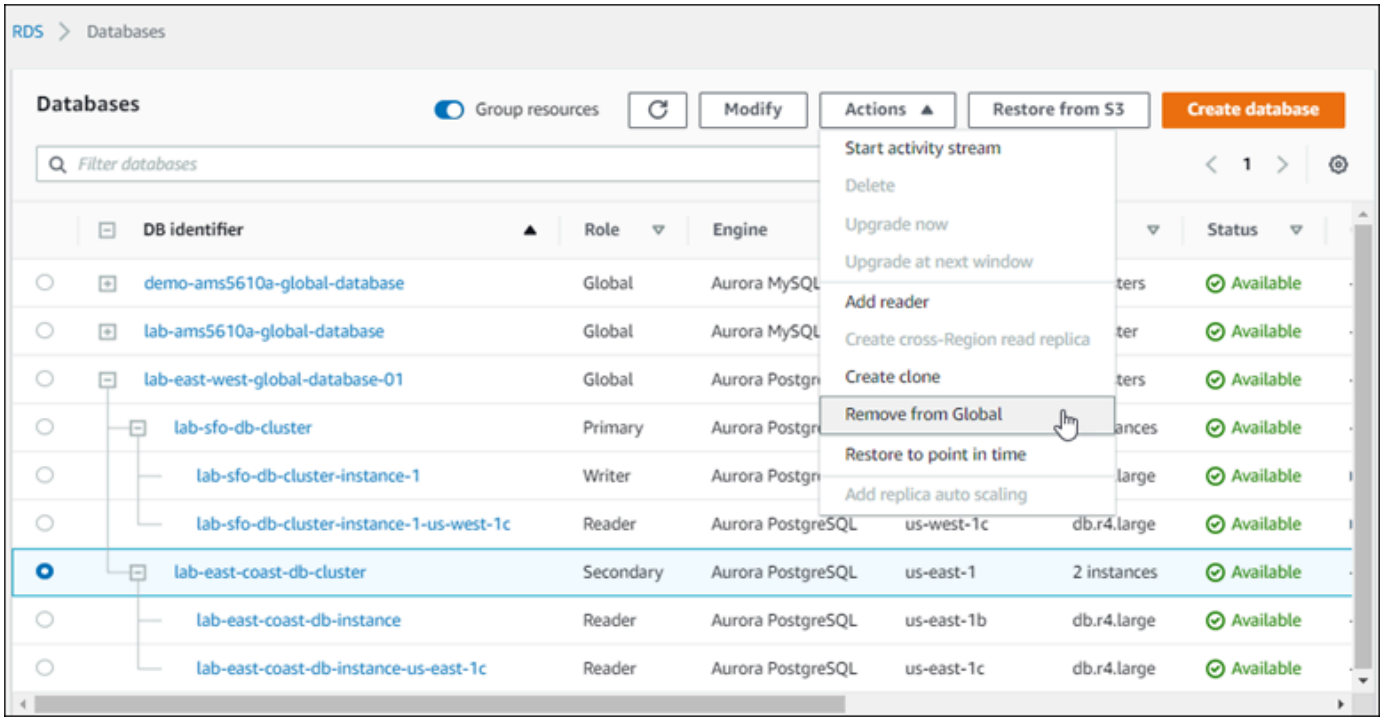
Ketika klaster DB Aurora dilepaskan dari basis data global Aurora, klaster tersebut tidak lagi disinkronkan dengan klaster primer. Klaster tersebut menjadi klaster DB Aurora yang berdiri sendiri dengan kapabilitas baca/tulis penuh.

Anda dapat menghapus cluster Aurora DB dari database global Aurora Anda menggunakan, file AWS Management Console, atau file. AWS CLI RDS API

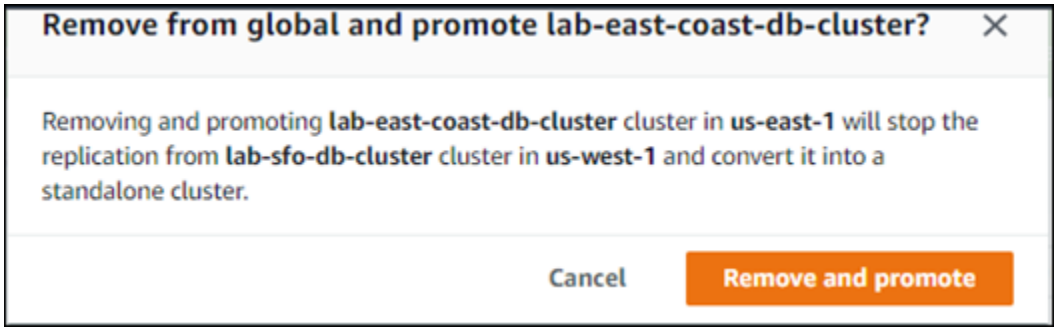
Konsol

Menghapus klaster Aurora dari basis data global Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih klaster pada halaman Basis data.
3. Untuk Tindakan, pilih Hapus dari Global.

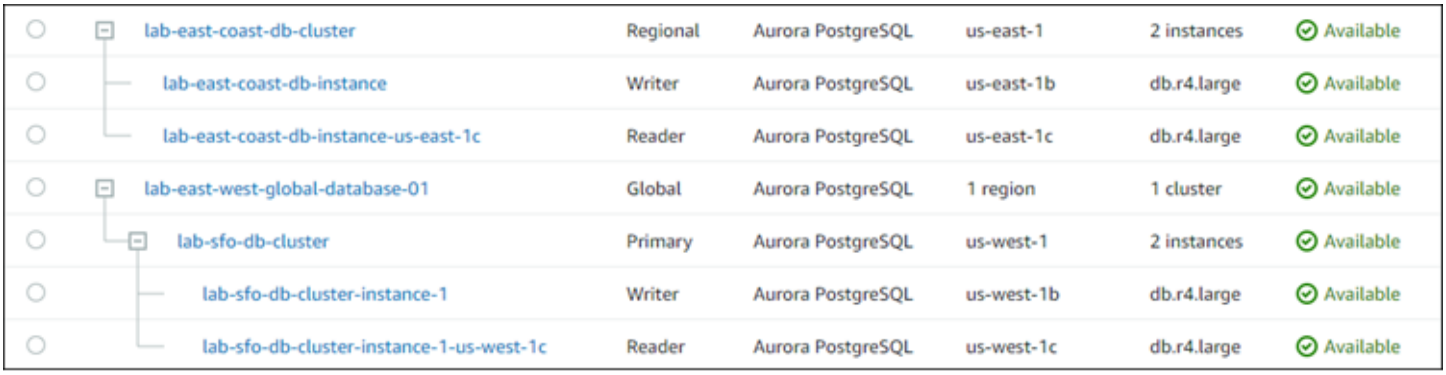



Anda melihat sebuah prompt untuk mengonfirmasi bahwa Anda ingin melepaskan klaster sekunder dari basis data global Aurora.




4. Pilih Hapus dan promosikan untuk menghapus klaster dari basis data global.

Klaster DB Aurora sudah tidak melayani sebagai klaster sekunder di dalam basis data global Aurora, dan sudah tidak disinkronkan dengan klaster DB primer. Ini adalah klaster DB Aurora yang berdiri sendiri dengan kapabilitas baca/tulis penuh.




Setelah Anda menghapus semua klaster sekunder, Anda dapat menghapus klaster primer dengan cara yang sama. Anda tidak dapat melepaskan (menghapus) klaster DB Aurora primer dari basis data global Aurora sampai Anda menghapus semua klaster sekunder.

Database global Aurora mungkin tetap ada dalam daftar Database, dengan nol Wilayah dan. AZs Anda dapat menghapusnya jika Anda sudah tidak ingin menggunakan basis data global Aurora ini. Untuk informasi selengkapnya, lihat Menghapus basis data global Amazon Aurora.

AWS CLI

Untuk menghapus cluster Aurora dari database global Aurora, jalankan remove-from-global-clusterCLIperintah dengan parameter berikut:

- `--global-cluster-identifier` — Nama (pengidentifikasi) basis data global Aurora Anda.
- `--db-cluster-identifier` — Nama setiap klaster DB Aurora yang akan dihapus dari basis data global Aurora. Hapus semua klaster DB Aurora sekunder sebelum menghapus klaster primer.

Contoh berikut ini terlebih dahulu menghapus klaster sekunder dan kemudian menghapus klaster primer dari basis data global Aurora.

Untuk Linux, macOS, atau Unix:

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

Ulangi `remove-from-global-cluster --db-cluster-identifier` *`secondary_cluster_ARN`* perintah untuk setiap sekunder Wilayah AWS di database global Aurora Anda.

Untuk Windows:

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

Ulangi `remove-from-global-cluster --db-cluster-identifier` *`secondary_cluster_ARN`* perintah untuk setiap sekunder Wilayah AWS di database global Aurora Anda.

RDS API

Untuk menghapus cluster Aurora dari database global Aurora dengan RDSAPI, jalankan tindakan. RemoveFromGlobalCluster

## Menghapus basis data global Amazon Aurora

Karena basis data global Aurora biasanya menyimpan data penting bisnis, Anda tidak dapat menghapus basis data global dan klaster terkait dalam satu langkah. Untuk menghapus basis data global Aurora, lakukan hal berikut:

- Hapus klaster sekunder dari basis data global Aurora. Setiap klaster menjadi sebuah klaster DB Aurora yang berdiri sendiri. Untuk mempelajari caranya, lihat Menghapus klaster dari basis data global Amazon Aurora.
- Dari setiap klaster DB Aurora yang berdiri sendiri, hapus semua Replika Aurora.
- Hapus klaster DB primer dari basis data global Aurora. Klaster ini menjadi klaster DB Aurora yang berdiri sendiri.
- Dari klaster DB Aurora, pertama-tama hapus semua Replika Aurora, kemudian hapus instans DB penulis.

Menghapus instans penulis dari klaster DB Aurora baru yang berdiri sendiri juga biasanya menghapus klaster DB Aurora dan basis data global Aurora.

Untuk informasi umum selengkapnya, lihat Menghapus instans dari klaster DB Aurora.

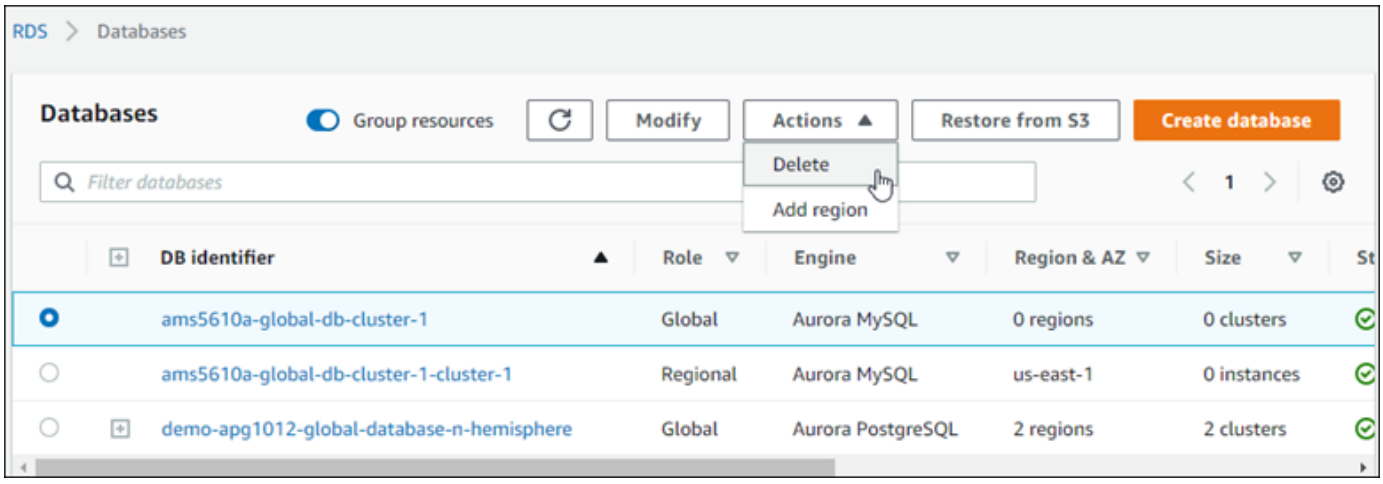
Untuk menghapus database global Aurora, Anda dapat menggunakan AWS Management Console, file AWS CLI, atau file. RDS API

Konsol

Menghapus basis data global Aurora

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Basis data dan temukan basis data global Aurora yang ingin Anda hapus di daftar.
3. Konfirmasikan bahwa semua klaster lainnya telah dihapus dari basis data global Aurora. Database global Aurora harus menunjukkan 0 Wilayah dan AZs dan ukuran 0 cluster.

   Jika berisi klaster DB Aurora, basis data global Aurora tidak dapat dihapus. Jika perlu, lepaskan klaster DB Aurora primer dan sekunder dari basis data global Aurora. Untuk informasi selengkapnya, lihat Menghapus klaster dari basis data global Amazon Aurora.
4. Pilih basis data global Aurora Anda di dalam daftar, lalu pilih Hapus dari menu Tindakan.




AWS CLI

Untuk menghapus database global Aurora, jalankan delete-global-clusterCLIperintah dengan nama Wilayah AWS dan pengidentifikasi database global Aurora, seperti yang ditunjukkan pada contoh berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

Untuk Windows:

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

RDS API

Untuk menghapus cluster yang merupakan bagian dari database global Aurora, jalankan operasi. DeleteGlobalClusterAPI

## Menandai sumber daya Basis Data Global Amazon Aurora

Dengan fitur Aurora Global Database, Anda dapat menerapkan RDS tag ke sumber daya pada tingkat yang berbeda dalam database global. Jika Anda tidak terbiasa dengan bagaimana tag digunakan AWS atau sumber daya Aurora, lihat Menandai sumber daya Amazon Aurora dan Amazon RDS sebelum menerapkan tag dalam database global Anda.

> **Note**
>
> Karena AWS proses menandai data sebagai bagian dari mekanisme pelaporan biaya, jangan sertakan data sensitif atau informasi yang dapat diidentifikasi secara pribadi (PII) dalam nama atau nilai tag.

Anda dapat menerapkan tag ke jenis sumber daya berikut dalam database global:

- Objek kontainer untuk seluruh database global Anda. Sumber daya ini dikenal sebagai cluster global.

  Setelah membuat klaster global dengan melakukan operasi Tambah AWS Wilayah di konsol, Anda dapat menambahkan tag menggunakan halaman detail untuk klaster global. Pada tab Tag dalam halaman detail klaster global, Anda dapat menambahkan, menghapus, atau memodifikasi tag dan nilai terkait dengan memilih Kelola tag.

Dengan AWS CLI dan RDSAPI, Anda dapat menambahkan tag ke cluster global pada saat yang sama Anda membuatnya. Anda juga dapat menambahkan, menghapus, atau memodifikasi tag untuk klaster global yang ada.

- Cluster utama. Anda menggunakan prosedur yang sama untuk bekerja dengan tag di sini seperti untuk cluster Aurora mandiri. Anda dapat mengatur tag sebelum mengubah cluster Aurora asli menjadi database global. Anda juga dapat menambahkan, menghapus, atau memodifikasi tag dan nilai terkait dengan memilih Kelola tag pada tab Tag dalam halaman detail cluster DB.
- Cluster sekunder apa pun. Anda menggunakan prosedur yang sama untuk bekerja dengan tag di sini seperti untuk cluster Aurora mandiri. Anda dapat mengatur tag pada saat yang sama saat Anda membuat klaster Aurora sekunder dengan menggunakan tindakan Tambah AWS Wilayah di konsol. Anda juga dapat menambahkan, menghapus, atau memodifikasi tag dan nilai terkait dengan memilih Kelola tag pada tab Tag dalam halaman detail cluster DB.
- Instance DB individu dalam cluster primer atau sekunder. Anda menggunakan prosedur yang sama untuk bekerja dengan tag di sini seperti untuk instans Aurora atau RDS DB. Anda dapat mengatur tag pada saat yang sama saat Anda menambahkan instans DB baru ke cluster Aurora dengan menggunakan tindakan Add reader di konsol. Anda juga dapat menambahkan, menghapus, atau memodifikasi tag dan nilai terkait dengan memilih Kelola tag pada tab Tag dalam halaman detail instans DB.

Berikut adalah beberapa contoh jenis tag yang mungkin Anda tetapkan dalam database global:

- Anda dapat menambahkan tag ke klaster global untuk merekam informasi keseluruhan tentang aplikasi Anda, seperti pengidentifikasi anonim yang mewakili pemilik dan kontak dalam organisasi Anda. Anda dapat menggunakan tag untuk mewakili properti aplikasi yang menggunakan database global.
- Anda dapat menambahkan tag ke klaster primer dan klaster sekunder untuk melacak biaya aplikasi Anda di tingkat AWS Wilayah. Untuk detail tentang prosedur itu, lihatCara kerja AWS penagihan dengan tag di Amazon RDS.
- Anda dapat menambahkan tag ke instans DB tertentu dengan cluster Aurora untuk menunjukkan tujuan khusus mereka. Misalnya, dalam klaster utama, Anda mungkin memiliki instance pembaca dengan prioritas failover rendah yang digunakan secara eksklusif untuk pembuatan laporan. Sebuah tag dapat membedakan instans DB tujuan khusus ini dari instance lain yang didedikasikan untuk ketersediaan tinggi dalam klaster utama.

- Anda dapat menggunakan tag di semua tingkatan sumber daya database global Anda untuk mengontrol akses melalui IAM kebijakan. Untuk informasi selengkapnya, lihat Mengontrol akses ke AWS sumber daya di Panduan Pengguna AWS Identity and Access Management.

  > **Tip**
  >
  > Di AWS Management Console, Anda menambahkan tag ke wadah cluster global sebagai langkah terpisah setelah Anda membuatnya. Jika Anda ingin menghindari interval waktu ketika klaster global ada tanpa tag kontrol akses, Anda dapat menerapkan tag selama `CreateGlobalCluster` operasi dengan membuat sumber daya itu melalui AWS CLI RDSAPI,, atau AWS CloudFormation templat.

- Anda dapat menggunakan tag di tingkat klaster, atau untuk klaster global, untuk merekam informasi tentang jaminan kualitas dan pengujian aplikasi Anda. Misalnya, Anda dapat menentukan tag pada kluster DB untuk merekam terakhir kali Anda melakukan peralihan ke cluster tersebut. Anda dapat menentukan tag pada klaster global untuk mencatat waktu latihan pemulihan bencana terakhir untuk seluruh aplikasi.

## AWS CLI contoh penandaan untuk database global

AWS CLI Contoh berikut menunjukkan bagaimana Anda dapat menentukan dan memeriksa tag untuk semua jenis sumber daya Aurora di database global Anda.

Anda dapat menentukan tag untuk wadah cluster global dengan `create-global-cluster` perintah. Contoh berikut membuat cluster global dan menetapkan dua tag untuk itu. Tag memiliki kunci `tag1` dan`tag2`.

```
$  aws rds create-global-cluster --global-cluster-identifier my_global_cluster_id \
  --engine aurora-mysql --tags Key=tag1,Value=val1 Key=tag2,Value=val2
```

Anda dapat membuat daftar tag pada wadah cluster global dengan `describe-global-clusters` perintah. Saat bekerja dengan tag, Anda biasanya menjalankan perintah ini terlebih dahulu untuk mengambil Amazon Resource Name (ARN) dari cluster global. Anda menggunakan ARN sebagai parameter dalam perintah berikutnya untuk bekerja dengan tag. Perintah berikut menampilkan informasi tag dalam `TagList` atribut. Ini juga menunjukkanARN, yang digunakan sebagai parameter dalam contoh selanjutnya.

```
$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
```

```
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "GlobalClusterArn": "my_global_cluster_arn",
            ...
            "TagList": [
                {
                    "Value": "val1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                }
            ]
        }
    ]
}
```

Anda dapat menambahkan tag baru dengan `add-tags-to-resource` perintah. Dengan perintah ini, Anda menentukan Amazon Resource Name (ARN) dari cluster global, bukan pengenalnya. Menambahkan tag dengan nama yang sama dengan tag yang ada menimpa nilai tag tersebut. Jika Anda menyertakan spasi atau karakter khusus dalam nilai tag, kutip nilai yang sesuai untuk sistem operasi atau shell perintah Anda. Contoh berikut memodifikasi tag cluster global dari contoh sebelumnya. Awalnya, cluster memiliki tag dengan kunci `tag1` dan`tag2`. Setelah perintah selesai, cluster global memiliki tag baru dengan kunci`tag3`, dan tag dengan kunci `tag1` memiliki nilai yang berbeda.

```
$  aws rds add-tags-to-resource --resource-name my_global_cluster_arn \
  --tags Key=tag1,Value="new value for tag1" Key=tag3,Value="entirely new tag"

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
```

```
                    "Value": "new value for tag1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                },
                {
                    "Value": "entirely new tag",
                    "Key": "tag3"
                }
            ]
        }
    ]
}
```

Anda dapat menghapus tag dari cluster global dengan `remove-tags-from-resource` perintah. Dengan perintah ini, Anda hanya menentukan satu set kunci tag, tanpa nilai tag apa pun. Contoh berikut memodifikasi tag cluster global dari contoh sebelumnya. Awalnya, cluster memiliki tag dengan kunci`tag1`,`tag2`, dan`tag3`. Setelah perintah selesai, hanya tag dengan kunci yang `tag1` tersisa.

```
$  aws rds remove-tags-from-resource --resource-name my_global_cluster_arn --tag-keys
 tag2 tag3

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                }
            ]
        }
    ]
}
```

# Menghubungkan ke Amazon Aurora Global Database

Setiap Database Global Aurora dilengkapi dengan titik akhir penulis yang secara otomatis diperbarui oleh Aurora untuk merutekan permintaan ke instance penulis saat ini dari cluster DB primer. Dengan titik akhir penulis, Anda tidak perlu memodifikasi string koneksi Anda setelah Anda mengubah lokasi Wilayah utama menggunakan kemampuan switchover dan failover Aurora Global Database yang dikelola. Untuk mempelajari lebih lanjut tentang menggunakan titik akhir penulis bersama dengan peralihan dan failover Aurora Global Database, lihat. Menggunakan switchover atau failover di Amazon Aurora Global Database Untuk informasi tentang menghubungkan ke Database Global Aurora dengan Proksi RDS, lihat Menggunakan RDS Proxy dengan database global Aurora.

Topik


- Memilih titik akhir yang memenuhi kebutuhan aplikasi Anda
- Melihat titik akhir database global Amazon Aurora
- Pertimbangan dengan menggunakan titik akhir penulis Global


## Memilih titik akhir yang memenuhi kebutuhan aplikasi Anda

Menghubungkan ke Database Global Aurora tergantung pada kebutuhan Anda untuk membaca atau menulis dari database dan AWS Wilayah yang ingin Anda rutekan permintaan Anda. Berikut adalah beberapa kasus penggunaan yang khas:

- Permintaan perutean ke instance penulis: Sambungkan ke titik akhir penulis Aurora Global Database jika Anda perlu menjalankan pernyataan bahasa manipulasi data (DHTML) dan bahasa definisi data (DDL), atau jika Anda memerlukan konsistensi yang kuat antara membaca dan menulis. Titik akhir itu merutekan permintaan ke instance penulis di cluster utama database global Anda. Titik akhir ini diperbarui secara otomatis untuk merutekan permintaan ke instance penulis, menghilangkan kebutuhan untuk memperbarui aplikasi Anda setiap kali Anda mengubah lokasi penulis di cluster global Anda. Anda juga dapat menggunakan endpoint global untuk mengirim read/write permintaan lintas wilayah ke penulis Anda.

> **Note**
>
> Jika Anda menyiapkan database global sebelum titik akhir penulis Aurora Global Database tersedia, aplikasi Anda mungkin terhubung ke titik akhir klaster klaster utama. Dalam hal ini, sebaiknya alihkan pengaturan koneksi Anda untuk menggunakan titik akhir

penulis global sebagai gantinya. Melakukannya menghindari kebutuhan untuk mengubah pengaturan koneksi Anda setelah setiap peralihan atau failover Aurora Global Database. Bagian pertama dari nama titik akhir penulis adalah nama Database Global Aurora Anda. Jadi, jika Anda mengganti nama Aurora Global Database Anda, nama titik akhir penulis berubah, dan kode apa pun yang menggunakannya harus diperbarui dengan nama baru.

- Penskalaan dibaca lebih dekat ke wilayah aplikasi Anda: Untuk menskalakan permintaan hanya-baca di AWS Wilayah yang sama atau terdekat dengan aplikasi Anda, sambungkan ke titik akhir pembaca klaster Aurora primer atau sekunder.
- Penskalaan pembacaan dengan penulisan lintas wilayah sesekali: Untuk pernyataan DHTML sesekali seperti untuk pemeliharaan dan pembersihan data, sambungkan ke titik akhir pembaca klaster sekunder yang mengaktifkan penerusan tulis. Dengan penerusan tulis, Aurora secara otomatis meneruskan pernyataan tulis kepada penulis di Wilayah utama Basis Data Global Aurora Anda. Penerusan tulis memberikan manfaat berikut:
  - Anda tidak perlu melakukan pekerjaan berat untuk membangun konektivitas antara cluster sekunder dan primer untuk mengirim penulisan lintas wilayah.
  - Anda tidak perlu membagi permintaan baca dan tulis dalam aplikasi.
  - Anda tidak perlu mengembangkan logika kompleks untuk mengelola konsistensi read-after-write permintaan.

  Namun, dengan penerusan tulis, Anda perlu memperbarui kode atau konfigurasi aplikasi Anda untuk terhubung ke titik akhir pembaca Region primer yang baru dipromosikan setelah melakukan failover atau switchover lintas wilayah. Kami menyarankan Anda memantau latensi operasi yang dilakukan melalui penerusan tulis, untuk memeriksa overhead pemrosesan permintaan tulis. Terakhir, penerusan tulis tidak mendukung operasi MySQL atau PostgreSQL tertentu, seperti membuat perubahan atau pernyataan bahasa definisi data (DDL). `SELECT FOR UPDATE`

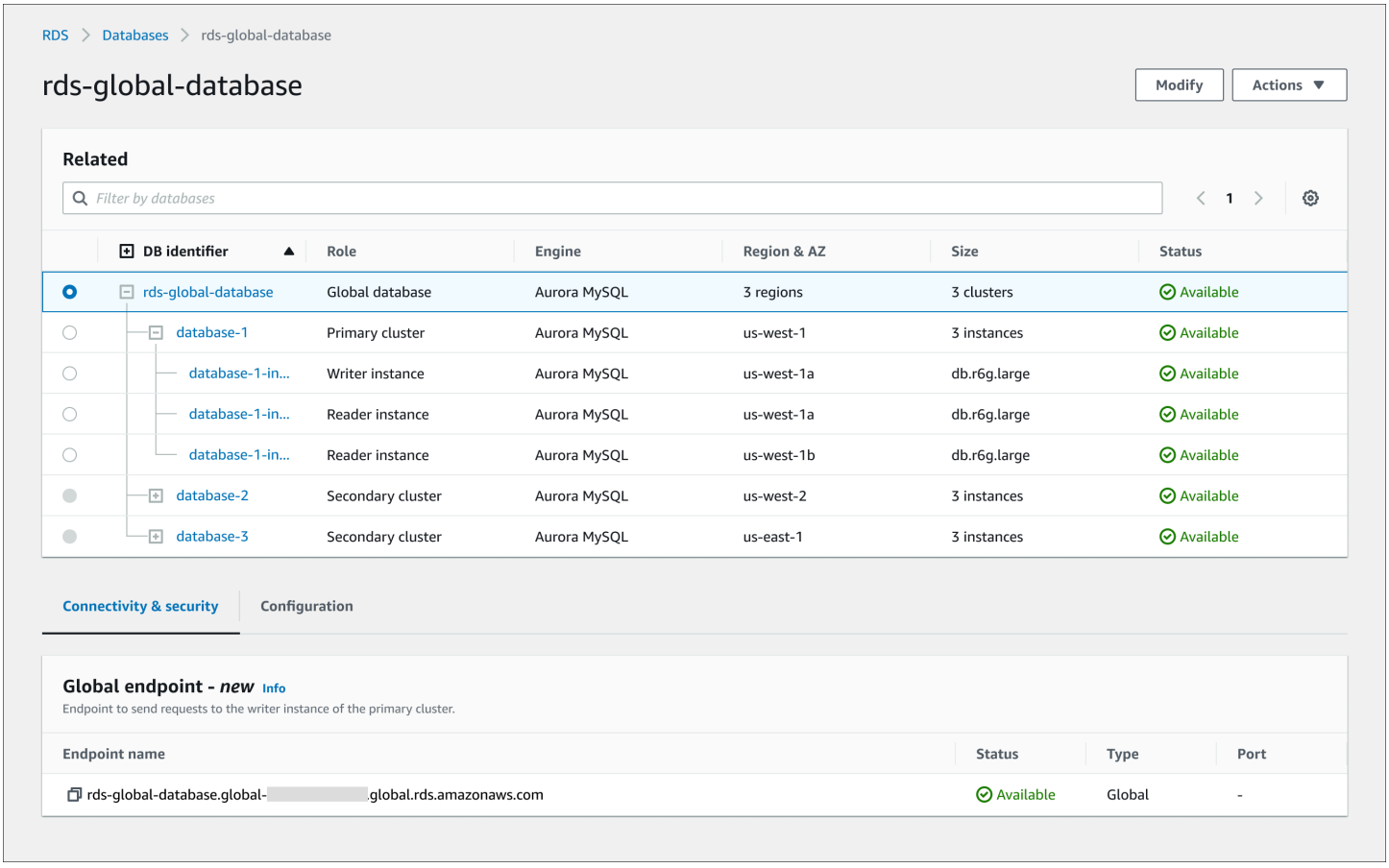
  Untuk mempelajari lebih lanjut tentang menggunakan penerusan tulis di seluruh AWS Wilayah, lihat. Menggunakan penerusan menulis dalam basis data global Amazon Aurora

Untuk detail tentang berbagai jenis titik akhir Aurora, lihat. Menghubungkan ke klaster DB Amazon Aurora

# Melihat titik akhir database global Amazon Aurora

Saat Anda melihat Aurora Global Database di konsol, Anda dapat melihat semua titik akhir yang terkait dengan semua klasternya. Gambar berikut menunjukkan contoh jenis titik akhir yang Anda lihat saat Anda melihat detail untuk cluster DB utama Anda:

- Penulis global — read/write Titik akhir tunggal yang selalu menunjuk ke instans DB penulis saat ini untuk cluster database global.
- Writer — Titik akhir koneksi untuk read/write permintaan ke cluster DB utama di cluster database global.
- Pembaca — Titik akhir koneksi untuk permintaan hanya-baca ke cluster DB primer atau sekunder di cluster database global. Untuk meminimalkan latensi, pilih titik akhir pembaca mana yang ada di Anda Wilayah AWS atau yang paling dekat dengan Anda. Wilayah AWS



Konsol

Untuk melihat titik akhir dari database global

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Dalam daftar, pilih database global, atau cluster DB primer atau sekunder yang titik akhirnya ingin Anda lihat.
4. Pilih tab Konektivitas & keamanan untuk melihat detail titik akhir. Titik akhir yang ditampilkan bergantung pada jenis cluster yang Anda pilih, sebagai berikut:

   - Database global — Titik akhir penulis global.
   - Cluster DB Primer — Titik akhir penulis global, dan titik akhir cluster dan titik akhir pembaca untuk cluster primer.
   - Cluster DB sekunder — Titik akhir cluster dan titik akhir pembaca untuk cluster sekunder. Pada klaster sekunder, titik akhir klaster menampilkan status tidak aktif karena tidak menangani permintaan tulis. Anda masih dapat terhubung ke titik akhir cluster, tetapi hanya untuk kueri baca.

AWS CLI

Untuk melihat titik akhir penulis cluster global, gunakan AWS CLI describe-global-clustersperintah, seperti pada contoh berikut.

```
aws rds describe-global-clusters --region aws_region
{
    "GlobalClusters": [
        {
            "GlobalClusterIdentifier": "global_cluster_id",
            "GlobalClusterResourceId": "cluster-unique_string",
            "GlobalClusterArn": "arn:aws:rds::123456789012:global-
cluster:global_cluster_id",
            "Status": "available",
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",
            "GlobalClusterMembers": [

              ...
```

```
            ],
            "Endpoint":
 "global_cluster_id.global-unique_string.global.rds.amazonaws.com"
        }
    ]
}
```

Untuk melihat titik akhir cluster dan pembaca untuk cluster DB anggota cluster global, gunakan AWS CLI describe-db-clustersperintah, seperti pada contoh berikut. Nilai yang dikembalikan untuk `Endpoint` dan `ReaderEndpoint` merupakan titik akhir cluster dan pembaca, masing-masing.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "az_1",
                "az_2",
                "az_3"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "db_cluster_id",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "DBSubnetGroup": "default",
            "Status": "available",
            "EarliestRestorableTime": "2023-08-01T18:21:11.301Z",
            "Endpoint":
 "db_cluster_id.cluster-unique_string.primary_region.rds.amazonaws.com",
            "ReaderEndpoint": "db_cluster_id.cluster-
ro-unique_string.primary_region.rds.amazonaws.com",
            "MultiAZ": false,
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",

            "ReadReplicaIdentifiers": [
                "arn:aws:rds:secondary_region:123456789012:cluster:db_cluster_id"
            ],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "db_instance_id",
                    "IsClusterWriter": true,
```

```
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                }
            ],

            ...
            "TagList": [],
            "GlobalWriteForwardingRequested": false
        }
    ]
}
```

API RDS

Untuk melihat titik akhir penulis klaster global, gunakan operasi RDS API DescribeGlobalClusters. Untuk melihat titik akhir cluster dan pembaca untuk kluster DB anggota klaster global, gunakan operasi RDS API Description. DBClusters

## Pertimbangan dengan menggunakan titik akhir penulis Global

Anda dapat memanfaatkan titik akhir penulis Aurora Global Database secara efektif dengan mengikuti pedoman dan praktik terbaik ini:

- Untuk meminimalkan gangguan setelah failover atau peralihan lintas wilayah, Anda dapat mengatur konektivitas VPC antara komputasi aplikasi dan Wilayah primer dan sekunder. AWS Misalnya, Anda memiliki aplikasi atau sistem klien yang berjalan di VPC yang sama dengan cluster utama. Jika cluster sekunder dipromosikan, titik akhir penulis global secara otomatis berubah untuk menunjuk ke cluster itu. Meskipun titik akhir penulis global memungkinkan Anda menghindari mengubah pengaturan koneksi untuk aplikasi Anda, aplikasi Anda tidak dapat mengakses alamat IP di VPC AWS Wilayah utama yang baru dipromosikan hingga Anda mengatur jaringan di antara keduanya. VPCs Lihat opsi konektivitas Amazon VPC-to-Amazon VPC untuk mengevaluasi berbagai opsi untuk mengatur konektivitas ini.
- Pembaruan titik akhir penulis global setelah failover atau peralihan basis data global dapat memakan waktu lama tergantung pada durasi caching Layanan Nama Domain (DNS) Anda. Lihat Buku Pegangan Administrator Basis Data MySQL Amazon Aurora untuk mempelajari lebih lanjut. Aurora Global Database memancarkan Peristiwa RDS ketika melihat perubahan DNS pada titik akhir penulis global. Anda dapat menggunakan acara untuk merancang strategi untuk memastikan cache DNS tidak melampaui waktu setelah acara dibuat. Untuk informasi selengkapnya, lihat Peristiwa klaster DB.

- Aurora Global Database mereplikasi data secara asinkron. Metode failover lintas wilayah dapat menghasilkan beberapa data transaksi tulis yang tidak direplikasi ke sekunder yang dipilih sebelum failover dimulai. Meskipun Aurora mencoba dengan upaya terbaik untuk memblokir penulisan di AWS Wilayah primer asli, failover dapat rentan terhadap masalah otak terbelah. Pertimbangan untuk meminimalkan kehilangan data dan risiko otak terbelah juga berlaku untuk titik akhir penulis Aurora Global Database. Lihat informasi yang lebih lengkap di Melakukan failover terkelola untuk basis data global Aurora.

# Menggunakan penerusan menulis dalam basis data global Amazon Aurora

Anda dapat mengurangi jumlah titik akhir yang perlu dikelola untuk aplikasi yang berjalan di basis data global Aurora Anda, dengan menggunakan penerusan tulis. Dengan penerusan tulis diaktifkan, klaster sekunder dalam basis data global Aurora meneruskan pernyataan SQL yang melakukan operasi tulis ke klaster primer. Klaster primer memperbarui sumber kemudian menyebarkan perubahan yang dihasilkan kembali ke semua Wilayah AWS sekunder.

Dengan konfigurasi penerusan tulis, Anda tidak perlu menerapkan mekanisme Anda sendiri untuk mengirimkan operasi tulis dari Wilayah AWS sekunder ke Wilayah primer. Aurora menangani pengaturan jaringan lintas Wilayah. Aurora juga mengirimkan semua sesi yang diperlukan dan konteks transaksional untuk setiap pernyataan. Data selalu diubah terlebih dahulu pada klaster primer kemudian direplikasi ke klaster sekunder dalam basis data global Aurora. Dengan cara ini, klaster primer adalah sumber kebenaran dan selalu memiliki salinan terkini dari semua data Anda.

Topik

- Menggunakan penerusan tulis di dalam basis data global Aurora MySQL
- Menggunakan penerusan tulis di dalam basis data global Aurora PostgreSQL

## Menggunakan penerusan tulis di dalam basis data global Aurora MySQL

Topik

- Ketersediaan wilayah dan versi penerusan tulis di Aurora MySQL
- Mengaktifkan penerusan tulis di Aurora MySQL
- Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora MySQL
- Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora MySQL

- Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL
- Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora MySQL
- Transaksi dengan penerusan tulis di Aurora MySQL
- Parameter konfigurasi untuk penerusan tulis di Aurora MySQL
- CloudWatch Metrik Amazon untuk penerusan tulis di Aurora MySQL
- Variabel status Aurora MySQL untuk penerusan tulis

## Ketersediaan wilayah dan versi penerusan tulis di Aurora MySQL

Penerusan tulis didukung dengan Aurora MySQL 2.08.1 dan versi yang lebih tinggi, di setiap Wilayah di mana basis data global berbasis Aurora MySQL tersedia.

Untuk informasi tentang versi dan ketersediaan Wilayah basis data global Aurora MySQL, lihat Basis data global Aurora dengan Aurora MySQL.

## Mengaktifkan penerusan tulis di Aurora MySQL

Secara default, penerusan tulis tidak diaktifkan saat Anda menambahkan klaster sekunder ke basis data global Aurora.

Untuk mengaktifkan penerusan tulis menggunakan AWS Management Console, pilih kotak centang Aktifkan penerusan tulis global di bawah Baca replika tulis penerusan saat Anda menambahkan Wilayah untuk database global. Untuk klaster sekunder yang sudah ada, ubah klaster tersebut untuk Mengaktifkan penerusan tulis global. Untuk menonaktifkan penerusan tulis, kosongkan Aktifkan penerusan tulis global saat menambahkan Wilayah atau memodifikasi klaster sekunder.

Untuk mengaktifkan penerusan tulis menggunakan AWS CLI, gunakan opsi. `--enable-global-write-forwarding` Opsi ini berfungsi saat Anda membuat klaster sekunder baru menggunakan perintah `create-db-cluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster sekunder yang ada dengan menggunakan perintah `modify-db-cluster`. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan menggunakan opsi `--no-enable-global-write-forwarding` dengan perintah CLI yang sama ini.

Untuk mengaktifkan penerusan tulis menggunakan API Amazon RDS, atur parameter `EnableGlobalWriteForwarding` ke `true`. Parameter ini berfungsi saat Anda membuat klaster sekunder baru menggunakan operasi `CreateDBCluster`. Opsi ini juga berfungsi saat Anda

memodifikasi klaster sekunder yang ada dengan menggunakan operasi `ModifyDBCluster`. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan mengatur parameter `EnableGlobalWriteForwarding` ke `false`.

> **Note**
>
> Agar sesi basis data memanfaatkan penerusan tulis, tentukan pengaturan untuk parameter konfigurasi `aurora_replica_read_consistency`. Lakukan hal ini di setiap sesi yang menggunakan fitur penerusan tulis. Untuk informasi tentang parameter ini, lihat Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL.
>
> Fitur RDS Proxy tidak mendukung nilai `SESSION` untuk variabel `aurora_replica_read_consistency`. Mengatur nilai ini dapat menghasilkan perilaku tak terduga.

Contoh CLI berikut menunjukkan bagaimana Anda dapat menyiapkan basis data global Aurora dengan penerusan tulis diaktifkan atau dinonaktifkan. Item yang disorot mewakili perintah dan opsi yang penting untuk menentukan dan menjaga infrastruktur yang konsisten untuk basis data global Aurora.

Contoh berikut membuat basis data global Aurora, klaster primer, dan klaster sekunder dengan penerusan tulis diaktifkan. Ganti pilihan Anda sendiri untuk nama pengguna, kata sandi, dan Wilayah AWS primer dan sekunder.

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
```

```
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

Contoh berikut adalah lanjutan contoh sebelumnya. Contoh ini membuat klaster sekunder tanpa mengaktifkan penerusan tulis, kemudian mengaktifkan penerusan tulis. Setelah contoh ini selesai, semua klaster sekunder dalam basis data global mengaktifkan penerusan tulis.

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
```

```
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

## Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora MySQL

Untuk menentukan apakah Anda dapat menggunakan penerusan tulis dari klaster sekunder, Anda dapat memeriksa apakah klaster tersebut memiliki atribut `"GlobalWriteForwardingStatus": "enabled"`.

Di AWS Management Console, pada tab Konfigurasi halaman detail untuk cluster, Anda melihat status Enabled for Global read replica write forwarding.

Untuk melihat status setelan penerusan tulis global untuk semua cluster Anda, jalankan perintah berikut. AWS CLI

Klaster sekunder menunjukkan nilai `"enabled"` atau `"disabled"` untuk menunjukkan apakah penerusan tulis diaktifkan atau dinonaktifkan. Nilai `null` menunjukkan bahwa penerusan tulis tidak tersedia untuk klaster tersebut. Klaster ini bukan bagian dari basis data global, atau merupakan klaster primer, bukan klaster sekunder. Nilainya juga bisa `"enabling"` atau `"disabling"` jika penerusan tulis sedang dalam proses diaktifkan atau dinonaktifkan.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu

[
    {
        "GlobalWriteForwardingStatus": "enabled",
```

```
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

Untuk menemukan semua klaster sekunder yang memiliki penerusan tulis global, jalankan perintah berikut. Perintah ini juga mengembalikan titik akhir pembaca klaster ini. Anda menggunakan titik akhir pembaca klaster sekunder ketika Anda menggunakan penerusan tulis dari sekunder ke primer dalam basis data global Aurora Anda.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

## Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora MySQL

Anda dapat menggunakan jenis pernyataan SQL berikut dengan penerusan tulis:

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`. Ada beberapa pembatasan pada properti pernyataan ini yang dapat Anda gunakan dengan penerusan tulis, seperti yang dijelaskan di bawah ini.
- Pernyataan `SELECT ... LOCK IN SHARE MODE` dan `SELECT FOR UPDATE`.
- Pernyataan `PREPARE` dan `EXECUTE`.

Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil usang saat Anda menggunakannya dalam basis data global dengan penerusan tulis. Dengan demikian, pengaturan `EnableGlobalWriteForwarding` dinonaktifkan secara default untuk klaster sekunder. Sebelum mengaktifkan pengaturan tersebut, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.

Batasan berikut berlaku untuk pernyataan SQL yang Anda gunakan untuk penerusan tulis. Dalam beberapa kasus, Anda dapat menggunakan pernyataan pada klaster sekunder dengan penerusan tulis yang diaktifkan pada tingkat klaster. Pendekatan ini berfungsi jika penerusan tulis tidak diaktifkan dalam sesi oleh parameter konfigurasi `aurora_replica_read_consistency`. Mencoba menggunakan pernyataan saat tidak diizinkan karena penerusan tulis menyebabkan pesan kesalahan dengan format berikut.

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

Bahasa definisi data (DDL)

Buat koneksi ke klaster primer untuk menjalankan pernyataan DDL. Anda tidak dapat menjalankannya dari instans DB pembaca.

Memperbarui tabel permanen menggunakan data dari tabel sementara

Anda dapat menggunakan tabel sementara pada klaster sekunder dengan penerusan tulis yang diaktifkan. Namun, Anda tidak dapat menggunakan pernyataan DML untuk mengubah tabel permanen jika pernyataan tersebut mengacu pada tabel sementara. Misalnya, Anda tidak dapat menggunakan pernyataan `INSERT ... SELECT` yang mengambil data dari tabel sementara. Tabel sementara ada pada klaster sekunder dan tidak tersedia ketika pernyataan berjalan pada klaster primer.

Transaksi XA

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster sekunder yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Sebelum mengaktifkan penerusan tulis dalam sebuah sesi, periksa apakah kode Anda menggunakan pernyataan ini.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
```

```
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

Pernyataan LOAD untuk tabel permanen

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder dengan penerusan tulis yang diaktifkan.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

Anda dapat memuat data ke dalam tabel sementara pada klaster sekunder. Namun, pastikan Anda menjalankan semua pernyataan `LOAD` yang mengacu pada tabel permanen hanya ada pada klaster primer.

Pernyataan plugin

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder dengan penerusan tulis yang diaktifkan.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

Pernyataan SAVEPOINT

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster sekunder yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Periksa apakah kode Anda menggunakan pernyataan ini sebelum mengaktifkan penerusan tulis dalam sebuah sesi.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

## Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL

Dalam sesi yang menggunakan penerusan tulis, Anda hanya dapat menggunakan tingkat isolasi `REPEATABLE READ`. Meskipun Anda juga dapat menggunakan tingkat isolasi `READ COMMITTED`

dengan klaster hanya-baca di Wilayah AWS sekunder, tingkat isolasi tersebut tidak berfungsi dengan penerusan tulis. Untuk informasi tentang tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`, lihat Tingkat isolasi Aurora MySQL.

Anda dapat mengontrol tingkat konsistensi baca di klaster sekunder. Tingkat konsistensi baca menentukan seberapa lama waktu tunggu klaster sekunder sebelum setiap operasi baca untuk memastikan bahwa beberapa atau semua perubahan direplikasi dari klaster primer. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi tulis yang diteruskan dari sesi Anda terlihat dalam klaster sekunder sebelum kueri berikutnya. Anda juga dapat menggunakan pengaturan ini untuk memastikan bahwa kueri pada klaster sekunder selalu melihat pembaruan terkini dari klaster primer. Hal ini bahkan berlaku untuk yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, Anda memilih nilai untuk parameter tingkat sesi `aurora_replica_read_consistency`.

> **Important**
>
> Selalu atur parameter `aurora_replica_read_consistency` untuk sesi apa pun yang ingin Anda terapkan penerusan tulis. Jika tidak, Aurora tidak mengaktifkan penerusan tulis untuk sesi tersebut. Parameter ini mempunyai nilai kosong secara default, jadi pilih nilai tertentu apabila Anda menggunakan parameter ini. Parameter `aurora_replica_read_consistency` memiliki efek hanya pada klaster sekunder dengan penerusan tulis diaktifkan.
>
> Untuk Aurora MySQL versi 2 dan versi 3 sebelum 3.04, gunakan `aurora_replica_read_consistency` sebagai variabel sesi. Untuk Aurora MySQL versi 3.04 dan lebih tinggi, Anda dapat menggunakan `aurora_replica_read_consistency` sebagai variabel sesi atau sebagai parameter klaster DB.

Untuk parameter `aurora_replica_read_consistency`, Anda dapat menentukan nilai `EVENTUAL`, `SESSION`, dan `GLOBAL`.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu menunggu perubahan disebarkan antar AWS Wilayah. Anda dapat memilih keseimbangan antara waktu respons yang cepat dan memastikan bahwa perubahan yang dilakukan di dalam lokasi lainnya sepenuhnya tersedia sebelum kueri Anda berjalan.

Dengan konsistensi baca yang disetel ke`EVENTUAL`, kueri di AWS Wilayah sekunder yang menggunakan penerusan tulis mungkin melihat data yang sedikit basi karena kelambatan replikasi.

Hasil operasi tulis dalam sesi yang sama tidak terlihat sampai operasi tulis dilakukan pada Wilayah primer dan direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi.

Dengan konsistensi baca disetel ke`SESSION`, semua kueri di AWS Wilayah sekunder yang menggunakan penerusan tulis melihat hasil dari semua perubahan yang dibuat dalam sesi tersebut. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi tulis yang diteruskan untuk direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui dari operasi tulis yang dilakukan di Wilayah lain atau dalam sesi lain di dalam Wilayah saat ini.

Dengan konsistensi baca diatur ke`GLOBAL`, sesi di AWS Wilayah sekunder melihat perubahan yang dibuat oleh sesi itu. Ini juga melihat semua perubahan yang dilakukan dari AWS Wilayah primer dan AWS Wilayah sekunder lainnya. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlangsung ketika cluster sekunder up-to-date dengan semua data yang dikomit dari cluster primer, pada saat kueri dimulai.

Untuk informasi selengkapnya tentang semua parameter yang terlibat dalam penerusan tulis, lihat Parameter konfigurasi untuk penerusan tulis di Aurora MySQL.

Contoh menggunakan penerusan tulis

Contoh-contoh ini menggunakan `aurora_replica_read_consistency` sebagai variabel sesi. Untuk Aurora MySQL versi 3.04 dan lebih tinggi, Anda dapat menggunakan `aurora_replica_read_consistency` sebagai variabel sesi atau sebagai parameter klaster DB.

Dalam contoh ini, klaster primer berada di Wilayah AS Timur (Virginia Utara). Klaster sekunder berada di Wilayah AS Timur (Ohio). Contoh ini menunjukkan efek menjalankan pernyataan `INSERT` yang diikuti dengan pernyataan `SELECT`. Tergantung nilai pengaturan `aurora_replica_read_consistency`, hasilnya mungkin berbeda tergantung waktu pernyataan. Untuk mencapai konsistensi yang lebih tinggi, Anda mungkin menunggu sebentar sebelum mengeluarkan pernyataan `SELECT`. Atau Aurora dapat secara otomatis menunggu sampai hasil selesai mereplikasi sebelum melanjutkan dengan `SELECT`.

Dalam contoh ini, ada pengaturan konsistensi baca `eventual`. Menjalankan pernyataan `INSERT` yang langsung diikuti oleh pernyataan `SELECT` tetap mengembalikan nilai `COUNT(*)`. Nilai ini mencerminkan jumlah baris sebelum baris baru disisipkan. Dengan menjalankan `SELECT` lagi beberapa saat kemudian, muncul hitungan baris yang diperbarui. Pernyataan `SELECT` tidak menunggu.

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca `session`, pernyataan `SELECT` langsung setelah `INSERT` akan menunggu sampai perubahan dari pernyataan `INSERT` terlihat. Pernyataan `SELECT` selanjutnya tidak menunggu.

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
```

```
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca tetap disetel ke `session`, memperkenalkan waktu tunggu singkat setelah melaksanakan pernyataan `INSERT` membuat hitungan baris yang diperbarui tersedia pada saat pernyataan `SELECT` berikutnya berjalan.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca `global`, setiap pernyataan `SELECT` menunggu untuk memastikan bahwa semua data berubah sejak waktu mulai laporan terlihat sebelum melakukan kueri. Durasi tunggu untuk setiap pernyataan `SELECT` berbeda-beda, tergantung jumlah lag replikasi antara klaster primer dan sekunder.

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
```

```
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

## Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora MySQL

Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke klaster primer dan berjalan di sana.

## Transaksi dengan penerusan tulis di Aurora MySQL

Apakah transaksi diteruskan ke klaster primer ditentukan oleh mode akses transaksi. Anda dapat menentukan mode akses untuk transaksi dengan menggunakan pernyataan `SET TRANSACTION` atau pernyataan `START TRANSACTION`. Anda juga dapat menentukan mode akses transaksi dengan mengubah nilai variabel sesi transaction_read_only. Anda dapat mengubah nilai sesi ini hanya saat Anda terhubung dengan klaster DB yang memiliki penerusan tulis diaktifkan.

Jika transaksi yang berlangsung lama tidak mengeluarkan pernyataan apa pun dalam waktu yang lama, transaksi tersebut kemungkinan melebihi periode batas waktu idle. Periode ini memiliki nilai default satu menit. Anda dapat meningkatkannya hingga satu hari. Transaksi yang melebihi batas waktu idle dibatalkan oleh klaster primer. Pernyataan berikutnya yang Anda kirimkan akan menerima kesalahan waktu habis. Kemudian Aurora akan mengembalikan transaksi.

Jenis kesalahan ini dapat terjadi dalam kasus lain ketika penerusan tulis menjadi tidak tersedia. Misalnya, Aurora membatalkan transaksi apa pun yang menggunakan penerusan tulis jika Anda memulai ulang klaster primer atau jika Anda menonaktifkan pengaturan konfigurasi penerusan tulis.

## Parameter konfigurasi untuk penerusan tulis di Aurora MySQL

Grup parameter klaster Aurora mencakup pengaturan untuk fitur penerusan tulis. Karena ini adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk variabel ini.

Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

| Nama | Cakupan | Tipe | Nilai default | Nilai valid |
|---|---|---|---|---|
| `aurora_fwd_master_idle_timeout` (Aurora MySQL versi 2) | Global | integer tanpa tanda | 60 | 1-86.400 |
| `aurora_fwd_master_max_connections_pct` (Aurora MySQL versi 2) | Global | integer panjang tanpa tanda | 10 | 0–90 |
| `aurora_fwd_writer_idle_timeout` (Aurora MySQL versi 3) | Global | integer tanpa tanda | 60 | 1-86.400 |
| `aurora_fwd_writer_max_connections_pct` (Aurora MySQL versi 3) | Global | integer panjang tanpa tanda | 10 | 0–90 |
| `aurora_replica_read_consistency` | Sesi untuk versi 2 dan versi 3 lebih rendah dari 3.04, Global untuk versi 3.04 dan lebih tinggi | Enum | " (null) | `EVENTUAL`, `SESSION`, `GLOBAL` |

Untuk mengontrol permintaan tulis yang masuk dari klaster sekunder, gunakan pengaturan ini pada klaster primer:

- `aurora_fwd_master_idle_timeout`, `aurora_fwd_writer_idle_timeout`: Jumlah detik yang diperlukan klaster primer untuk menunggu aktivitas pada koneksi yang diteruskan dari klaster sekunder sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan membatalkan sesi.
- `aurora_fwd_master_max_connections_pct`, `aurora_fwd_writer_max_connections_pct`: Batas atas pada koneksi basis data yang dapat digunakan pada instans DB penulis untuk menangani kueri yang diteruskan dari pembaca. Opsi ini dinyatakan dalam bentuk persentase pengaturan `max_connections` untuk instans DB dalam klaster primer. Misalnya, jika `max_connections` adalah 800 dan `aurora_fwd_master_max_connections_pct` atau `aurora_fwd_writer_max_connections_pct` adalah 10, maka penulis mengizinkan maksimal 80 sesi terusan serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan `max_connections`.

  Pengaturan ini hanya berlaku pada klaster primer, saat satu atau beberapa klaster sekunder memiliki penerusan tulis yang diaktifkan. Jika Anda menurunkan nilai, koneksi yang ada tidak akan terpengaruh. Aurora mempertimbangkan nilai baru pengaturan saat mencoba membuat koneksi baru dari klaster sekunder. Nilai default adalah 10, mewakili 10% dari nilai `max_connections` . Jika Anda mengaktifkan penerusan kueri di klaster sekunder mana pun, pengaturan ini harus memiliki nilai bukan nol agar operasi tulis dari klaster sekunder dapat berhasil. Jika nilainya nol, operasi tulis menerima kode kesalahan `ER_CON_COUNT_ERROR` dengan pesan `Not enough connections on writer to handle your request`.

`aurora_replica_read_consistency`Parameter memungkinkan penerusan tulis. Anda menggunakannya dalam setiap sesi. Anda dapat menentukan `EVENTUAL`, `SESSION`, atau `GLOBAL` untuk tingkat konsistensi baca. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL. Aturan berikut berlaku untuk parameter ini:

- Nilai default adalah " (kosong).
- Penerusan tulis tersedia dalam sebuah sesi hanya jika `aurora_replica_read_consistency` diatur ke `EVENTUAL` atau `SESSION` atau `GLOBAL`. Parameter ini hanya relevan dalam instans pembaca klaster sekunder yang memiliki penerusan tulis diaktifkan dan yang berada di dalam basis data global Aurora.
- Anda tidak dapat mengatur variabel ini (saat kosong) atau membatalkan pengaturan (saat sudah diatur) di dalam transaksi multipernyataan. Namun, Anda dapat mengubahnya dari satu nilai

yang valid (`EVENTUAL`, `SESSION`, atau `GLOBAL`) ke nilai valid lainnya (`EVENTUAL`, `SESSION`, atau `GLOBAL`) selama transaksi tersebut.

- Variabel tidak boleh `SET` ketika penerusan tulis tidak diaktifkan pada klaster sekunder.

## CloudWatch Metrik Amazon untuk penerusan tulis di Aurora MySQL

CloudWatch Metrik Amazon berikut berlaku untuk klaster utama saat Anda menggunakan penerusan tulis pada satu atau beberapa klaster sekunder. Metrik ini semua diukur pada DB instance dalam cluster primer.

| CloudWatch metrik | Unit | Deskripsi |
|---|---|---|
| `AuroraDMLRejectedMasterFull` | Hitung | Jumlah kueri yang diteruskan yang ditolak karena sesi penuh pada instance DB penulis.<br><br>Untuk Aurora MySQL versi 2. |
| `AuroraDMLRejectedWriterFull` | Hitung | Jumlah kueri yang diteruskan yang ditolak karena sesi penuh pada instance DB penulis.<br><br>Untuk Aurora MySQL versi 3. |
| `ForwardingMasterDMLLatency` | Milidetik | Rata-rata waktu untuk memproses setiap pernyataan DML yang diteruskan pada instans DB penulis.<br><br>Ini tidak termasuk waktu yang diperlukan klaster sekunder untuk meneruskan permintaan tulis, atau waktu untuk mereplikasi perubahan kembali ke klaster sekunder. |

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| | | Untuk Aurora MySQL versi 2. |
| `ForwardingMasterDMLThroughput` | Hitungan per detik | Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini.<br><br>Untuk Aurora MySQL versi 2. |
| `ForwardingMasterOpenSessions` | Hitungan | Jumlah sesi yang diteruskan pada instans DB tulis.<br><br>Untuk Aurora MySQL versi 2. |
| `ForwardingWriterDMLLatency` | Milidetik | Rata-rata waktu untuk memproses setiap pernyataan DML yang diteruskan pada instans DB penulis.<br><br>Ini tidak termasuk waktu yang diperlukan klaster sekunder untuk meneruskan permintaan tulis, atau waktu untuk mereplikasi perubahan kembali ke klaster sekunder.<br><br>Untuk Aurora MySQL versi 3. |
| `ForwardingWriterDMLThroughput` | Hitungan per detik | Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini.<br>Untuk Aurora MySQL versi 3. |
| `ForwardingWriterOpenSessions` | Hitungan | Jumlah sesi yang diteruskan pada instans DB tulis.<br>Untuk Aurora MySQL versi 3. |

CloudWatch Metrik berikut berlaku untuk setiap cluster sekunder. Metrik ini diukur pada setiap DB instance pembaca dalam cluster sekunder dengan penerusan tulis yang diaktfikan.

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| `ForwardingReplicaDMLLatency` | Milidetik | Waktu respons rata-rata diteruskan DMLs pada replika. |
| `ForwardingReplicaDMLThroughput` | Hitungan per detik | Jumlah pernyataan DML yang diteruskan yang diproses setiap detiknya. |
| `ForwardingReplicaOpenSessions` | Hitungan | Jumlah sesi yang menggunakan penerusan tulis pada instans DB pembaca. |
| `ForwardingReplicaReadWaitLatency` | Milidetik | Rata-rata waktu tunggu yang diperlukan sebuah pernyataan `SELECT` di instans DB pembaca untuk menunggu agar dapat mengimbangi klaster primer.<br><br>Batas tunggu instans DB pembaca sebelum memproses kueri ditentukan oleh pengaturan `aurora_replica_read_consistency` . |
| `ForwardingReplicaReadWaitThroughput` | Hitungan per detik | Jumlah total pernyataan `SELECT` yang diproses setiap detik di semua sesi yang meneruskan tulis. |
| `ForwardingReplicaSelectLatency` | Milidetik | Latensi `SELECT` yang diteruskan, rata-rata atas semua pernyataan `SELECT` |

| CloudWatch metrik | Unit | Deskripsi |
| --- | --- | --- |
| | | yang diteruskan dalam periode pemantauan. |
| `ForwardingReplicaSelectThroughput` | Hitungan per detik | Rata-rata throughput `SELECT` per detik yang diteruskan dalam periode pemantauan. |

## Variabel status Aurora MySQL untuk penerusan tulis

Variabel status MySQL Aurora berikut berlaku untuk cluster utama ketika Anda menggunakan penerusan tulis pada satu atau lebih cluster sekunder. Semua metrik ini diukur pada instans DB penulis dalam klaster primer.

| Variabel status Aurora MySQL | Unit | Deskripsi |
| --- | --- | --- |
| `Aurora_fwd_master_dml_stmt_count` | Hitung | Total jumlah pernyataan DML yang diteruskan ke instans DB penulis ini.<br>Untuk Aurora MySQL versi 2. |
| `Aurora_fwd_master_dml_stmt_duration` | Mikrodetik | Total durasi pernyataan DML yang diteruskan ke instans DB penulis ini.<br>Untuk Aurora MySQL versi 2. |
| `Aurora_fwd_master_open_sessions` | Hitungan | Jumlah sesi yang diteruskan pada instans DB tulis.<br>Untuk Aurora MySQL versi 2. |
| `Aurora_fwd_master_select_stmt_count` | Hitungan | Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB penulis ini.<br>Untuk Aurora MySQL versi 2. |

| Variabel status Aurora MySQL | Unit | Deskripsi |
| --- | --- | --- |
| `Aurora_fwd_master_select_stmt_duration` | Mikrodetik | Total durasi pernyataan `SELECT` yang diteruskan ke instans DB penulis ini.<br><br>Untuk Aurora MySQL versi 2. |
| `Aurora_fwd_writer_dml_stmt_count` | Hitungan | Total jumlah pernyataan DML yang diteruskan ke instans DB penulis ini.<br>Untuk Aurora MySQL versi 3. |
| `Aurora_fwd_writer_dml_stmt_duration` | Mikrodetik | Total durasi pernyataan DML yang diteruskan ke instans DB penulis ini. |
| `Aurora_fwd_writer_open_sessions` | Hitung | Jumlah sesi yang diteruskan pada instans DB tulis.<br>Untuk Aurora MySQL versi 3. |
| `Aurora_fwd_writer_select_stmt_count` | Hitung | Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB penulis ini.<br>Untuk Aurora MySQL versi 3. |
| `Aurora_fwd_writer_select_stmt_duration` | Mikrodetik | Total durasi pernyataan `SELECT` yang diteruskan ke instans DB penulis ini.<br><br>Untuk Aurora MySQL versi 3. |

Variabel status MySQL Aurora berikut berlaku untuk setiap cluster sekunder. Metrik-metrik ini diukur pada setiap instans DB pembaca dalam klaster sekunder dengan penerusan tulis yang diaktifkan.

| Variabel status Aurora MySQL | Unit | Deskripsi |
| --- | --- | --- |
| `Aurora_fwd_replica_dml_stmt_count` | Hitung | Total jumlah pernyataan DML yang diteruskan ke instans DB pembaca ini. |
| `Aurora_fwd_replica_dml_stmt_duration` | Mikrodetik | Total durasi semua pernyataan DML yang diteruskan ke instans DB pembaca ini. |
| `Aurora_fwd_replica_errors_session_limit` | Hitungan | Jumlah sesi yang ditolak oleh klaster primer karena salah satu kondisi kesalahan berikut:<br>• penulis penuh<br>• Terlalu banyak pernyataan yang diteruskan dalam proses. |
| `Aurora_fwd_replica_open_sessions` | Hitung | Jumlah sesi yang menggunakan penerusan tulis pada instans DB pembaca. |
| `Aurora_fwd_replica_read_wait_count` | Hitung | Jumlah total read-after-write menunggu pada instans DB pembaca ini. |
| `Aurora_fwd_replica_read_wait_duration` | Mikrodetik | Total durasi tunggu karena pengaturan konsistensi baca pada instans DB pembaca ini. |
| `Aurora_fwd_replica_select_stmt_count` | Hitungan | Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini. |
| `Aurora_fwd_replica_select_stmt_duration` | Mikrodetik | Total durasi pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini. |

# Menggunakan penerusan tulis di dalam basis data global Aurora PostgreSQL

Topik


- Ketersediaan wilayah dan versi penerusan tulis di Aurora PostgreSQL
- Mengaktifkan penerusan tulis di Aurora PostgreSQL
- Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora PostgreSQL
- Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora PostgreSQL
- Isolasi dan konsistensi untuk penerusan tulis di Aurora PostgreSQL
- Mode akses transaksi dengan penerusan tulis
- Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora PostgreSQL
- Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL
- CloudWatch Metrik Amazon untuk penerusan tulis di Aurora PostgreSQL
- Peristiwa tunggu untuk penerusan tulis di Aurora PostgreSQL


## Ketersediaan wilayah dan versi penerusan tulis di Aurora PostgreSQL

Di Aurora PostgreSQL versi 16 dan versi mayor yang lebih tinggi, penerusan penulisan global didukung di semua versi minor. Untuk versi Aurora PostgreSQL sebelumnya, penerusan penulisan global didukung dengan versi 15.4 dan versi minor yang lebih tinggi, dan versi 14.9 dan versi minor yang lebih tinggi. Penerusan tulis tersedia di setiap AWS Wilayah di mana basis data global berbasis Aurora PostgreSQL tersedia.

Untuk informasi lebih lanjut tentang versi dan ketersediaan Wilayah basis data global Aurora PostgreSQL, lihat Basis data global Aurora dengan Aurora PostgreSQL.

## Mengaktifkan penerusan tulis di Aurora PostgreSQL

Secara default, penerusan tulis tidak diaktifkan saat Anda menambahkan klaster sekunder ke basis data global Aurora. Anda dapat mengaktifkan penerusan tulis untuk klaster DB sekunder Anda saat Anda membuatnya atau kapan saja setelah Anda membuatnya. Jika perlu, Anda dapat menonaktifkannya nanti. Mengaktifkan atau menonaktifkan penerusan tulis tidak menyebabkan waktu henti atau boot ulang.

> **Note**
>
> Anda dapat menggunakan penerusan tulis lokal untuk aplikasi Anda yang sesekali menulis dan memerlukan read-after-write konsistensi, yang merupakan kemampuan untuk membaca tulisan terbaru dalam suatu transaksi.

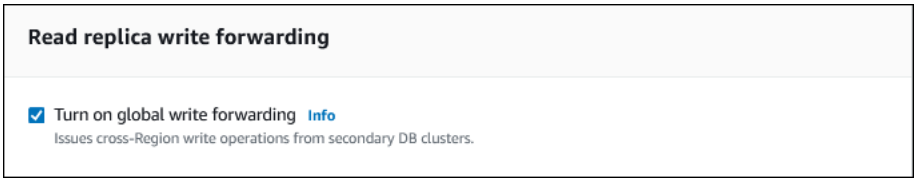
Konsol

Di konsol, Anda dapat mengaktifkan atau menonaktifkan penerusan tulis saat Anda membuat atau memodifikasi klaster DB sekunder.

Mengaktifkan atau menonaktifkan penerusan tulis saat membuat klaster DB sekunder

Saat Anda membuat klaster DB sekunder baru, Anda mengaktifkan penerusan tulis dengan memilih kotak centang Aktifkan penerusan tulis global di bagian Penerusan tulis replika baca. Atau kosongkan kotak centang untuk menonaktifkannya. Untuk membuat klaster DB sekunder, ikuti petunjuk untuk mesin DB Anda di Membuat klaster DB Amazon Aurora.

Tangkapan layar berikut menunjukkan bagian Penerusan tulis replika baca dengan kotak centang Aktifkan penerusan tulis global dipilih.




Mengaktifkan atau menonaktifkan penerusan tulis saat memodifikasi klaster DB sekunder

Di konsol, Anda dapat memodifikasi klaster DB sekunder untuk mengaktifkan atau menonaktifkan penerusan tulis.

Mengaktifkan atau menonaktifkan penerusan tulis untuk klaster DB sekunder dengan menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih klaster DB sekunder, dan pilih Modifikasi.

4. Di bagian Penerusan tulis replika baca, pilih atau kosongkan kotak centang Aktifkan penerusan tulis global.
5. Pilih Lanjutkan.
6. Untuk Jadwalkan modifikasi, pilih Terapkan segera. Jika Anda memilih Terapkan selama periode pemeliharaan terjadwal berikutnya, Aurora akan mengabaikan pengaturan ini dan langsung mengaktifkan penerusan tulis.
7. Pilih Modifikasi klaster.

AWS CLI

Untuk mengaktifkan penerusan tulis dengan menggunakan AWS CLI, gunakan opsi. `--enable-global-write-forwarding` Opsi ini berfungsi saat Anda membuat klaster sekunder baru menggunakan perintah create-db-cluster. Opsi ini juga berfungsi saat Anda memodifikasi klaster sekunder yang ada dengan menggunakan perintah modify-db-cluster. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan menggunakan opsi `--no-enable-global-write-forwarding` dengan perintah CLI yang sama ini.

Prosedur berikut ini menjelaskan cara mengaktifkan atau menonaktifkan penerusan tulis untuk klaster DB sekunder di klaster global Anda dengan menggunakan AWS CLI.

Mengaktifkan atau menonaktifkan penerusan tulis untuk klaster DB sekunder yang ada

- Panggil modify-db-cluster AWS CLI perintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--enable-global-write-forwarding` untuk mengaktifkan atau `--no-enable-global-write-forwarding` untuk menonaktifkan.

  Contoh berikut mengaktifkan penerusan tulis untuk klaster DB `sample-secondary-db-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
    --enable-global-write-forwarding
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

API RDS

Untuk mengaktifkan penerusan tulis menggunakan API Amazon RDS, atur parameter `EnableGlobalWriteForwarding` ke `true`. Parameter ini berfungsi saat Anda membuat cluster sekunder baru menggunakan DBCluster operasi Create. Ini juga berfungsi ketika Anda memodifikasi cluster sekunder yang ada menggunakan DBCluster operasi Modify. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan mengatur parameter `EnableGlobalWriteForwarding` ke `false`.

## Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora PostgreSQL

Untuk menentukan apakah Anda dapat menggunakan penerusan tulis dari klaster sekunder, Anda dapat memeriksa apakah klaster tersebut memiliki atribut `"GlobalWriteForwardingStatus": "enabled"`.

Di AWS Management Console, Anda lihat `Read replica write forwarding` pada tab Konfigurasi halaman detail untuk cluster. Untuk melihat status setelan penerusan tulis global untuk semua cluster Anda, jalankan perintah berikut. AWS CLI

Klaster sekunder menunjukkan nilai `"enabled"` atau `"disabled"` untuk menunjukkan apakah penerusan tulis diaktifkan atau dinonaktifkan. Nilai `null` menunjukkan bahwa penerusan tulis tidak tersedia untuk klaster tersebut. Klaster ini bukan bagian dari basis data global, atau merupakan klaster primer, bukan klaster sekunder. Nilainya juga bisa `"enabling"` atau `"disabling"` jika penerusan tulis sedang dalam proses diaktifkan atau dinonaktifkan.

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
[
    {
        "GlobalWriteForwardingStatus": "enabled",
```

```
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

Untuk menemukan semua klaster sekunder yang memiliki penerusan tulis global, jalankan perintah berikut. Perintah ini juga mengembalikan titik akhir pembaca klaster ini. Anda menggunakan titik akhir pembaca klaster sekunder ketika Anda menggunakan penerusan tulis dari sekunder ke primer dalam basis data global Aurora Anda.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

## Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora PostgreSQL

Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil usang saat Anda menggunakannya dalam basis data global dengan penerusan tulis. Selain itu, fungsi yang ditentukan pengguna dan prosedur yang ditentukan pengguna tidak didukung. Dengan demikian, pengaturan `EnableGlobalWriteForwarding` dinonaktifkan secara default untuk klaster sekunder. Sebelum mengaktifkan pengaturan tersebut, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.

Anda dapat menggunakan jenis pernyataan SQL berikut dengan penerusan tulis:

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`
- Pernyataan `SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }`
- Pernyataan `PREPARE` dan `EXECUTE`
- Pernyataan `EXPLAIN` dengan pernyataan di dalam daftar ini

Jenis pernyataan SQL berikut ini tidak didukung dengan penerusan tulis:

- Pernyataan bahasa definisi data (DDL)
- `ANALYZE`
- `CLUSTER`
- `COPY`
- Kursor - Kursor tidak didukung, jadi pastikan untuk menutupnya sebelum menggunakan penerusan tulis.
- `GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL`
- `LOCK`
- Pernyataan `SAVEPOINT`
- `SELECT INTO`
- `SET CONSTRAINTS`
- `TRUNCATE`
- `VACUUM`

## Isolasi dan konsistensi untuk penerusan tulis di Aurora PostgreSQL

Dalam sesi yang menggunakan penerusan tulis, Anda dapat menggunakan tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`. Namun, tingkat isolasi `SERIALIZABLE` tidak didukung.

Anda dapat mengontrol tingkat konsistensi baca di klaster sekunder. Tingkat konsistensi baca menentukan seberapa lama waktu tunggu klaster sekunder sebelum setiap operasi baca untuk memastikan bahwa beberapa atau semua perubahan direplikasi dari klaster primer. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi tulis yang diteruskan dari sesi Anda terlihat dalam klaster sekunder sebelum kueri berikutnya. Anda juga dapat menggunakan pengaturan ini untuk memastikan bahwa kueri pada klaster sekunder selalu melihat pembaruan terkini dari klaster primer. Hal ini bahkan berlaku untuk yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, Anda memilih

nilai yang sesuai untuk `apg_write_forward.consistency_mode` parameter. Parameter `apg_write_forward.consistency_mode` memiliki efek hanya pada klaster sekunder dengan penerusan tulis diaktifkan.

> **Note**
>
> Untuk parameter `apg_write_forward.consistency_mode`, Anda dapat menentukan nilai `SESSION`, `EVENTUAL`, `GLOBAL`, atau `OFF`. Secara default, nilainya diatur ke `SESSION`. Mengatur nilai ini ke `OFF` akan menonaktifkan penerusan tulis dalam sesi.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu menunggu perubahan disebarkan antar AWS Wilayah. Anda dapat memilih keseimbangan antara waktu respons yang cepat dan memastikan bahwa perubahan yang dilakukan di dalam lokasi lainnya sepenuhnya tersedia sebelum kueri Anda berjalan.

Dengan setiap pengaturan mode konsistensi yang tersedia, efeknya adalah sebagai berikut:

- `SESSION`— Semua kueri di AWS Wilayah sekunder yang menggunakan penerusan tulis melihat hasil dari semua perubahan yang dibuat dalam sesi itu. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi tulis yang diteruskan untuk direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui dari operasi tulis yang dilakukan di Wilayah lain atau dalam sesi lain di dalam Wilayah saat ini.
- `EVENTUAL`— Kueri di AWS Wilayah sekunder yang menggunakan penerusan tulis mungkin melihat data yang sedikit basi karena kelambatan replikasi. Hasil operasi tulis dalam sesi yang sama tidak terlihat sampai operasi tulis dilakukan pada Wilayah primer dan direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi.
- `GLOBAL`— Sesi di AWS Wilayah sekunder melihat perubahan yang dibuat oleh sesi itu. Ini juga melihat semua perubahan yang dilakukan dari AWS Wilayah primer dan AWS Wilayah sekunder lainnya. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlangsung ketika cluster sekunder up-to-date dengan semua data yang dikomit dari cluster primer, pada saat kueri dimulai.
- `OFF` – Penerusan tulis dinonaktifkan dalam sesi.

Untuk informasi selengkapnya tentang semua parameter yang terlibat dalam penerusan tulis, lihat Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL.

### Mode akses transaksi dengan penerusan tulis

Jika mode akses transaksi disetel ke hanya-baca, penerusan tulis tidak digunakan. Anda dapat mengatur mode akses untuk membaca tulis hanya saat Anda terhubung ke cluster DB dan sesi yang telah mengaktifkan penerusan tulis.

Untuk informasi selengkapnya tentang mode akses transaksi, lihat MENGATUR TRANSAKSI.

### Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora PostgreSQL

Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke klaster primer dan berjalan di sana.

### Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL

Grup parameter klaster Aurora mencakup pengaturan untuk fitur penerusan tulis. Karena ini adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk variabel ini. Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

| Nama | Cakupan | Tipe | Nilai default | Nilai valid |
|---|---|---|---|---|
| `apg_write_forward.connect_timeout` | Sesi | detik | 30 | 0–2147483647 |
| `apg_write_forward.consistency_mode` | Sesi | enum | Sesi | `SESSION`, `EVENTUAL`, `GLOBAL`, `OFF` |
| `apg_write_forward.idle_in_transaction_session_timeout` | Sesi | milidetik | 86400000 | 0–2147483647 |
| `apg_write_forward.idle_session_timeout` | Sesi | milidetik | 300000 | 0–2147483647 |

| Nama | Cakupan | Tipe | Nilai default | Nilai valid |
|---|---|---|---|---|
| apg_write_forward.max_forwarding_connections_percent | Global | int | 25 | 1–100 |

Parameter apg_write_forward.max_forwarding_connections_percent adalah batas atas pada slot koneksi basis data yang dapat digunakan untuk menangani kueri yang diteruskan dari pembaca. Opsi ini dinyatakan dalam bentuk persentase pengaturan max_connections untuk instans DB dalam klaster primer. Misalnya, jika max_connections adalah 800 dan apg_write_forward.max_forwarding_connections_percent adalah 10, maka penulis mengizinkan maksimum 80 sesi yang diteruskan secara serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan max_connections. Pengaturan ini hanya berlaku pada klaster primer saat setidaknya ada satu klaster sekunder yang memiliki penerusan tulis yang diaktifkan.

Gunakan pengaturan berikut pada klaster sekunder:

- apg_write_forward.consistency_mode – Parameter tingkat sesi yang mengontrol tingkat konsistensi baca pada klaster sekunder. Nilai yang valid adalah SESSION, EVENTUAL, GLOBAL, atau OFF. Secara default, nilainya diatur ke SESSION. Mengatur nilai ini ke OFF akan menonaktifkan penerusan tulis dalam sesi. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Isolasi dan konsistensi untuk penerusan tulis di Aurora PostgreSQL. Parameter ini hanya relevan dalam instans pembaca klaster sekunder yang memiliki penerusan tulis diaktifkan dan yang berada di dalam basis data global Aurora.
- apg_write_forward.connect_timeout – Jumlah maksimum detik yang diperlukan klaster sekunder untuk menunggu saat membuat koneksi ke klaster primer sebelum menyerah. Nilai 0 berarti menunggu tanpa batas.
- apg_write_forward.idle_in_transaction_session_timeout – Jumlah milidetik yang diperlukan klaster primer untuk menunggu aktivitas pada koneksi yang diteruskan dari klaster sekunder yang memiliki transaksi terbuka sebelum menutupnya. Jika sesi tetap idle dalam transaksi setelah periode ini, Aurora akan mengakhiri sesi. Nilai 0 menonaktifkan batas waktu.
- apg_write_forward.idle_session_timeout – Jumlah milidetik yang diperlukan klaster primer untuk menunggu aktivitas pada koneksi yang diteruskan dari klaster sekunder sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan mengakhiri sesi. Nilai 0 menonaktifkan batas waktu.

## CloudWatch Metrik Amazon untuk penerusan tulis di Aurora PostgreSQL

CloudWatch Metrik Amazon berikut berlaku untuk klaster utama saat Anda menggunakan penerusan tulis pada satu atau beberapa klaster sekunder. Metrik ini semua diukur pada DB instance dalam cluster primer.

| CloudWatch Metrik | Unit dan deskripsi |
| --- | --- |
| `AuroraForwardingWriterDMLThroughput` | Hitungan (per detik). Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini. |
| `AuroraForwardingWriterOpenSessions` | Hitungan. Jumlah sesi terbuka pada instans DB penulis ini yang memproses kueri yang diteruskan. |
| `AuroraForwardingWriterTotalSessions` | Hitungan. Total jumlah sesi yang diteruskan pada instans DB penulis ini. |

CloudWatch Metrik berikut berlaku untuk setiap cluster sekunder. Metrik ini diukur pada setiap DB instance pembaca dalam cluster sekunder dengan penerusan tulis yang diaktfikan.

| CloudWatch Metrik | Unit dan deskripsi |
| --- | --- |
| `AuroraForwardingReplicaCommitThroughput` | Hitungan (per detik). Jumlah commit dalam sesi yang diteruskan oleh replika ini setiap detik. |
| `AuroraForwardingReplicaDMLLatency` | Milidetik. Waktu respons rata-rata dalam milidetik diteruskan DMLs pada replika. |
| `AuroraForwardingReplicaDMLThroughput` | Hitungan (per detik). Jumlah pernyataan DML yang diteruskan yang diproses pada replika ini setiap detiknya. |
| `AuroraForwardingReplicaErrorSessionsLimit` | Hitungan. Jumlah sesi yang ditolak oleh klaster primer karena batas untuk koneksi maksimum atau koneksi penerusan tulis maksimum tercapai. |

| CloudWatch Metrik | Unit dan deskripsi |
| --- | --- |
| `AuroraForwardingReplicaOpenSessions` | Hitungan. Jumlah sesi yang menggunakan penerusan tulis pada instans replika. |
| `AuroraForwardingReplicaReadWaitLatency` | Milidetik. Rata-rata waktu tunggu dalam milidetik yang diperlukan oleh replika untuk menunggu agar konsisten dengan LSN klaster primer. Batas tunggu instans DB pembaca ditentukan oleh pengaturan `apg_write_forward.consistency_mode`. Untuk informasi tentang pengaturan ini, lihat the section called “Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL”. |

## Peristiwa tunggu untuk penerusan tulis di Aurora PostgreSQL

Amazon Aurora menghasilkan peristiwa tunggu berikut ini saat Anda menggunakan penerusan tulis dengan Aurora PostgreSQL.

Topik

- IPC: AuroraWriteForwardConnect
- IPC: AuroraWriteForwardConsistencyPoint
- IPC: AuroraWriteForwardExecute
- IPC: AuroraWriteForwardGetGlobalConsistencyPoint
- IPC: AuroraWriteForwardXactAbort
- IPC: AuroraWriteForwardXactCommit
- IPC: AuroraWriteForwardXactStart

### IPC: AuroraWriteForwardConnect

Peristiwa `IPC:AuroraWriteForwardConnect` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu koneksi ke simpul penulis klaster DB primer dibuka.

Kemungkinan menyebabkan peningkatan waktu tunggu

Peristiwa ini meningkat seiring meningkatnya jumlah upaya koneksi dari simpul pembaca Wilayah sekunder ke simpul penulis klaster DB primer.

Tindakan

Kurangi jumlah koneksi serentak dari simpul sekunder ke simpul penulis Wilayah primer.

IPC: AuroraWriteForwardConsistencyPoint

Peristiwa `IPC:AuroraWriteForwardConsistencyPoint` menjelaskan durasi kueri dari simpul di klaster DB sekunder akan menunggu hasil operasi tulis yang diteruskan untuk direplikasi ke Wilayah saat ini. Peristiwa ini hanya dihasilkan jika parameter tingkat sesi `apg_write_forward.consistency_mode` diatur ke salah satu dari berikut ini:

- `SESSION` – kueri pada simpul sekunder menunggu hasil semua perubahan yang dibuat dalam sesi tersebut.
- `GLOBAL` – kueri pada simpul sekunder menunggu hasil perubahan yang dibuat oleh sesi tersebut, ditambah semua perubahan yang di-commit dari Wilayah primer dan Wilayah sekunder lainnya di klaster global.

Untuk informasi selengkapnya tentang pengaturan parameter `apg_write_forward.consistency_mode`, lihat the section called “Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL”.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum untuk waktu tunggu yang lebih lama termasuk yang berikut:

- Peningkatan lag replika, yang diukur dengan CloudWatch `ReplicaLag` metrik Amazon. Untuk informasi selengkapnya tentang metrik ini, lihat Memantau replikasi Aurora PostgreSQL.
- Peningkatan beban pada simpul penulis Wilayah primer atau pada simpul sekunder.

Tindakan

Ubah mode konsistensi Anda, tergantung kebutuhan aplikasi Anda.

### IPC: AuroraWriteForwardExecute

Peristiwa `IPC:AuroraWriteForwardExecute` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu kueri yang diteruskan selesai dan mendapatkan hasil dari simpul penulis klaster DB primer.

#### Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Mengambil banyak baris dari simpul penulis Wilayah primer.
- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

#### Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Optimalkan kueri untuk mengambil hanya data yang diperlukan.
- Optimalkan operasi bahasa manipulasi data (DML) untuk hanya memodifikasi data yang diperlukan.
- Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

### IPC: AuroraWriteForwardGetGlobalConsistencyPoint

Peristiwa `IPC:AuroraWriteForwardGetGlobalConsistencyPoint` terjadi ketika proses backend pada klaster DB sekunder yang menggunakan mode konsistensi GLOBAL sedang menunggu untuk mendapatkan titik konsistensi global dari simpul penulis sebelum menjalankan kueri.

#### Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Ubah mode konsistensi Anda, tergantung kebutuhan aplikasi Anda.
- Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardXactAbort

Peristiwa `IPC:AuroraWriteForwardXactAbort` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu hasil kueri pembersihan jarak jauh. Kueri pembersihan dikeluarkan untuk mengembalikan proses ke keadaan yang sesuai setelah transaksi dengan penerusan tulis dibatalkan. Amazon Aurora melakukannya karena ada kesalahan yang ditemukan atau karena pengguna mengeluarkan perintah `ABORT` eksplisit atau membatalkan kueri yang sedang berjalan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri pembersihan dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Selidiki penyebab transaksi yang dibatalkan.
- Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardXactCommit

Peristiwa `IPC:AuroraWriteForwardXactCommit` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu hasil perintah transaksi commit yang diteruskan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardXactStart

Peristiwa `IPC:AuroraWriteForwardXactStart` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu hasil perintah transaksi mulai yang diteruskan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.

- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

# Menggunakan switchover atau failover di Amazon Aurora Global Database

Fitur Aurora Global Database memberikan lebih banyak perlindungan kontinuitas bisnis dan pemulihan bencana (BCDR) daripada ketersediaan standar tinggi yang disediakan oleh cluster Aurora DB dalam satu. Wilayah AWS Dengan menggunakan Aurora Global Database, Anda dapat merencanakan pemulihan yang lebih cepat dari bencana Regional yang jarang dan tidak direncanakan atau menyelesaikan pemadaman tingkat layanan dengan cepat.

Anda dapat berkonsultasi dengan panduan dan prosedur berikut untuk merencanakan, menguji, dan menerapkan strategi BCDR Anda menggunakan fitur Aurora Global Database.

Topik

- Perencanaan untuk kelangsungan bisnis dan pemulihan bencana
- Melakukan switchover untuk basis data global Amazon Aurora
- Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan
- Mengelola RPOs database global berbasis Aurora PostgreSQL
- Ketahanan Lintas Wilayah untuk klaster sekunder Global Database

## Perencanaan untuk kelangsungan bisnis dan pemulihan bencana

Untuk merencanakan kelangsungan bisnis Anda dan strategi pemulihan bencana, sangat membantu untuk memahami terminologi industri berikut dan bagaimana istilah-istilah tersebut berhubungan dengan fitur Aurora Global Database.

Pemulihan dari bencana biasanya didorong oleh dua tujuan bisnis berikut:

- Sasaran waktu pemulihan (RTO)—Waktu yang diperlukan sistem untuk kembali ke status kerja aktif setelah terjadinya bencana atau pemadaman layanan. Dengan kata lain, RTO mengukur waktu henti operasional. Untuk Aurora Global Database, RTO bisa dalam urutan menit.
- Sasaran titik pemulihan (RPO)—Jumlah data yang dapat hilang (diukur berdasarkan waktu) setelah terjadinya bencana atau pemadaman layanan. Kehilangan data tersebut biasanya disebabkan oleh keterlambatan replikasi asinkron. Untuk basis data global Aurora, RPO biasanya diukur dalam hitungan detik. Dengan basis data global berbasis Aurora PostgreSQL, Anda dapat menggunakan parameter `rds.global_db_rpo` untuk mengatur dan melacak batas atas RPO, tetapi melakukannya dapat memengaruhi pemrosesan transaksi pada simpul penulis di klaster primer. Untuk informasi selengkapnya, lihat Mengelola RPOs database global berbasis Aurora PostgreSQL.

Melakukan switchover atau failover dengan Aurora Global Database melibatkan mempromosikan cluster DB sekunder menjadi cluster DB utama. Istilah "pemadaman regional" sering kali digunakan untuk mendeskripsikan berbagai skenario kegagalan. Skenario terburuk dapat berupa pemadaman luas karena peristiwa bencana yang memengaruhi wilayah seluas ratusan mil persegi. Namun, kebanyakan pemadaman sudah jauh lebih terlokalisasi, dengan hanya memengaruhi sebagian kecil subset layanan cloud atau sistem pelanggan. Pertimbangkan cakupan pemadaman secara menyeluruh untuk memastikan bahwa failover lintas Wilayah merupakan solusi yang tepat dan untuk memilih metode failover yang sesuai dengan situasi tersebut. Penggunaan pendekatan failover atau switchover bergantung pada skenario pemadaman tertentu:

- Failover—Gunakan pendekatan ini untuk melakukan pemulihan dari pemadaman yang tidak direncanakan. Dengan pendekatan ini, Anda melakukan failover lintas Wilayah ke salah satu klaster DB sekunder dalam basis data global Aurora Anda. RPO untuk pendekatan ini biasanya merupakan nilai bukan nol yang diukur dalam hitungan detik. Jumlah kehilangan data tergantung pada kelambatan replikasi database global Aurora Wilayah AWS pada saat kegagalan. Untuk mempelajari selengkapnya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.
- Switchover — Operasi ini sebelumnya disebut “failover terencana terkelola”. Gunakan pendekatan ini untuk skenario terkontrol, seperti pemeliharaan operasional dan prosedur operasional terencana lainnya di mana semua cluster Aurora dan layanan lain yang berinteraksi dengan mereka berada dalam keadaan sehat. Karena fitur ini menyinkronkan klaster DB sekunder dengan primer

sebelum membuat perubahan lain, RPO adalah 0 (tidak ada kehilangan data). Untuk mempelajari selengkapnya, lihat Melakukan switchover untuk basis data global Amazon Aurora.

> **Note**
>
> Sebelum Anda dapat melakukan switchover atau failover ke cluster Aurora DB sekunder tanpa kepala, Anda harus menambahkan instance DB ke dalamnya. Untuk informasi selengkapnya tentang klaster DB headless, lihat Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder.

## Melakukan switchover untuk basis data global Amazon Aurora

> **Note**
>
> Switchover sebelumnya disebut failover terencana terkelola.

Dengan menggunakan switchover, Anda dapat mengubah Wilayah cluster utama Anda secara rutin. Pendekatan ini ditujukan untuk skenario yang terkontrol, seperti pemeliharaan operasional dan prosedur operasional terencana lainnya.

Terdapat tiga kasus umum untuk penggunaan switchover.

- Untuk persyaratan "rotasi regional" yang diberlakukan pada industri tertentu. Misalnya, peraturan layanan keuangan mungkin menginginkan sistem tier-0 untuk beralih ke Wilayah yang berbeda selama beberapa bulan untuk memastikan prosedur pemulihan bencana dilaksanakan secara teratur.
- Untuk aplikasi Multi-region follow-the-sun "". Misalnya, suatu bisnis mungkin ingin menyediakan penulisan dengan latensi lebih rendah di berbagai Wilayah berdasarkan jam kerja di zona waktu yang berbeda.
- Sebagai zero-data-loss metode untuk gagal kembali ke Wilayah primer asli setelah failover.

> **Note**
>
> Switchovers dirancang untuk digunakan pada database global Aurora di mana semua cluster Aurora dan layanan lain yang berinteraksi dengan mereka berada dalam keadaan sehat.

Untuk pulih dari pemadaman yang tak terduga, ikuti prosedur yang sesuai di Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.
Anda dapat melakukan switchover lintas wilayah terkelola dengan Aurora Global Database hanya jika cluster DB primer dan sekunder memiliki versi mesin mayor dan minor yang sama. Tergantung pada versi mesin dan mesin, level patch mungkin harus identik atau level patch bisa berbeda. Untuk daftar mesin dan versi mesin yang memungkinkan operasi ini antara cluster primer dan sekunder dengan tingkat patch yang berbeda, lihatKompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola. Sebelum Anda memulai switchover, periksa versi mesin di klaster global untuk memastikan versi mesin tersebut mendukung switchover lintas Wilayah terkelola, lalu tingkatkan jika diperlukan.

Selama peralihan, Aurora membuat cluster di Wilayah sekunder pilihan Anda menjadi cluster utama Anda. Mekanisme peralihan mempertahankan topologi replikasi database global Anda yang ada: masih memiliki jumlah cluster Aurora yang sama di Wilayah yang sama. Sebelum Aurora memulai proses peralihan, ia menunggu cluster Region sekunder target disinkronkan sepenuhnya dengan cluster Region primer. Kemudian, cluster DB di Region primer menjadi hanya-baca. Cluster sekunder yang dipilih mempromosikan salah satu node hanya-baca ke status penulis penuh, yang memungkinkan cluster sekunder untuk mengambil peran cluster primer. Karena cluster sekunder target disinkronkan dengan primer pada awal proses, primer baru melanjutkan operasi untuk database global Aurora tanpa kehilangan data apa pun. Basis data Anda tidak tersedia untuk sementara selama klaster primer dan klaster sekunder yang dipilih mengambil peran barunya masing-masing.

Note

Untuk mengelola slot replikasi untuk Aurora PostgreSQL setelah melakukan peralihan, lihat. Mengelola slot logis untuk Aurora Postgre SQL

Untuk mengoptimalkan ketersediaan aplikasi, sebaiknya lakukan hal berikut sebelum menggunakan fitur ini:

- Lakukan operasi ini di luar jam sibuk atau pada waktu lainnya saat operasi penulisan ke klaster DB primer tidak banyak.
- Periksa waktu keterlambatan untuk semua klaster DB Aurora sekunder dalam basis data global Aurora. Untuk semua database global berbasis Aurora PostgreSQL dan untuk database global

berbasis Aurora MySQL yang dimulai dengan versi mesin 3.04.0 dan lebih tinggi atau 2.12.0 dan lebih tinggi, gunakan Amazon untuk melihat metrik untuk semua cluster DB sekunder. CloudWatch `AuroraGlobalDBRPOLag` Untuk basis data global berbasis Aurora MySQL versi minor yang lebih rendah, lihat metrik `AuroraGlobalDBReplicationLag`. Metrik ini menunjukkan seberapa terlambatnya (dalam milidetik) replikasi ke klaster sekunder dibandingkan ke klaster DB primer. Nilai ini berbanding lurus dengan waktu yang diperlukan Aurora untuk menyelesaikan switchover. Karena itu, semakin besar nilai keterlambatan, semakin lama durasi switchover. Saat Anda memeriksa metrik ini, lakukan dari cluster utama saat ini.

Untuk informasi selengkapnya tentang CloudWatch metrik untuk Aurora, lihat. Metrik tingkat klaster untuk Amazon Aurora

- Cluster DB sekunder yang dipromosikan selama peralihan mungkin memiliki pengaturan konfigurasi yang berbeda dari cluster DB primer lama. Kami menyarankan agar Anda menjaga jenis pengaturan konfigurasi berikut konsisten di semua cluster di cluster database global Aurora Anda. Melakukannya membantu meminimalkan masalah kinerja, ketidakcocokan beban kerja, dan perilaku anomali lainnya setelah peralihan.
  - Konfigurasikan grup parameter cluster Aurora DB untuk primer baru, jika perlu — Saat Anda mempromosikan cluster DB sekunder untuk mengambil alih peran utama, grup parameter dari sekunder mungkin dikonfigurasi secara berbeda dari yang primer. Jika demikian, ubah grup parameter klaster DB sekunder yang dipromosikan agar sesuai dengan pengaturan klaster primer Anda. Untuk mempelajari caranya, lihat Memodifikasi parameter basis data global Aurora.
  - Konfigurasikan alat dan opsi pemantauan, seperti Amazon CloudWatch Events dan alarm — Konfigurasikan cluster DB yang dipromosikan dengan kemampuan logging, alarm, dan sebagainya yang sama sesuai kebutuhan untuk database global. Seperti grup parameter, konfigurasi untuk fitur ini tidak diwariskan dari klaster primer selama proses switchover berlangsung. Beberapa CloudWatch metrik, seperti replikasi lag, hanya tersedia untuk Wilayah sekunder. Karena itu, switchover akan mengubah cara Anda melihat metrik tersebut dan mengatur alarmnya, serta mengharuskan adanya perubahan pada dasbor yang ditentukan sebelumnya. Untuk informasi lebih lanjut tentang cluster dan pemantauan Aurora DB, lihat. Memantau Amazon Aurora dengan Amazon CloudWatch
  - Konfigurasikan integrasi dengan AWS layanan lain — Jika database global Aurora Anda terintegrasi AWS dengan layanan, AWS Secrets Manager seperti, AWS Identity and Access Management Amazon S3, AWS Lambda dan, pastikan untuk mengonfigurasi integrasi Anda dengan layanan ini sesuai kebutuhan. Untuk informasi selengkapnya tentang cara mengintegrasi basis data global Aurora dengan IAM, Amazon S3, dan Lambda, lihat Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya. Untuk mempelajari Secrets Manager

selengkapnya, lihat Cara mengotomatiskan replikasi rahasia secara AWS Secrets Manager keseluruhan. Wilayah AWS

Jika Anda menggunakan titik akhir penulis Aurora Global Database, Anda tidak perlu mengubah pengaturan koneksi di aplikasi Anda. Verifikasi bahwa perubahan DNS telah menyebar dan Anda dapat menghubungkan dan melakukan operasi tulis pada cluster utama yang baru. Kemudian Anda dapat melanjutkan operasi penuh aplikasi Anda.

Misalkan koneksi aplikasi Anda menggunakan titik akhir cluster dari cluster primer lama, bukan titik akhir penulis global. Dalam hal ini, pastikan untuk mengubah setelan koneksi aplikasi Anda untuk menggunakan titik akhir klaster klaster utama yang baru. Jika Anda menyetujui nama yang disediakan saat membuat basis data global Aurora, Anda dapat mengubah titik akhir dengan menghapus `-ro` dari string titik akhir klaster yang dipromosikan dalam aplikasi Anda. Sebagai contoh, titik akhir klaster sekunder `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` menjadi `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` ketika klaster tersebut dipromosikan menjadi klaster primer.

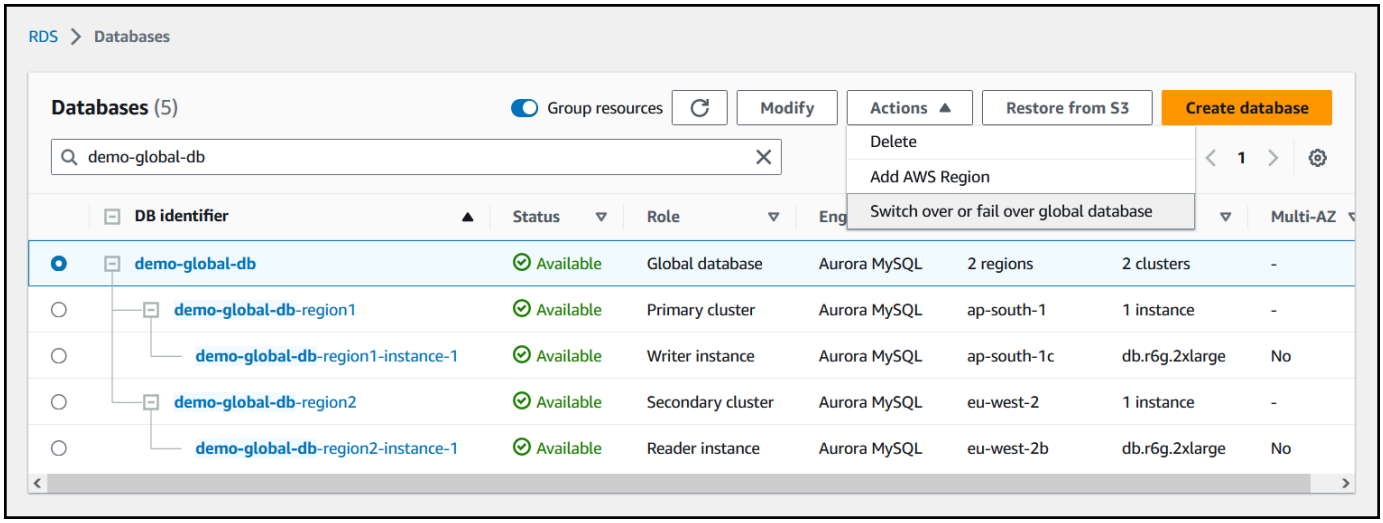
Jika Anda menggunakan RDS Proxy, pastikan untuk mengalihkan operasi penulisan aplikasi Anda ke titik akhir yang sesuairead/write endpoint of the proxy that's associated with the new primary cluster. This proxy endpoint might be the default endpoint or a custom read/write. Untuk mengetahui informasi selengkapnya, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

Anda dapat melakukan peralihan Aurora Global Database menggunakan, API AWS Management Console AWS CLI, atau RDS.

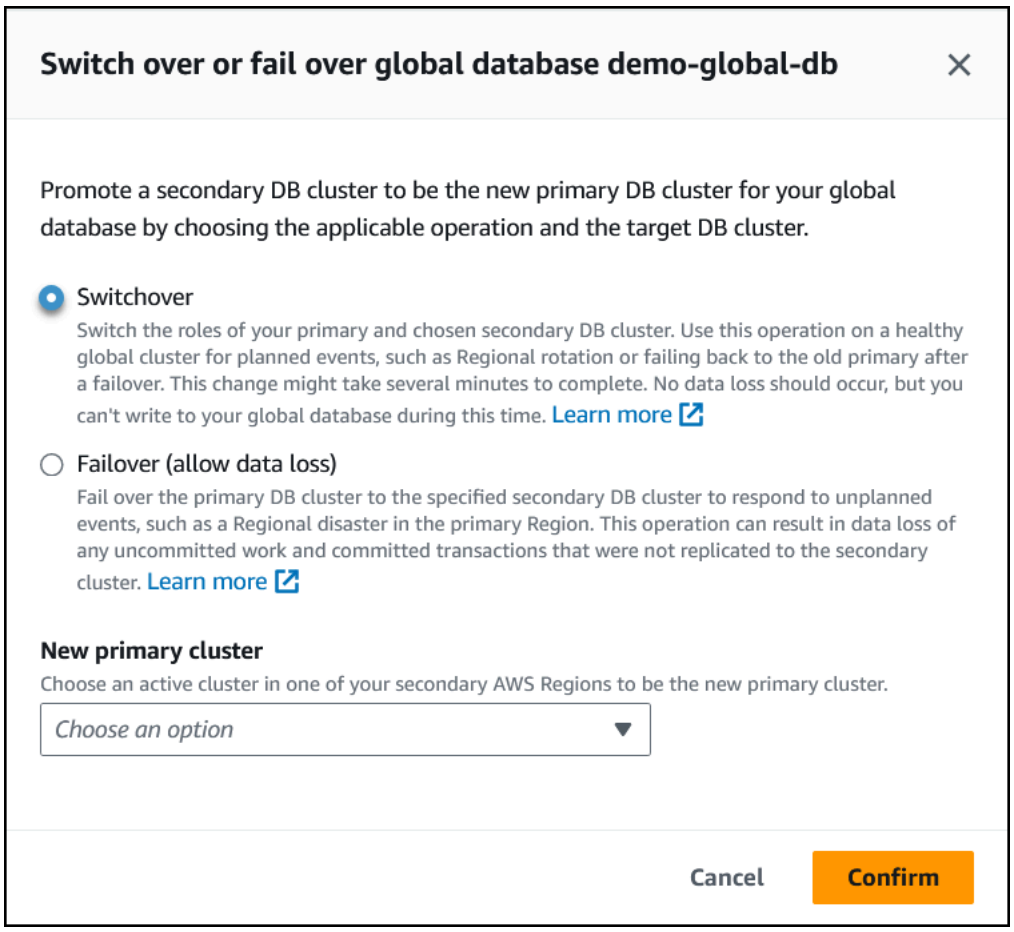
Konsol

Cara melakukan switchover pada basis data global Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Database dan temukan database global Aurora tempat Anda ingin melakukan peralihan.
3. Pilih Alihkan atau lakukan failover basis data global dari menu Tindakan.

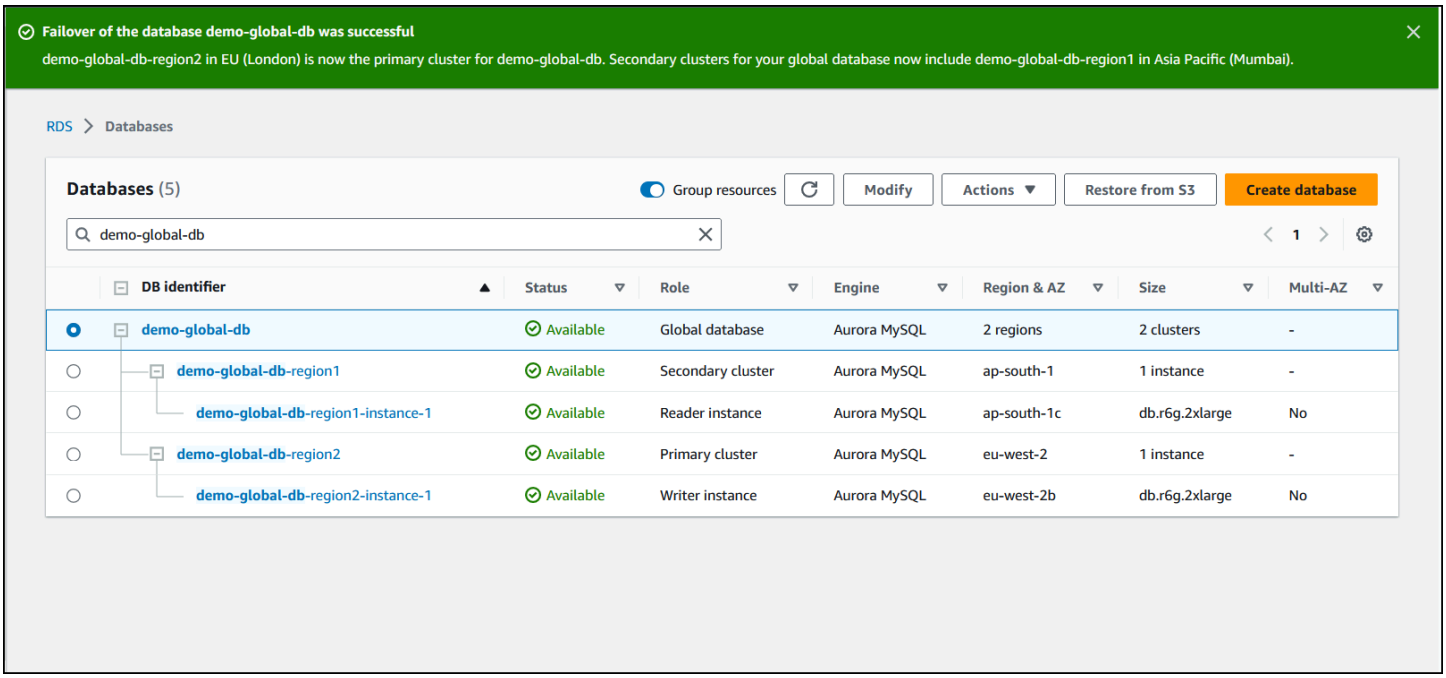



4. Pilih Switchover.




5. Untuk Klaster primer baru, pilih klaster aktif di salah satu Wilayah AWS sekunder untuk dijadikan klaster primer baru.
6. Pilih Konfirmasi.

Setelah switchover selesai, Anda dapat melihat klaster DB Aurora dan perannya saat ini di daftar Basis Data, sebagaimana ditunjukkan pada gambar berikut.




AWS CLI

Untuk melakukan switchover pada basis data global Aurora:

Gunakan perintah `switchover-global-cluster` CLI untuk melakukan peralihan untuk Aurora Global Database. Dengan perintah tersebut, loloskan nilai untuk parameter berikut.

- `--region`— Tentukan Wilayah AWS di mana cluster DB utama dari database global Aurora berjalan.
- `--global-cluster-identifier`—Tentukan nama basis data global Aurora.
- `--target-db-cluster-identifier`—Tentukan Amazon Resource Name (ARN) untuk klaster DB Aurora yang ingin Anda promosikan menjadi klaster primer basis data global Aurora.

Untuk Linux, macOS, atau Unix:

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

Untuk Windows:

```
aws rds --region region_of_primary ^
   switchover-global-cluster --global-cluster-identifier global_database_id ^
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

API RDS

Untuk melakukan peralihan untuk Aurora Global Database, jalankan operasi API. SwitchoverGlobalCluster

## Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan

Pada kesempatan langka, database global Aurora Anda mungkin mengalami pemadaman tak terduga pada awalnya. Wilayah AWS Jika hal ini terjadi, klaster DB Aurora primer dan simpul penulisnya tidak akan tersedia, dan replikasi antara klaster DB primer dan sekunder akan terhenti. Untuk meminimalkan waktu henti (RTO) dan kehilangan data (RPO), Anda dapat segera melakukan failover lintas Wilayah.

Aurora Global Database memiliki dua metode failover yang dapat Anda gunakan dalam situasi pemulihan bencana:

- Failover terkelola—Metode ini direkomendasikan untuk pemulihan bencana. Jika Anda menggunakan metode ini, Aurora akan secara otomatis menambahkan kembali Wilayah primer yang lama ke basis data global sebagai Wilayah sekunder saat sudah tersedia lagi. Dengan begitu, topologi asli klaster global akan dipertahankan. Untuk mempelajari cara menggunakan metode ini, lihat Melakukan failover terkelola untuk basis data global Aurora.
- Failover manual—Metode alternatif ini dapat digunakan ketika failover terkelola bukan merupakan pilihan, misalnya, ketika Wilayah primer dan sekunder Anda menjalankan versi mesin yang tidak kompatibel. Untuk mempelajari cara menggunakan metode ini, lihat Melakukan failover manual untuk basis data global Aurora.

> **Important**
>
> Kedua metode failover tersebut dapat mengakibatkan hilangnya data transaksi tulis yang tidak direplikasi ke tujuan sekunder yang dipilih sebelum peristiwa failover terjadi. Namun, proses pemulihan yang mempromosikan instans DB pada klaster DB sekunder pilihan

menjadi instans DB penulis primer akan memastikan bahwa data berada dalam kondisi yang konsisten secara transaksional. Kegagalan juga rentan terhadap masalah otak terbelah.

## Melakukan failover terkelola untuk basis data global Aurora

Pendekatan ini ditujukan untuk kelangsungan bisnis saat terjadi bencana alam Regional yang riil atau pemadaman tingkat layanan secara menyeluruh.

Selama failover terkelola, klaster sekunder di Region sekunder yang Anda pilih menjadi cluster primer baru. Klaster sekunder yang dipilih mempromosikan salah satu simpul hanya-bacanya ke status penulis penuh. Langkah ini memungkinkan klaster untuk mengambil peran sebagai klaster primer. Basis data Anda tidak akan tersedia untuk sementara saat klaster ini mengambil peran barunya. Segera setelah Wilayah primer lama itu sehat dan tersedia kembali, Aurora secara otomatis menambahkannya kembali ke cluster global sebagai Wilayah sekunder. Dengan demikian, topologi replikasi database global Aurora Anda yang ada dipertahankan.

> **Note**
>
> Untuk mengelola slot replikasi untuk Aurora PostgreSQL setelah melakukan failover, lihat. Mengelola slot logis untuk Aurora Postgre SQL

> **Note**
>
> Anda dapat melakukan failover lintas wilayah terkelola dengan Aurora Global Database hanya jika cluster DB primer dan sekunder memiliki versi mesin mayor dan minor yang sama. Tergantung pada versi mesin dan mesin, level patch mungkin harus identik atau level patch bisa berbeda. Untuk daftar mesin dan versi mesin yang memungkinkan operasi ini antara cluster primer dan sekunder dengan tingkat patch yang berbeda, lihatKompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola. Sebelum Anda memulai failover, periksa versi mesin di cluster global Anda untuk memastikan bahwa mereka mendukung peralihan lintas wilayah terkelola, dan tingkatkan jika diperlukan. Jika versi mesin Anda memerlukan level patch yang identik tetapi menjalankan level patch yang berbeda, Anda dapat melakukan failover secara manual dengan mengikuti langkah-langkah masukMelakukan failover manual untuk basis data global Aurora.

Failover terkelola tidak menunggu data disinkronkan antara Wilayah sekunder yang dipilih dan Wilayah primer saat ini. Karena Aurora Global Database mereplikasi data secara asinkron, ada kemungkinan bahwa tidak semua transaksi direplikasi ke AWS Wilayah sekunder yang dipilih sebelum dipromosikan untuk menerima kemampuan baca/tulis penuh.

Untuk memastikan data dalam keadaan konsisten, Aurora membuat volume penyimpanan baru untuk Wilayah primer lama setelah pulih. Sebelum membuat volume penyimpanan baru di AWS Wilayah, Aurora mencoba mengambil snapshot dari volume penyimpanan lama pada titik kegagalan. Dengan begitu, Anda dapat memulihkan snapshot dan memulihkan data yang hilang darinya. Jika operasi ini berhasil, Aurora menempatkan snapshot ini yang dinamai `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`* di bagian snapshot. AWS Management Console Anda juga dapat menggunakan `describe-db-cluster-snapshots` AWS CLI perintah atau operasi `DescribeDBClusterSnapshots` API untuk melihat detail snapshot.

Saat Anda memulai failover terkelola, Aurora juga mencoba menghentikan lalu lintas tulis melalui lapisan penyimpanan Aurora yang sangat tersedia. Kami menyebut mekanisme ini sebagai “tulis pagar”. Jika proses berhasil, Aurora memancarkan Acara RDS yang memberi tahu Anda bahwa penulisan dihentikan. Jika terjadi beberapa kegagalan AZ di suatu Wilayah, ada kemungkinan bahwa proses pagar tulis tidak berhasil tepat waktu. Dalam hal ini, Aurora memancarkan acara RDS yang memberi tahu Anda bahwa proses untuk menghentikan penulisan habis waktunya. Jika cluster primer lama dapat dijangkau di jaringan, Aurora mencatat peristiwa ini di sana. Jika tidak, Aurora merekam peristiwa di cluster primer baru. Untuk mempelajari lebih lanjut tentang peristiwa ini, lihatPeristiwa klaster DB. Karena penulisan pagar adalah upaya terbaik, ada kemungkinan bahwa tulisan dapat diterima sesaat di Wilayah primer lama, menyebabkan masalah otak terbelah.

Kami menyarankan Anda menyelesaikan tugas-tugas berikut sebelum Anda melakukan failover dengan Aurora Global Database. Melakukannya meminimalkan kemungkinan masalah otak terbelah, atau memulihkan data yang tidak direplikasi dari snapshot cluster primer lama.

- Untuk mencegah penulisan dikirim ke cluster utama Aurora Global Database, gunakan aplikasi secara offline.
- Pastikan bahwa aplikasi apa pun yang terhubung ke cluster DB primer menggunakan titik akhir penulis global. Titik akhir ini memiliki nilai yang tetap sama bahkan ketika Wilayah baru menjadi cluster utama karena peralihan atau failover. Aurora menerapkan perlindungan tambahan untuk meminimalkan kemungkinan kehilangan data untuk operasi tulis yang diajukan melalui titik akhir global. Untuk informasi lebih lanjut tentang titik akhir penulis global, lihatMenghubungkan ke Amazon Aurora Global Database.

- Jika Anda menggunakan titik akhir penulis global dan nilai DNS cache aplikasi atau lapisan jaringan Anda, kurangi time-to-live (TTL) cache DNS Anda ke nilai rendah seperti 5 detik. Dengan begitu, aplikasi Anda dengan cepat mendaftarkan perubahan DNS dengan titik akhir penulis global. Meskipun Aurora mencoba untuk memblokir tulisan di Wilayah primer lama, tindakan tersebut tidak dijamin akan berhasil. Mengurangi durasi cache DNS semakin mengurangi kemungkinan masalah otak terbelah. Sebagai alternatif, Anda dapat memeriksa peristiwa RDS yang memberi tahu Anda ketika Aurora mengamati perubahan DNS untuk titik akhir penulis global. Dengan begitu, Anda dapat memvalidasi bahwa aplikasi Anda juga mendaftarkan perubahan DNS sebelum memulai ulang lalu lintas tulis aplikasi Anda.
- Periksa waktu jeda untuk semua cluster Aurora DB sekunder di Aurora Global Database. Memilih Wilayah sekunder dengan keterlambatan replikasi minimum dapat meminimalkan kehilangan data dari Wilayah primer yang mengalami kegagalan.

  Untuk semua versi database global berbasis Aurora PostgreSQL, dan untuk database global berbasis Aurora MySQL yang dimulai dengan versi mesin 3.04.0 dan lebih tinggi atau 2.12.0 dan lebih tinggi, gunakan Amazon untuk melihat metrik untuk semua cluster DB sekunder. CloudWatch `AuroraGlobalDBRPOLag` Untuk basis data global berbasis Aurora MySQL versi minor yang lebih rendah, lihat metrik `AuroraGlobalDBReplicationLag`. Metrik ini menunjukkan seberapa terlambatnya (dalam milidetik) replikasi ke klaster sekunder dibandingkan ke klaster DB primer.

  Untuk informasi selengkapnya tentang CloudWatch metrik untuk Aurora, lihat. Metrik tingkat klaster untuk Amazon Aurora

Selama failover terkelola, klaster DB sekunder yang dipilih dipromosikan ke peran barunya sebagai klaster primer. Namun, klaster ini tidak mewarisi berbagai opsi konfigurasi klaster DB primer. Ketidakcocokan dalam konfigurasi dapat menyebabkan masalah performa, inkompatibilitas beban kerja, dan perilaku anomali lainnya. Untuk mencegah masalah tersebut, sebaiknya atasi perbedaan di antara klaster basis data global Aurora untuk konfigurasi berikut:

- Konfigurasikan grup parameter cluster Aurora DB untuk primer baru, jika perlu — Anda dapat mengonfigurasi grup parameter cluster Aurora DB Anda secara independen untuk setiap cluster Aurora di Database Global Aurora Anda. Karena itu, ketika Anda mempromosikan klaster DB sekunder untuk mengambil alih peran sebagai klaster primer, grup parameter dari klaster sekunder mungkin memiliki konfigurasi yang berbeda dengan klaster primer. Jika demikian, ubah grup parameter klaster DB sekunder yang dipromosikan agar sesuai dengan pengaturan klaster primer Anda. Untuk mempelajari caranya, lihat Memodifikasi parameter basis data global Aurora.

- Konfigurasikan alat dan opsi pemantauan, seperti Amazon CloudWatch Events dan alarm — Konfigurasikan cluster DB yang dipromosikan dengan kemampuan logging, alarm, dan sebagainya yang sama sesuai kebutuhan untuk database global. Seperti grup parameter, konfigurasi untuk fitur ini tidak diwariskan dari klaster primer selama proses failover berlangsung. Beberapa CloudWatch metrik, seperti replikasi lag, hanya tersedia untuk Wilayah sekunder. Karena itu, failover akan mengubah cara Anda melihat metrik tersebut dan mengatur alarmnya, serta mengharuskan adanya perubahan pada dasbor yang ditentukan sebelumnya. Untuk informasi selengkapnya tentang pemantauan klaster Aurora DB, lihat. Memantau Amazon Aurora dengan Amazon CloudWatch
- Konfigurasikan integrasi dengan AWS layanan lain — Jika Database Global Aurora Anda terintegrasi dengan layanan AWS lain, AWS Secrets Manager seperti, AWS Identity and Access Management Amazon S3, AWS Lambda dan, Anda perlu memastikan bahwa ini dikonfigurasi sesuai kebutuhan untuk akses dari Wilayah sekunder mana pun. Untuk informasi selengkapnya tentang cara mengintegrasi basis data global Aurora dengan IAM, Amazon S3, dan Lambda, lihat Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya. Untuk mempelajari Secrets Manager selengkapnya, lihat Cara mengotomatiskan replikasi rahasia secara AWS Secrets Manager keseluruhan. Wilayah AWS

Biasanya, klaster sekunder yang dipilih akan mengasumsikan peran primer dalam beberapa menit. Segera setelah instans DB penulis Region utama baru tersedia, Anda dapat menghubungkan aplikasi Anda ke sana dan melanjutkan beban kerja Anda. Setelah mempromosikan klaster primer yang baru, Aurora secara otomatis membangun ulang semua klaster Wilayah sekunder tambahan.

Karena basis data global Aurora menggunakan replikasi asinkron, keterlambatan replikasi di masing-masing Wilayah sekunder dapat bervariasi. Aurora membangun kembali Wilayah sekunder ini untuk memiliki point-in-time data yang sama persis dengan cluster Region primer yang baru. Durasi penyelesaian tugas pembangunan ulang dapat memerlukan waktu beberapa menit hingga beberapa jam, bergantung pada ukuran volume penyimpanan dan jarak di antara Wilayah. Saat klaster Wilayah sekunder selesai dibuat ulang dari Wilayah primer yang baru, klaster ini menjadi tersedia untuk akses baca.

Segera setelah penulis primer baru dipromosikan dan tersedia, klaster Wilayah primer yang baru dapat menangani operasi baca dan tulis untuk basis data global Aurora.

Jika Anda menggunakan titik akhir global, Anda tidak perlu mengubah pengaturan koneksi di aplikasi Anda. Verifikasi bahwa perubahan DNS telah menyebar dan Anda dapat menghubungkan dan

melakukan operasi tulis pada cluster utama yang baru. Kemudian Anda dapat melanjutkan operasi penuh aplikasi Anda.

Jika Anda tidak menggunakan titik akhir global, pastikan untuk mengubah titik akhir aplikasi Anda untuk menggunakan titik akhir cluster untuk klaster DB primer yang baru dipromosikan. Jika Anda menyetujui nama yang disediakan saat membuat basis data global Aurora, Anda dapat mengubah titik akhir dengan menghapus `-ro` dari string titik akhir klaster yang dipromosikan dalam aplikasi Anda.

Sebagai contoh, titik akhir klaster sekunder `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` menjadi `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` ketika klaster tersebut dipromosikan menjadi klaster primer.

Jika Anda menggunakan RDS Proxy, pastikan untuk mengarahkan operasi penulisan aplikasi Anda ke titik akhir yang sesuairead/write endpoint of the proxy that's associated with the new primary cluster. This proxy endpoint might be the default endpoint or a custom read/write. Untuk mengetahui informasi selengkapnya, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

Untuk memulihkan topologi asli klaster basis data global, Aurora memantau ketersediaan Wilayah primer yang lama. Segera setelah Wilayah tersebut normal dan tersedia kembali, Aurora secara otomatis menambahkannya kembali ke klaster global sebagai Wilayah sekunder. Sebelum membuat volume penyimpanan baru di Wilayah primer yang lama, Aurora mencoba mengambil snapshot dari volume penyimpanan lama pada titik kegagalan. Hal ini dilakukan agar Anda dapat menggunakannya untuk memulihkan setiap data yang hilang. Jika operasi ini berhasil, Aurora membuat snapshot bernama. `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`* Anda dapat menemukan snapshot ini di bagian Snapshots di. AWS Management Console Anda juga dapat melihat snapshot ini tercantum dalam informasi yang ditampilkan oleh operasi Deskripsikan DBCluster Snapshots API.

> **Note**
>
> Snapshot dari volume penyimpanan lama adalah snapshot sistem yang tunduk pada periode retensi pencadangan yang dikonfigurasi pada klaster primer yang lama. Untuk mempertahankan snapshot ini di luar periode retensi, Anda dapat menyalin snapshot untuk disimpan sebagai snapshot manual. Untuk mempelajari selengkapnya tentang cara menyalin snapshot, termasuk harga, lihat Penyalinan snapshot cluster DB.

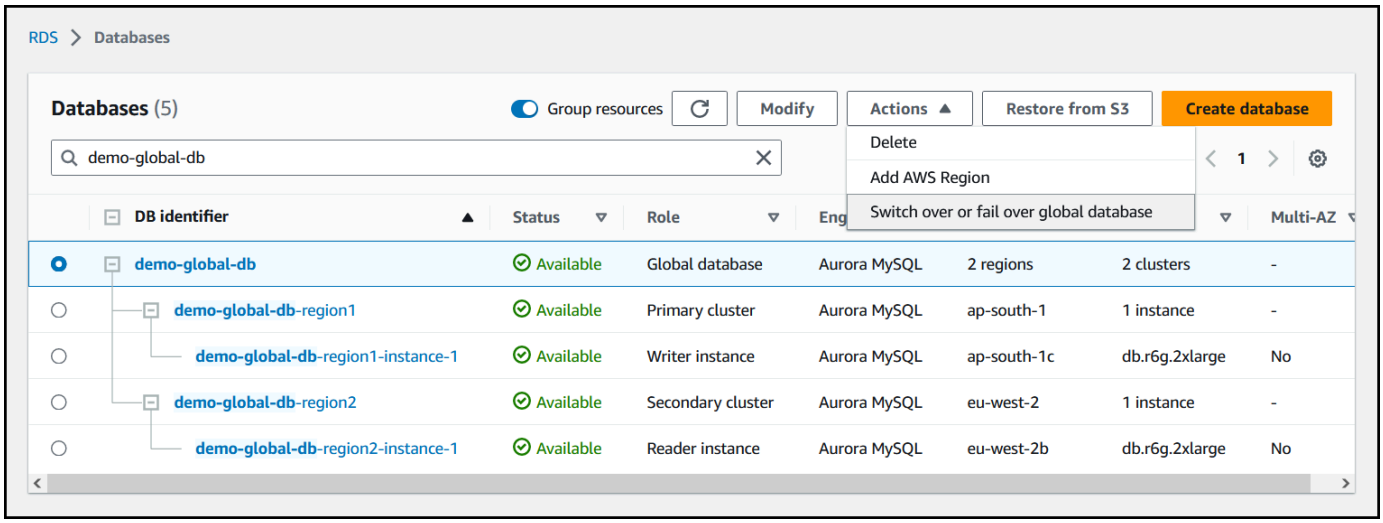
Setelah topologi asli dipulihkan, Anda dapat mengembalikan basis data global ke Wilayah primer asli dengan melakukan operasi switchover jika dianggap sebagai tindakan terbaik untuk bisnis dan beban kerja Anda. Untuk melakukannya, ikuti langkah-langkah yang ada di Melakukan switchover untuk basis data global Amazon Aurora.

Anda dapat melakukan failover dengan Aurora Global Database menggunakan, AWS Management Console API, AWS CLI atau RDS.

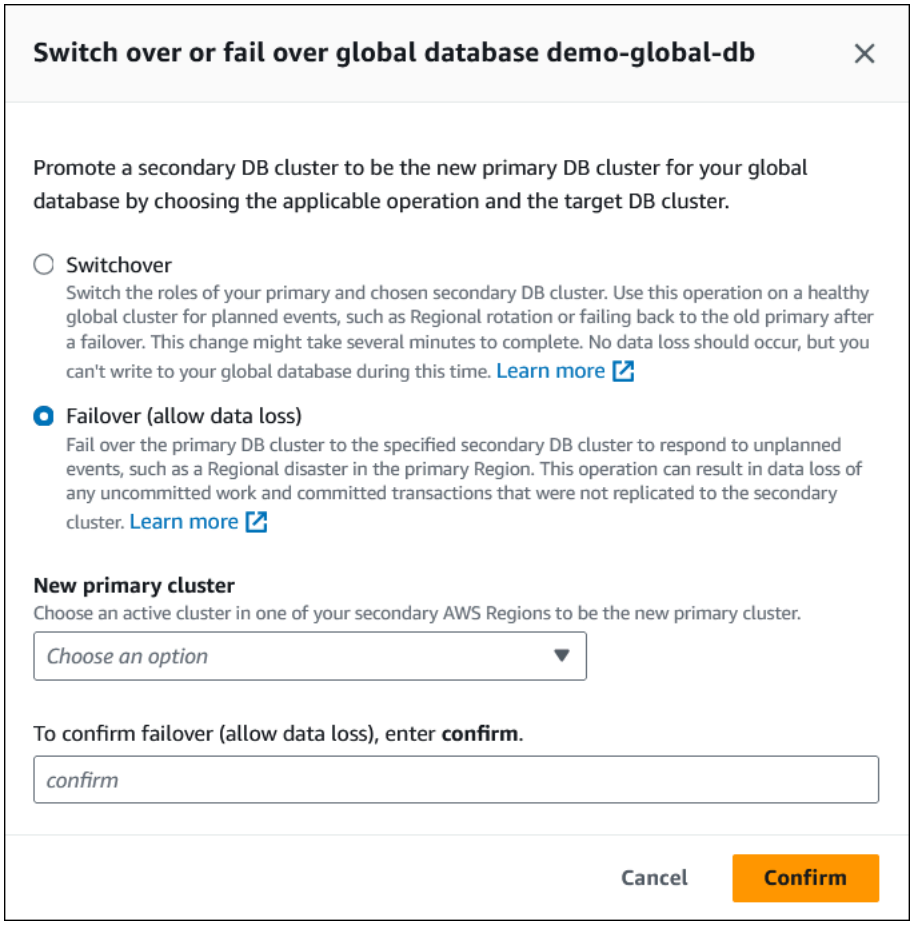
Konsol

Untuk melakukan failover terkelola pada basis data global Aurora:

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Database dan temukan database global Aurora tempat Anda ingin melakukan failover.
3. Pilih Alihkan atau lakukan failover basis data global dari menu Tindakan.

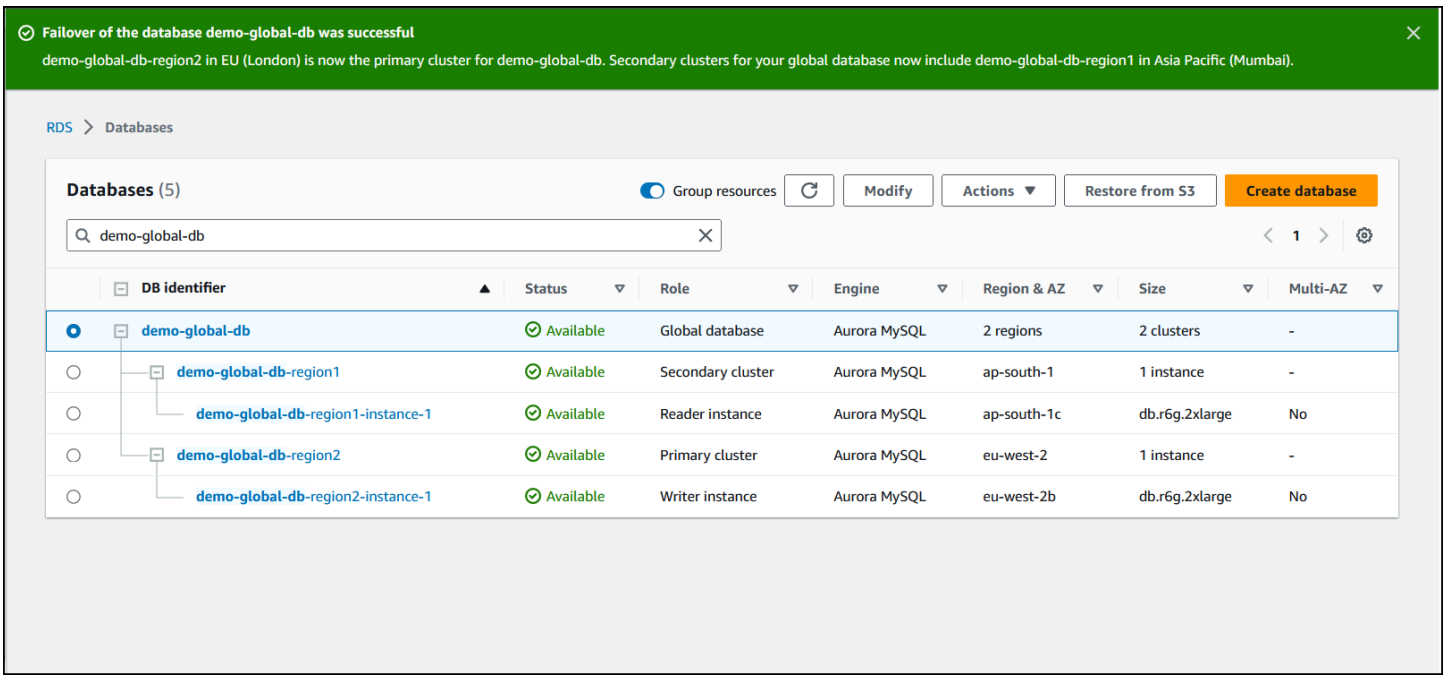



4. Pilih Failover (memungkinkan kehilangan data).




5. Untuk Klaster primer baru, pilih klaster aktif di salah satu Wilayah AWS sekunder untuk dijadikan klaster primer baru.
6. Masukkan **`confirm`**, lalu pilih Konfirmasi.

Ketika failover selesai, Anda dapat melihat cluster Aurora DB dan statusnya saat ini dalam daftar Database, seperti yang ditunjukkan pada gambar berikut.




AWS CLI

Untuk melakukan failover terkelola pada basis data global Aurora:

Gunakan perintah `failover-global-cluster` CLI untuk melakukan failover dengan Aurora Global Database. Dengan perintah tersebut, loloskan nilai untuk parameter berikut.

- `--region`— Tentukan Wilayah AWS di mana cluster DB sekunder yang Anda inginkan menjadi primer baru untuk database global Aurora berjalan.
- `--global-cluster-identifier`—Tentukan nama basis data global Aurora.
- `--target-db-cluster-identifier`—Tentukan Amazon Resource Name (ARN) untuk klaster DB Aurora yang ingin Anda promosikan menjadi klaster primer baru untuk basis data global Aurora.
- `--allow-data-loss`—Secara eksplisit jadikan tindakan ini sebagai operasi failover, bukan operasi switchover. Operasi failover dapat membuat Anda kehilangan sejumlah data jika komponen replikasi asinkron belum selesai mengirim semua data yang direplikasi ke Wilayah sekunder.

Untuk Linux, macOS, atau Unix:

```
aws rds --region region_of_selected_secondary \
   failover-global-cluster --global-cluster-identifier global_database_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

Untuk Windows:

```
aws rds --region region_of_selected_secondary ^
   failover-global-cluster --global-cluster-identifier global_database_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

API RDS

Untuk melakukan failover dengan Aurora Global Database, jalankan operasi API FailoverGlobalCluster.

## Melakukan failover manual untuk basis data global Aurora

Dalam skenario tertentu, Anda mungkin tidak dapat menggunakan proses failover terkelola. Salah satu contohnya adalah jika klaster DB primer dan sekunder tidak menjalankan versi mesin yang kompatibel. Dalam hal ini, Anda dapat mengikuti proses manual ini untuk melakukan failover ke Wilayah sekunder target Anda.

> **Tip**
>
> Sebaiknya Anda memahami proses ini sebelum menggunakannya. Siapkan rencana untuk segera memulai saat pertama kali muncul tanda masalah tingkat Wilayah. Anda dapat siap mengidentifikasi Wilayah sekunder dengan kelambatan replikasi paling sedikit dengan menggunakan Amazon CloudWatch secara teratur untuk melacak waktu jeda untuk klaster sekunder. Pastikan Anda menguji rencana untuk memastikan prosedur sudah lengkap dan akurat, dan bahwa staf Anda terlatih untuk melakukan failover pemulihan bencana sebelum bencana benar-benar terjadi.

Untuk melakukan failover manual ke cluster sekunder setelah pemadaman yang tidak direncanakan di Wilayah primer

1. Berhenti mengeluarkan pernyataan DML dan operasi tulis lainnya ke klaster DB Aurora primer di Wilayah AWS saat terjadi pemadaman.
2. Identifikasi cluster Aurora DB dari sekunder Wilayah AWS untuk digunakan sebagai cluster DB primer baru. Jika Anda memiliki dua atau lebih sekunder Wilayah AWS dalam database global Aurora Anda, pilih cluster sekunder yang memiliki kelambatan replikasi paling sedikit.

3. Lepaskan klaster DB sekunder pilihan Anda dari basis data global Aurora.

   Pelepasan klaster DB sekunder dari basis data global Aurora akan segera menghentikan replikasi dari klaster primer ke klaster sekunder tersebut dan mempromosikannya menjadi klaster DB Aurora yang berdiri sendiri dengan kapabilitas baca/tulis penuh. Klaster DB Aurora sekunder lainnya yang terkait dengan klaster primer di Wilayah yang mengalami pemadaman akan tetap tersedia dan dapat menerima panggilan dari aplikasi Anda. Klaster tersebut juga mengonsumsi sumber daya. Karena Anda membuat ulang basis data global Aurora, hapus klaster DB sekunder lainnya sebelum membuat basis data global Aurora yang baru dengan mengikuti langkah-langkah berikut. Tindakan ini akan mencegah inkonsistensi data di antara klaster DB dalam basis data global Aurora (masalah split-brain).

   Untuk langkah-langkah pelepasan terperinci, lihat Menghapus klaster dari basis data global Amazon Aurora.

4. Konfigurasi ulang aplikasi Anda untuk mengirim semua operasi tulis ke klaster DB Aurora yang kini berdiri sendiri ini menggunakan titik akhir barunya. Jika Anda menyetujui nama yang disediakan ketika Anda membuat basis data global Aurora, Anda dapat mengubah titik akhir dengan menghapus `-ro` dari string titik akhir klaster dalam aplikasi Anda.

   Sebagai contoh, titik akhir klaster sekunder `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` menjadi `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` saat klaster tersebut dilepas dari basis data global Aurora.

   Klaster DB Aurora ini menjadi klaster primer untuk basis data global Aurora yang baru ketika Anda mulai menambahkan Wilayah ke dalamnya pada langkah berikutnya.

   Jika Anda menggunakan RDS Proxy, pastikan untuk mengarahkan operasi penulisan aplikasi Anda ke titik akhir yang sesuairead/write endpoint of the proxy that's associated with the new primary cluster. This proxy endpoint might be the default endpoint or a custom read/write. Untuk mengetahui informasi selengkapnya, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

5. Tambahkan Wilayah AWS ke cluster DB. Saat Anda melakukannya, proses replikasi dari klaster primer ke klaster sekunder akan dimulai. Untuk langkah menambahkan Wilayah secara mendetail, lihat Menambahkan Wilayah AWS ke database global Amazon Aurora.

6. Tambahkan lebih banyak Wilayah AWS sesuai kebutuhan untuk membuat ulang topologi yang diperlukan untuk mendukung aplikasi Anda.

Pastikan penulisan aplikasi dikirim ke klaster DB Aurora yang benar, baik sebelum, selama, maupun sesudah membuat perubahan ini. Tindakan ini akan mencegah inkonsistensi data di antara klaster DB dalam basis data global Aurora (masalah split-brain).

Jika Anda mengonfigurasi ulang sebagai respons terhadap pemadaman di sebuah Wilayah AWS, Anda dapat menjadikannya primer lagi setelah pemadaman diselesaikan. Wilayah AWS Untuk melakukannya, Anda menambahkan yang lama Wilayah AWS ke database global baru Anda, dan kemudian menggunakan proses peralihan untuk beralih perannya. Basis data global Aurora harus menggunakan versi Aurora PostgreSQL atau Aurora MySQL yang mendukung switchover. Untuk informasi selengkapnya, lihat Melakukan switchover untuk basis data global Amazon Aurora.

## Mengelola RPOs database global berbasis Aurora PostgreSQL

Dengan basis data global berbasis Aurora PostgreSQL, Anda dapat mengelola sasaran titik pemulihan (RPO) untuk basis data global Aurora menggunakan parameter `rds.global_db_rpo`. RPO mewakili jumlah maksimum data yang dapat hilang ketika terjadi pemadaman.

Ketika Anda mengatur RPO untuk basis data global berbasis Aurora PostgreSQL, Aurora memantau Waktu keterlambatan RPO dari semua klaster sekunder untuk memastikan bahwa setidaknya satu klaster sekunder tetap dalam jendela target RPO. Waktu keterlambatan RPO adalah salah satu metrik berbasis waktu.

RPO digunakan ketika database Anda melanjutkan operasi di baru Wilayah AWS setelah failover. Aurora mengevaluasi RPO dan waktu keterlambatan RPO untuk melakukan commit (atau memblokir) transaksi pada klaster primer sebagai berikut:

- Menjalankan commit transaksi jika setidaknya satu klaster DB sekunder memiliki waktu keterlambatan RPO lebih rendah dari RPO.
- Memblokir transaksi jika semua klaster DB sekunder memiliki waktu keterlambatan RPO yang lebih tinggi dari RPO. Aurora juga mencatat peristiwa ke file log PostgreSQL dan mengeluarkan peristiwa "tunggu" yang menunjukkan sesi terblokir.

Dengan kata lain, jika semua klaster sekunder berada di belakang RPO target, Aurora akan menjeda transaksi pada klaster primer hingga setidaknya satu klaster sekunder mengimbangi. Transaksi yang dijeda akan dilanjutkan dan di-commit segera setelah waktu keterlambatan dari setidaknya satu klaster DB sekunder menjadi lebih rendah dari RPO. Hasilnya adalah bahwa tidak ada transaksi yang dapat di-commit hingga RPO terpenuhi.

Parameter `rds.global_db_rpo` bersifat dinamis. Jika Anda memutuskan untuk tidak menghentikan semua transaksi tulis hingga keterlambatan cukup berkurang, Anda dapat meresetnya dengan cepat. Dalam kasus ini, Aurora akan menerima dan mengimplementasikan perubahan setelah beberapa saat.

> **Important**
>
> Dalam database global dengan hanya dua AWS Wilayah, kami sarankan untuk menjaga nilai default `rds.global_db_rpo` parameter di grup parameter Region sekunder. Jika tidak, melakukan failover karena hilangnya AWS Wilayah utama dapat menyebabkan Aurora menghentikan transaksi. Sebagai gantinya, tunggu hingga Aurora menyelesaikan pembangunan kembali cluster di AWS Wilayah lama yang gagal sebelum mengubah parameter ini untuk menerapkan RPO maksimum.

Jika Anda mengatur parameter ini seperti yang diuraikan di bawah ini, Anda juga dapat memantau metrik yang dihasilkannya. Anda dapat melakukannya menggunakan `psql` atau alat lain untuk membuat kueri klaster DB primer basis data global Aurora dan mendapatkan informasi terperinci tentang operasi basis data global berbasis Aurora PostgreSQL. Untuk mempelajari caranya, lihat Memantau basis data global berbasis Aurora Postgre SQL.

Topik

- Mengatur sasaran titik pemulihan
- Melihat sasaran titik pemulihan
- Menonaktifkan sasaran titik pemulihan

## Mengatur sasaran titik pemulihan

Parameter `rds.global_db_rpo` mengontrol pengaturan RPO untuk basis data PostgreSQL. Parameter ini didukung oleh Aurora PostgreSQL. Nilai yang valid untuk `rds.global_db_rpo` berkisar dari 20 detik hingga 2.147.483.647 detik (68 tahun). Pilih nilai yang realistis untuk memenuhi kasus penggunaan dan kebutuhan bisnis Anda. Misalnya, Anda mungkin ingin mengizinkan durasi RPO hingga 10 menit, sehingga Anda menetapkan nilai ke 600.

Anda dapat menetapkan nilai ini untuk basis data global berbasis Aurora PostgreSQL menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Cara mengatur RPO

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih klaster primer basis data global Aurora, lalu buka tab Konfigurasi untuk menemukan grup parameter klaster DB. Sebagai contoh, grup parameter default untuk klaster DB primer yang menjalankan Aurora PostgreSQL 11.7 adalah `default.aurora-postgresql11`.

   Grup parameter tidak dapat diedit secara langsung. Sebaliknya, lakukan hal berikut:

   - Buat grup parameter klaster DB kustom menggunakan grup parameter default yang sesuai sebagai titik awal. Misalnya, buat grup parameter klaster DB kustom berdasarkan `default.aurora-postgresql11`.
   - Pada grup parameter klaster DB kustom, tetapkan nilai parameter rds.global_db_rpo agar sesuai dengan kasus penggunaan Anda. Nilai yang valid berkisar dari 20 detik hingga nilai integer maksimum sebesar 2.147.483.647 (68 tahun).
   - Terapkan grup parameter klaster DB yang telah dimodifikasi ke klaster DB Aurora Anda.

Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.

AWS CLI

Untuk mengatur `rds.global_db_rpo` parameter, gunakan perintah modify-db-cluster-parameter-group CLI. Dalam perintahnya, tentukan nama grup parameter klaster primer dan nilai untuk parameter RPO.

Contoh berikut mengatur RPO ke 600 detik (10 menit) untuk grup parameter klaster DB primer, `my_custom_global_parameter_group`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

API RDS

Untuk memodifikasi `rds.global_db_rpo` parameter, gunakan operasi Amazon RDS Modify DBCluster ParameterGroup API.

## Melihat sasaran titik pemulihan

Sasaran titik pemulihan (RPO) dari basis data global disimpan di parameter `rds.global_db_rpo` untuk setiap klaster DB. Anda dapat terhubung ke titik akhir klaster sekunder yang ingin Anda lihat dan menggunakan `psql` untuk membuat kueri instans bagi nilai ini.

```
db-name=>show rds.global_db_rpo;
```

Jika parameter ini belum diatur, kueri akan menampilkan sebagai berikut:

```
rds.global_db_rpo
-------------------
 -1
(1 row)
```

Respons selanjutnya ini berasal dari klaster DB sekunder yang memiliki pengaturan RPO 1 menit.

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

Anda juga dapat menggunakan CLI untuk memperoleh nilai guna mengetahui apakah `rds.global_db_rpo` aktif pada salah satu klaster DB Aurora dengan menggunakan CLI untuk memperoleh nilai semua parameter `user` untuk klaster tersebut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

Untuk Windows:

```
aws rds describe-db-cluster-parameters ^
 --db-cluster-parameter-group-name lab-test-apg-global *
 --source user
```

Perintah akan menampilkan output serupa dengan yang berikut untuk semua parameter `user` selain parameter klaster DB `default-engine` atau `system`.

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
                "provisioned"
            ]
        }
    ]
}
```

Untuk mempelajari selengkapnya tentang cara melihat parameter dari grup parameter klaster, lihat Melihat nilai parameter untuk grup parameter cluster DB di Amazon Aurora.

## Menonaktifkan sasaran titik pemulihan

Untuk menonaktifkan RPO, reset parameter `rds.global_db_rpo`. Anda dapat mereset parameter menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Cara menonaktifkan RPO

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter klaster DB primer Anda.
4. Pilih Edit parameter.
5. Pilih kotak di sebelah parameter rds.global_db_rpo.
6. Pilih Reset.
7. Saat layar menampilkan Reset parameter dalam grup parameter DB, pilih Reset parameter.

Untuk informasi selengkapnya tentang cara mereset parameter dengan konsol, lihat Memodifikasi parameter dalam grup parameter cluster DB di Amazon Aurora.

AWS CLI

Untuk mengatur ulang `rds.global_db_rpo` parameter, gunakan perintah reset-db-cluster-parameter-group.

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

API RDS

Untuk mengatur ulang `rds.global_db_rpo` parameter, gunakan DBCluster ParameterGroup operasi Amazon RDS API Reset.

## Ketahanan Lintas Wilayah untuk klaster sekunder Global Database

Aurora PostgreSQL versi 16.6, 15.10, 14.15, 13.18, 12.22, atau lebih tinggi dan Aurora MySQL versi 3.09 atau lebih tinggi berisi peningkatan ketersediaan yang memungkinkan replika baca Region sekunder untuk mempertahankan kontinuitas layanan selama peristiwa yang tidak direncanakan seperti kegagalan perangkat keras, gangguan jaringan di seluruh Wilayah, volume besar transfer data antara cluster, dan lainnya. AWS

Meskipun replika baca tetap tersedia untuk permintaan aplikasi Anda, kelambatan replikasi dapat terus meningkat hingga resolusi acara yang tidak direncanakan. Anda dapat memantau jeda antara cluster primer dan sekunder menggunakan `AuroraGlobalDBProgressLag` CloudWatch metrik. Untuk mengukur end-to-end lag, termasuk jeda antara volume cluster dan instance DB dari cluster sekunder, tambahkan nilai `AuroraGlobalDBProgressLag` dan `AuroraReplicaLag` CloudWatch metrik. Untuk informasi selengkapnya tentang metrik, lihat. Referensi metrik untuk Aurora

Ketersediaan pembacaan Database Global untuk Aurora MySQL dan versi sebelumnya dari Aurora PostgreSQL dapat terpengaruh selama peristiwa yang tidak direncanakan tersebut.

Untuk informasi lebih lanjut tentang fitur baru di Aurora PostgreSQL 16.6, 15.10, 14.15, 13.18, dan 12.22, lihat PostgreSQL 16.6 di Catatan Rilis PostgreSQL Aurora.

Untuk informasi selengkapnya tentang fitur baru di Aurora MySQL versi 3.09 dan yang lebih tinggi, lihat Pembaruan mesin database untuk Amazon Aurora MySQL versi 3 di Catatan Rilis Aurora MySQL.

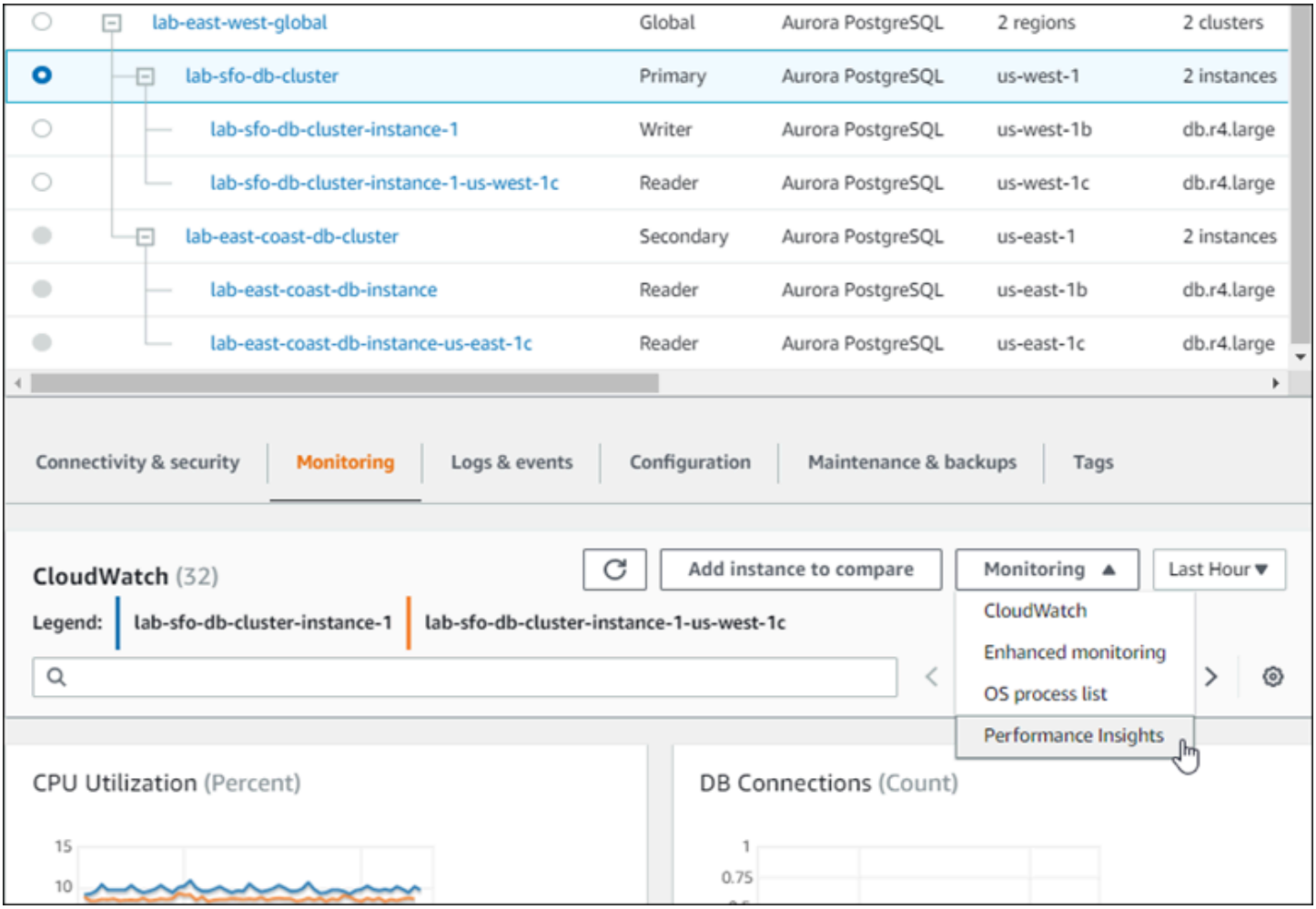
## Memantau basis data global Amazon Aurora

Ketika Anda membuat klaster DB Aurora yang membentuk basis data global Aurora Anda, Anda dapat memilih banyak pilihan yang memungkinkan Anda memantau performa DB klaster Anda. Opsi ini mencakup hal berikut:

- Amazon RDS Performance Insights — Mengaktifkan skema kinerja di mesin basis data Aurora yang mendasarinya. Untuk mempelajari selengkapnya tentang Wawasan Performa dan basis data global Aurora, lihat Memantau database global Amazon Aurora dengan Amazon Performance Insights RDS.
- Pemantauan yang disempurnakan — Menghasilkan metrik untuk proses atau pemanfaatan utas pada. CPU Untuk mempelajari tentang pemantauan yang ditingkatkan, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan.

- Amazon CloudWatch Logs - Menerbitkan jenis log tertentu ke CloudWatch Log. Log kesalahan diterbitkan secara default, tetapi Anda dapat memilih log lain khusus untuk mesin basis data Aurora Anda.
  - Untuk klaster Aurora DB SQL berbasis Aurora My, Anda dapat mengekspor log audit, log umum, dan log kueri lambat.
  - Untuk cluster Aurora DB SQL berbasis Aurora Postgre, Anda dapat mengekspor log Postgre. SQL
- Untuk database global SQL berbasis Aurora My, Anda dapat menanyakan `information_schema` tabel tertentu untuk memeriksa status database global Aurora Anda dan instance-nya. Untuk mempelajari caranya, lihat Memantau Aurora SQL Database global berbasis saya.
- Untuk database global SQL berbasis Aurora Postgre, Anda dapat menggunakan fungsi tertentu untuk memeriksa status database global Aurora Anda dan instance-nya. Untuk mempelajari caranya, lihat Memantau basis data global berbasis Aurora Postgre SQL.

Tangkapan layar berikut menunjukkan beberapa pilihan yang tersedia pada tab Pemantauan klaster DB Aurora primer dalam basis data global Aurora.




Untuk informasi selengkapnya, lihat Memantau metrik di klaster Amazon Aurora.

## Memantau database global Amazon Aurora dengan Amazon Performance Insights RDS

Anda dapat menggunakan Amazon RDS Performance Insights untuk database global Aurora Anda. Anda mengaktifkan fitur ini secara individual, untuk setiap klaster DB Aurora dalam basis data global Aurora Anda. Untuk melakukannya, pilih Aktifkan Wawasan Performa di bagian Konfigurasi tambahan pada halaman Buat basis data. Atau Anda dapat memodifikasi klaster DB Aurora Anda untuk menggunakan fitur ini setelah tersedia dan berjalan. Anda dapat mengaktifkan atau menonaktifkan Wawasan Performa untuk setiap klaster yang merupakan bagian dari basis data global Aurora Anda.

Pernyataan yang dibuat oleh Wawasan Performa berlaku untuk setiap klaster dalam global basis data. Saat Anda menambahkan sekunder baru Wilayah AWS ke database global Aurora yang sudah

menggunakan Performance Insights, pastikan Anda mengaktifkan Performance Insights di cluster yang baru ditambahkan. Klaster tersebut tidak mewarisi pengaturan Wawasan Performa dari basis data global yang sudah ada.

Anda dapat beralih Wilayah AWS saat melihat halaman Performance Insights untuk instance DB yang dilampirkan ke database global. Namun, Anda mungkin tidak melihat informasi kinerja segera setelah beralih Wilayah AWS. Meskipun instans DB mungkin memiliki nama yang identik di masing-masing Wilayah AWS, Performance URL Insights terkait berbeda untuk setiap instans DB. Setelah beralih Wilayah AWS, pilih nama instans DB lagi di panel navigasi Performance Insights.

Untuk instans DB yang dikaitkan dengan global basis data, faktor yang memengaruhi performa mungkin berbeda di setiap Wilayah AWS. Misalnya, instans DB di masing-masing Wilayah AWS mungkin memiliki kapasitas yang berbeda.

Untuk mempelajari lebih lanjut tentang menggunakan Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

## Memantau basis data global Aurora dengan Aliran Aktivitas Basis Data

Dengan menggunakan fitur Aliran Aktivitas Basis Data, Anda dapat memantau dan mengatur alarm untuk aktivitas audit di klaster DB di basis data global Anda. Anda memulai aliran aktivitas basis data pada setiap klaster DB secara terpisah. Setiap klaster mengirimkan data audit ke aliran Kinesis-nya sendiri dalam Wilayah AWS-nya sendiri. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

## Memantau Aurora SQL Database global berbasis saya

Untuk melihat status database global SQL berbasis Aurora My, kueri information_schema.aurora_global_db_status dan information_schema.aurora_global_db_instance_status tabel.

> **Note**
>
> `information_schema.aurora_global_db_instance_statusTabel information_schema.aurora_global_db_status` dan hanya tersedia dengan Aurora My SQL versi 3.04.0 dan database global yang lebih tinggi.

Untuk memantau basis data global SQL berbasis Aurora My

1. Connect ke endpoint cluster primer database global menggunakan SQL klien Saya. Untuk informasi selengkapnya tentang cara membuat koneksi, lihat Menghubungkan ke Amazon Aurora Global Database.
2. Lakukan kueri ke tabel `information_schema.aurora_global_db_status` dalam perintah mysql untuk membuat daftar volume primer dan sekunder. Kueri ini memunculkan waktu lag klaster DB sekunder basis data global, seperti pada contoh berikut.

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
-----------+---------------------+--------------------------------
+-------------------------+---------------------------------
+------------------------
us-east-1  |           183537946 |                              0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                              428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

Output mencakup baris untuk setiap klaster DB basis data global yang berisi kolom berikut:

- AWS_REGION— Di Wilayah AWS mana cluster DB ini berada. Untuk daftar tabel Wilayah AWS berdasarkan mesin, lihatKetersediaan wilayah.
- HIGHEST_ LSN _ WRITTEN — Nomor urutan log tertinggi (LSN) yang saat ini ditulis di cluster DB ini.

  Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNsdipesan sedemikian rupa sehingga yang lebih besar LSN mewakili transaksi selanjutnya.
- DURABILITY_ LAG _IN_ MILLISECONDS — Perbedaan nilai stempel waktu antara `HIGHEST_LSN_WRITTEN` pada cluster DB sekunder dan `HIGHEST_LSN_WRITTEN` pada cluster DB primer. Nilai ini selalu 0 pada klaster DB primer basis data global Aurora.
- RPO_ LAG _IN_ MILLISECONDS — Tujuan titik pemulihan (RPO) tertinggal. RPOKelambatan adalah waktu yang dibutuhkan untuk transaksi pengguna terbaru COMMIT untuk disimpan di

cluster DB sekunder setelah disimpan di cluster DB utama dari database global Aurora. Nilai ini selalu 0 pada klaster DB primer basis data global Aurora.

Secara sederhana, metrik ini menghitung tujuan titik pemulihan untuk setiap cluster Aurora SQL My DB di database global Aurora, yaitu, berapa banyak data yang mungkin hilang jika ada pemadaman. Seperti halnya lag, RPO diukur dalam waktu.

- LAST_ LAG _ CALCULATION _ TIMESTAMP — Stempel waktu yang menentukan kapan nilai terakhir dihitung untuk dan. `DURABILITY_LAG_IN_MILLISECONDS` `RPO_LAG_IN_MILLISECONDS` Nilai waktu seperti `1970-01-01 00:00:00+00` berarti ini adalah klaster DB primer.
- OLDEST_ READ _ VIEW _ TRX _ID — ID transaksi tertua yang dapat dibersihkan oleh instans DB penulis.

3. Lakukan kueri ke tabel `information_schema.aurora_global_db_instance_status` untuk membuat daftar semua instans DB sekunder baik untuk klaster DB primer maupun klaster DB sekunder.

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID            |              SESSION_ID              | AWS_REGION
 | DURABLE_LSN | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
---------------------+--------------------------------------+------------
+-------------+----------------------+-------------------------
+----------------------+-----------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                    0 |                      0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                       0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                     677
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |            183537698 |                20806930 |
 183537691 |                      21
(4 rows)
```

Output mencakup baris untuk setiap instans DB basis data global yang berisi kolom berikut:

- SERVER_ID — Pengidentifikasi server untuk instance DB.
- SESSION_ID — Pengidentifikasi unik untuk sesi saat ini. Nilai `MASTER_SESSION_ID` mengidentifikasi instans DB Penulis (primer).
- AWS_REGION— Di Wilayah AWS mana instans DB ini ada. Untuk daftar tabel Wilayah AWS berdasarkan mesin, lihatKetersediaan wilayah.
- DURABLE_ LSN — LSN Dibuat tahan lama dalam penyimpanan.
- HIGHEST_ LSN _ RECEIVED — Tertinggi yang LSN diterima oleh instans DB dari instans DB penulis.
- OLDEST_ READ _ VIEW _ TRX _ID — ID transaksi tertua yang dapat dibersihkan oleh instans DB penulis.
- OLDEST_ READ _ VIEW _ LSN — Yang tertua LSN digunakan oleh instans DB untuk membaca dari penyimpanan.
- VISIBILITY_ LAG _IN_ MSEC — Untuk pembaca di cluster DB primer, seberapa jauh instance DB ini tertinggal dari instance DB penulis dalam milidetik. Untuk pembaca di DB klaster sekunder, seberapa jauh DB instans ini tetap tertinggal di belakang volume sekunder dalam milidetik.

Untuk melihat perubahan nilai ini dari waktu ke waktu, pertimbangkan blok transaksi berikut di mana sisipan tabel memakan waktu satu jam.

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

Dalam beberapa kasus, mungkin terdapat pemutusan jaringan antara klaster DB primer dan klaster DB sekunder setelah pernyataan `BEGIN`. Jika demikian, MILLISECONDS nilai DURABILITY_ LAG _IN_ cluster DB sekunder mulai meningkat. Di akhir `INSERT` pernyataan, MILLISECONDS nilai DURABILITY_ LAG _IN_ adalah 1 jam. Namun, MILLISECONDS nilai RPO_ LAG _IN_ adalah 0 karena semua data pengguna yang dilakukan antara cluster DB primer dan cluster DB sekunder masih sama. Segera setelah `COMMIT` pernyataan selesai, nilai RPO_ LAG _IN_ meningkat MILLISECONDS.

# Memantau basis data global berbasis Aurora Postgre SQL

Untuk melihat status database global SQL berbasis Aurora Postgre, gunakan dan fungsi. `aurora_global_db_status` `aurora_global_db_instance_status`

> **Note**
>
> Hanya Aurora Postgre yang SQL mendukung dan fungsi. `aurora_global_db_status` `aurora_global_db_instance_status`

Untuk memantau basis data global berbasis Aurora Postgre SQL

1. Connect ke endpoint cluster primer database global menggunakan SQL utilitas Postgre seperti psql. Untuk informasi selengkapnya tentang cara membuat koneksi, lihat Menghubungkan ke Amazon Aurora Global Database.
2. Gunakan fungsi `aurora_global_db_status` dalam perintah psql untuk mencantumkan volume primer dan sekunder. Ini menunjukkan waktu lag klaster DB sekunder global basis data.

```
postgres=> select * from aurora_global_db_status();
```

```
aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
 last_lag_calculation_time  | feedback_epoch | feedback_xmin
------------+---------------------+------------------------+-----------------
+----------------------------+----------------+--------------
us-east-1  |         93763984222 |                     -1 |              -1 |
 1970-01-01 00:00:00+00     |              0 |              0
us-west-2  |         93763984222 |                    900 |            1090 |
 2020-05-12 22:49:14.328+00 |              2 |    3315479243
(2 rows)
```

Output mencakup baris untuk setiap klaster DB basis data global yang berisi kolom berikut:

- aws_region - Tempat cluster DB ini berada Wilayah AWS . Untuk daftar tabel Wilayah AWS berdasarkan mesin, lihatKetersediaan wilayah.
- highest_lsn_written — Nomor urutan log tertinggi (LSN) saat ini ditulis pada cluster DB ini.

Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan dalam log transaksi database. LSNsdipesan sedemikian rupa sehingga yang lebih besar LSN mewakili transaksi selanjutnya.

- durability_lag_in_msec – Perbedaan stempel waktu antara log sequence number tertinggi pada klaster DB sekunder (`highest_lsn_written`) dan `highest_lsn_written` pada klaster DB primer.
- rpo_lag_in_msec — Tujuan titik pemulihan () lag. RPO Lag ini adalah perbedaan waktu antara transaksi pengguna terbaru yang disimpan di DB klaster sekunder dan transaksi pengguna terbaru yang disimpan di DB klaster primer.
- last_lag_calculation_time – Stempel waktu ketika nilai terakhir dihitung untuk `durability_lag_in_msec` dan `rpo_lag_in_msec`.
- feedback_epoch – Masa yang digunakan klaster DB sekunder saat menghasilkan informasi hot standby.

  Hot standby adalah ketika klaster DB dapat terhubung dan mengueri saat server berada dalam mode pemulihan atau mode siaga. Umpan balik hot standby adalah informasi tentang klaster DB saat dalam status hot standby. Untuk informasi selengkapnya, lihat Siaga panas di dokumentasi SQL Postgre.
- feedback_xmin – ID transaksi aktif minimum (terlama) yang digunakan oleh klaster DB sekunder.

3. Gunakan fungsi `aurora_global_db_instance_status` untuk membuat daftar semua instans DB sekunder baik untuk klaster DB primer maupun klaster DB sekunder.

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------------+-------------------------------------
+------------+-------------+------------------+----------------+--------------
+----------------------+-----------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                  |                |              |
                     |
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |              2 |    3315479243 |
        93763985095 |                     10
```

```
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |             2 |    3315479243 |
        93763985089 |                  1017
(3 rows)
```

Output mencakup baris untuk setiap instans DB basis data global yang berisi kolom berikut:

- server_id – Pengidentifikasi server untuk instans DB.
- session_id – Pengidentifikasi unik untuk sesi saat ini.
- aws_region — Wilayah AWS Instance DB ini berada. Untuk daftar tabel Wilayah AWS berdasarkan mesin, lihatKetersediaan wilayah.
- durable_lsn — Dibuat tahan lama dalam penyimpanan. LSN
- highest_lsn_rcvd — Tertinggi yang LSN diterima oleh instans DB dari instance DB penulis.
- feedback_epoch – Masa yang digunakan instans DB saat menghasilkan informasi hot standby.

  Hot standby adalah ketika instans DB dapat terhubung dan mengueri saat server berada dalam mode pemulihan atau mode siaga. Umpan balik hot standby adalah informasi tentang instans DB saat dalam status hot standby. Untuk informasi selengkapnya, lihat SQL dokumentasi Postgre di Hot standby.
- feedback_xmin – ID transaksi aktif minimum (terlama) yang digunakan oleh instans DB.
- oldest_read_view_lsn — Yang tertua LSN digunakan oleh instans DB untuk membaca dari penyimpanan.
- visibility_lag_in_msec – Seberapa jauh instans DB ini tertinggal di belakang instans DB penulis.

Untuk melihat perubahan nilai ini dari waktu ke waktu, pertimbangkan blok transaksi berikut di mana sisipan tabel memakan waktu satu jam.

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

Dalam beberapa kasus, mungkin terdapat pemutusan jaringan antara klaster DB primer dan klaster DB sekunder setelah pernyataan `BEGIN`. Jika ya, nilai `durability_lag_in_msec` klaster DB sekunder mulai meningkat. Pada akhir pernyataan `INSERT`, nilai `durability_lag_in_msec` adalah 1 jam. Namun, nilai `rpo_lag_in_msec` adalah 0 karena semua data pengguna yang di-

commit antara klaster DB primer dan klaster DB sekunder masih sama. Segera setelah pernyataan `COMMIT` selesai, nilai `rpo_lag_in_msec` akan meningkat.

# Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya

Anda dapat menggunakan database global Aurora Anda dengan AWS layanan lain, seperti Amazon S3 dan. AWS Lambda Tindakan tersebut mengharuskan semua klaster DB Aurora dalam basis data global Anda memiliki hak istimewa, fungsi eksternal, dan sebagainya dalam Wilayah AWS masing-masing. Karena klaster DB sekunder Aurora hanya-baca dalam sebuah basis data global Aurora dapat dinaikkan ke peran primer, kami sarankan Anda mengatur hak tulis di awal, di semua klaster DB Aurora untuk layanan apa pun yang ingin Anda gunakan bersama basis data global Aurora Anda.

Prosedur berikut merangkum tindakan yang harus diambil untuk masing-masing Layanan AWS.

Untuk memanggil AWS Lambda fungsi dari database global Aurora

1. Untuk semua klaster Aurora yang membentuk basis data global Aurora, lakukan prosedur di Memanggil fungsi Lambda dari cluster Amazon Aurora My DB SQL.
2. Untuk setiap cluster dalam database global Aurora, atur (ARN) peran new IAM (IAM).
3. Untuk mengizinkan pengguna basis data dalam basis data global Aurora untuk menginvokasi fungsi Lambda, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan setiap klaster di basis data global Aurora.
4. Konfigurasikan setiap klaster dalam basis data global Aurora untuk memungkinkan koneksi keluar ke Lambda. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

Memuat data dari Amazon S3

1. Untuk semua klaster Aurora yang membentuk basis data global Aurora, lakukan prosedur di Memuat data ke dalam cluster Amazon Aurora My SQL DB dari file teks di bucket Amazon S3.
2. Untuk setiap klaster Aurora dalam database global, tetapkan parameter cluster `aurora_load_from_s3_role` atau `aws_default_s3_role` DB ke Amazon Resource Name (ARN) dari peran baruIAM. Jika IAM peran tidak ditentukan untuk`aurora_load_from_s3_role`, Aurora menggunakan IAM peran yang ditentukan dalam. `aws_default_s3_role`

3. Untuk mengizinkan pengguna basis data di basis data global Aurora untuk mengakses S3, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan setiap klaster Aurora dalam global basis data.
4. Konfigurasikan setiap klaster Aurora dalam basis data global untuk mengizinkan koneksi keluar ke S3. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

Untuk menyimpan data yang dikueri ke Amazon S3

1. Untuk semua cluster Aurora yang membentuk database global Aurora, lakukan prosedur di atau. Menyimpan data dari cluster Amazon Aurora My SQL DB ke dalam file teks di bucket Amazon S3 Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3
2. Untuk setiap klaster Aurora dalam database global, tetapkan parameter cluster `aurora_select_into_s3_role` atau `aws_default_s3_role` DB ke Amazon Resource Name (ARN) dari peran baruIAM. Jika IAM peran tidak ditentukan untuk`aurora_select_into_s3_role`, Aurora menggunakan IAM peran yang ditentukan dalam. `aws_default_s3_role`
3. Untuk mengizinkan pengguna basis data di basis data global Aurora untuk mengakses S3, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan setiap klaster Aurora dalam global basis data.
4. Konfigurasikan setiap klaster Aurora dalam basis data global untuk mengizinkan koneksi keluar ke S3. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.

# Meningkatkan basis data global Amazon Aurora

Peningkatan basis data global Aurora mengikuti prosedur yang sama seperti peningkatan klaster DB Aurora. Namun, berikut adalah beberapa perbedaan penting yang perlu diperhatikan sebelum Anda memulai prosesnya.

Kami menyarankan Anda meningkatkan klaster DB primer dan sekunder ke versi yang sama. Anda hanya dapat melakukan failover basis data lintas Wilayah terkelola pada basis data global Aurora jika klaster DB primer dan sekunder memiliki versi mesin tingkat utama, minor, dan patch yang sama. Namun, tingkat patch bisa berbeda, tergantung versi mesin kecil. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola.

# Peningkatan versi utama

Saat Anda melakukan peningkatan versi utama basis data global Amazon Aurora, Anda meningkatkan klaster basis data global, bukan klaster individual di dalamnya.

Untuk mempelajari cara meningkatkan basis data global Aurora PostgreSQL ke versi utama yang lebih tinggi, lihat Peningkatan mayor untuk basis data global.

> **Note**
>
> Dengan basis data global Aurora berdasarkan Aurora PostgreSQL, Anda tidak dapat melakukan peningkatan versi utama dari mesin DB Aurora jika fitur sasaran titik pemulihan (RPO) diaktifkan. Untuk informasi tentang fitur RPO, lihat Mengelola RPOs database global berbasis Aurora PostgreSQL.

Untuk mempelajari cara meningkatkan basis data global Aurora MySQL ke versi utama yang lebih tinggi, lihat Peningkatan besar di tempat untuk basis data global.

> **Note**
>
> Dengan database global Aurora berdasarkan Aurora MySQL, Anda dapat melakukan peningkatan di tempat dari Aurora MySQL versi 2 ke versi 3 hanya jika Anda mengatur parameter ke default dan Anda me-reboot database global Anda. `lower_case_table_names`
> Untuk melakukan peningkatan versi utama ke Aurora MySQL versi 3 saat menggunakan `lower_case_table_names`, gunakan proses berikut:
>
> 1. Hapus semua Wilayah sekunder dari klaster global. Ikuti langkah-langkahnya di Menghapus klaster dari basis data global Amazon Aurora.
> 2. Tingkatkan versi mesin dari Wilayah primer ke Aurora MySQL versi 3. Ikuti langkah-langkahnya di Cara melakukan peningkatan di tempat.
> 3. Tambahkan Wilayah sekunder ke klaster global. Ikuti langkah-langkahnya di Menambahkan Wilayah AWS ke database global Amazon Aurora.
>
> Anda juga dapat menggunakan metode pemulihan snapshot sebagai gantinya. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.

## Peningkatan versi kecil.

Untuk peningkatan kecil pada basis data global Aurora, Anda meningkatkan semua klaster sekunder sebelum meningkatkan klaster primer.

Untuk mempelajari cara meningkatkan basis data global Aurora PostgreSQL ke versi kecil yang lebih tinggi, lihat Cara melakukan peningkatan versi minor dan menerapkan patch. Untuk mempelajari cara meningkatkan basis data global Aurora MySQL ke versi kecil yang lebih tinggi, lihat Meningkatkan Aurora MySQL dengan mengubah versi mesin.

Sebelum Anda melakukan peningkatan, tinjau pertimbangan berikut:

- Peningkatan versi kecil klaster sekunder tidak memengaruhi ketersediaan atau penggunaan klaster primer dengan cara apa pun.
- Klaster sekunder harus memiliki setidaknya satu instans DB untuk melakukan peningkatan kecil.
- Jika Anda meng-upgrade database global MySQL Aurora ke versi 2.11.*, Anda harus meng-upgrade cluster DB primer dan sekunder Anda ke versi yang persis sama, termasuk tingkat patch.
- Jika Anda memutakhirkan database global Aurora PostgreSQL, Anda harus meng-upgrade cluster DB primer dan sekunder Anda ke versi dan tingkat patch yang persis sama. Untuk memperbarui level patch, terapkan semua tindakan pemeliharaan yang tertunda pada cluster sekunder.
- Untuk mendukung switchover atau failover lintas wilayah yang dikelola, Anda mungkin perlu memutakhirkan kluster DB primer dan sekunder ke versi yang sama persis, termasuk level patch. Persyaratan ini berlaku untuk Aurora MySQL dan beberapa versi Aurora PostgreSQL. Untuk daftar versi yang memungkinkan switchover dan failover antar cluster yang menjalankan level patch yang berbeda, lihat. Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola

### Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola

Jika Aurora Global Database Anda menjalankan salah satu versi mesin minor berikut, Anda dapat melakukan switchover atau failover lintas wilayah terkelola meskipun level patch cluster DB primer dan sekunder Anda tidak cocok. Untuk versi mesin minor yang lebih rendah dari yang ada di daftar ini, cluster DB primer dan sekunder Anda harus menjalankan level mayor, minor, dan patch yang sama untuk melakukan switchover atau failover lintas wilayah yang dikelola. Pastikan untuk meninjau informasi versi dan catatan dalam tabel berikut saat merencanakan peningkatan untuk klaster primer, klaster sekunder, atau keduanya.

> **Note**
>
> Untuk failover lintas Wilayah manual, Anda dapat melakukan proses failover selama klaster DB sekunder target menjalankan versi mesin utama dan kecil yang sama dengan klaster DB primer. Dalam hal ini, tingkat patch tidak perlu cocok.
> Jika versi mesin Anda memerlukan level patch yang identik, Anda dapat melakukan failover secara manual dengan mengikuti langkah-langkahnya. Melakukan failover manual untuk basis data global Aurora

| Mesin basis data | Versi mesin kecil | Catatan |
|---|---|---|
| Aurora MySQL | Tidak ada versi minor | Tak satu pun dari versi minor Aurora MySQL memungkinkan switchover lintas wilayah terkelola atau failover dengan tingkat patch yang berbeda antara cluster DB primer dan sekunder. |
| Aurora PostgreSQL | • Versi 15 atau versi utama yang lebih tinggi<br>• Versi 14.5 atau versi minor yang lebih tinggi<br>• Versi 13.8 atau versi minor yang lebih tinggi<br>• Versi 12.12 atau versi minor yang lebih tinggi<br>• Versi 11.17 atau versi minor yang lebih tinggi | Dengan versi engine yang tercantum di kolom sebelumnya, Anda dapat melakukan switchover atau failover lintas wilayah terkelola dari cluster DB primer dengan satu level patch ke cluster DB sekunder dengan tingkat patch yang berbeda.<br><br>Dengan versi minor yang lebih rendah dari ini, Anda dapat melakukan switchover atau failover lintas wilayah terkelola hanya jika level patch cluster DB primer dan sekunder cocok. |

| Mesin basis data | Versi mesin kecil | Catatan |
| --- | --- | --- |
| | | **Warning**<br>Saat memperbarui klaster di database global Anda ke salah satu versi tambalan berikut, Anda tidak akan dapat melakukan switchover atau failover lintas wilayah hingga semua cluster di database global Anda menjalankan salah satu versi tambalan ini atau yang lebih baru.<br>• Patch versi 16.1.6, 16.2.4, 16.3.2, dan 16.4.2<br>• Patch versi 15.3.8, 15.4.9, 15.5.6, 15.6.4, 15.7.2, dan 15.8.2<br>• Patch versi 14.8.8, 14.9.9, 14.10.6, 14.11.4, 14.12.2, dan 14.13.2 |

# Proksi Amazon RDS untuk Aurora

Dengan menggunakan Proksi Amazon RDS, Anda dapat mengizinkan berbagai aplikasi untuk berkumpul dan berbagi koneksi basis data untuk meningkatkan kemampuan penskalaannya. Proksi RDS membuat aplikasi lebih tangguh terhadap kegagalan basis data dengan secara otomatis menghubungkan ke sebuah instans DB siaga sekaligus menjaga koneksi aplikasi. Dengan menggunakan RDS Proxy, Anda juga dapat menerapkan autentikasi AWS Identity and Access Management (IAM) untuk database, dan menyimpan kredensyal dengan aman. AWS Secrets Manager

Dengan Proksi RDS, Anda dapat menangani lonjakan yang tidak dapat diprediksi dalam lalu lintas basis data. Jika tidak, lonjakan ini dapat menyebabkan masalah karena permintaan koneksi berlebihan atau koneksi baru yang dibuat sangatlah tinggi. Proksi RDS membangun kumpulan koneksi basis data dan menggunakan ulang koneksi dalam kumpulan ini. Pendekatan ini menghindari overhead memori dan CPU dari membuka koneksi basis data baru. Untuk melindungi basis data dari permintaan berlebihan, Anda dapat mengontrol jumlah koneksi basis data yang dibuat.

Proksi RDS mengantrekan atau membatasi koneksi aplikasi yang tidak dapat dilayani segera dari kumpulan koneksi. Meskipun latensi dapat meningkat, aplikasi Anda dapat terus diskalakan tanpa kegagalan mendadak atau membanjiri basis data. Jika permintaan koneksi melebihi batas yang Anda tentukan, Proksi RDS akan menolak koneksi aplikasi (yakni menumpahkan beban). Selain itu, Proksi RDS akan mempertahankan performa yang dapat diprediksi untuk beban yang dapat dilayani RDS dengan kapasitas yang tersedia.

Anda dapat mengurangi overhead untuk memproses kredensial dan membangun koneksi yang aman untuk setiap koneksi baru. Proksi RDS dapat menangani beberapa dari pekerjaan itu untuk mewakili basis data.

Proksi RDS sepenuhnya kompatibel dengan versi mesin yang didukungnya. Anda dapat mengaktifkan Proksi RDS pada sebagian besar aplikasi tanpa perubahan kode. Untuk daftar versi mesin yang didukung, lihat Daerah yang Didukung dan mesin Aurora DB untuk Amazon RDS Proxy.

Topik


- Ketersediaan wilayah dan versi
- Kuota dan Pembatasan untuk Proksi RDS
- Merencanakan lokasi penggunaan Proksi RDS

- RDSKonsep dan terminologi proxy
- Memulai dengan RDS Proxy
- Mengelola RDS Proxy
- Bekerja dengan titik akhir Proksi Amazon RDS
- Memantau metrik Proxy RDS dengan Amazon CloudWatch
- Bekerja dengan acara RDS Proxy
- RDSContoh baris perintah proxy
- Pemecahan masalah untuk Proksi RDS
- Penggunaan Proksi RDS dengan AWS CloudFormation
- Menggunakan Proksi RDS dengan basis data global Aurora

## Ketersediaan wilayah dan versi

Untuk informasi tentang dukungan versi mesin basis data dan ketersediaan Proxy RDS dalam bentuk tertentu Wilayah AWS, lihatDaerah yang Didukung dan mesin Aurora DB untuk Amazon RDS Proxy.

## Kuota dan Pembatasan untuk Proksi RDS

Kuota dan batasan berikut berlaku untuk Proksi RDS:

- Setiap Akun AWS ID dibatasi hingga 20 proxy. Jika aplikasi Anda memerlukan lebih banyak proxy, minta peningkatan melalui halaman Service Quotas di dalam. AWS Management Console Di halaman Service Quotas, pilih Amazon Relational Database Service (Amazon RDS) dan cari Proxy untuk meminta peningkatan kuota. AWS dapat secara otomatis meningkatkan kuota atau peninjauan permintaan Anda yang tertunda oleh Dukungan.
- Setiap proksi dapat memiliki hingga 200 rahasia Secrets Manager terkait. Sehingga, setiap proksi dapat terhubung ke hingga 200 akun pengguna yang berbeda pada waktu tertentu.
- Setiap proksi memiliki titik akhir default. Anda juga dapat menambahkan hingga 20 titik akhir proksi untuk setiap proksi. Anda dapat membuat, melihat, mengubah, dan menghapus titik akhir ini.
- Dalam sebuah klaster Aurora, semua koneksi yang menggunakan titik akhir proksi default ditangani oleh instans penulis Aurora. Untuk melakukan penyeimbangan beban untuk beban kerja padat baca, Anda dapat membuat titik akhir hanya-baca untuk proksi. Titik akhir tersebut meneruskan koneksi ke titik akhir pembaca klaster. Dengan begitu, koneksi proksi Anda dapat memanfaatkan skalabilitas baca Aurora. Untuk informasi selengkapnya, lihat Ikhtisar titik akhir proksi.

- Anda dapat menggunakan Proksi RDS dengan klaster Aurora Nirserver v2, tetapi tidak dengan klaster Aurora Nirserver v1.
- Proksi RDS Anda harus berada dalam cloud privat virtual (VPC) yang sama seperti basis data. Proksi tersebut tidak dapat diakses publik, meskipun basis datanya dapat diakses publik. Misalnya, jika membuat prototipe basis data di host lokal, Anda tidak dapat terhubung ke proksi kecuali Anda menyiapkan persyaratan jaringan yang diperlukan untuk mengizinkan koneksi ke proksi. Ini karena host lokal Anda berada di luar VPC proksi.

> **Note**
>
> Untuk klaster DB Aurora, Anda dapat mengaktifkan akses lintas-VPC. Untuk melakukannya, buat titik akhir tambahan untuk proksi dan tentukan VPC, subnet, dan grup keamanan yang berbeda dengan titik akhir tersebut. Untuk informasi selengkapnya, lihat Mengakses database RDS di seluruh VPCs.

- Anda tidak dapat menggunakan Proksi RDS dengan VPC yang penghuniannya diatur ke `dedicated`.
- Jika Anda menggunakan Proksi RDS dengan klaster DB Aurora yang autentikasi IAM-nya diaktifkan, periksa autentikasi pengguna. Pengguna yang terhubung melalui proksi harus melakukan autentikasi melalui kredensial masuk. Untuk detail tentang Secrets Manager dan dukungan IAM di Proksi RDS, lihat Menyiapkan kredensi database AWS Secrets Manager untuk RDS Proxy dan Mengkonfigurasi otentikasi IAM untuk RDS Proxy.
- Anda tidak dapat menggunakan Proksi RDS dengan DNS kustom saat menggunakan validasi nama host SSL.
- Setiap proksi dapat dikaitkan dengan satu klaster DB target. Namun, Anda dapat mengaitkan beberapa proksi dengan klaster DB yang sama.
- Pernyataan apa pun dengan teks berukuran lebih dari 16 KB menyebabkan proksi menyematkan sesi ke koneksi saat ini.
- Wilayah tertentu memiliki batasan Zona Ketersediaan (AZ) untuk dipertimbangkan saat membuat proksi. Wilayah AS Timur (Virginia Utara) tidak mendukung Proksi RDS di Zona Ketersediaan `use1-az3`. Wilayah AS Barat (California Utara) tidak mendukung Proksi RDS di Zona Ketersediaan `usw1-az2`. Saat memilih subnet sekaligus membuat proksi, pastikan Anda tidak memilih subnet di Zona Ketersediaan yang disebutkan di atas.
- Saat ini, RDS Proxy tidak mendukung kunci konteks kondisi global apa pun.

Untuk informasi selengkapnya tentang kunci konteks kondisi global, lihat Kunci konteks kondisi global AWS dalam Panduan Pengguna IAM.

- Anda tidak dapat menggunakan RDS Proxy dengan RDS Custom untuk SQL Server.
- Untuk mencerminkan modifikasi grup parameter database apa pun ke proxy Anda, reboot instance diperlukan bahkan jika Anda memilih untuk segera menerapkan perubahan Anda. Untuk parameter tingkat cluster, diperlukan reboot seluruh cluster.
- Proxy Anda secara otomatis membuat pengguna `rdsproxyadmin` DB saat Anda mendaftarkan target proxy. Ini adalah pengguna yang dilindungi yang penting untuk fungsionalitas proxy. Anda harus menghindari merusak `rdsproxyadmin` pengguna dalam kapasitas apa pun. Menghapus atau memodifikasi `rdsproxyadmin` pengguna atau izinnya dapat mengakibatkan tidak tersedianya proxy ke aplikasi Anda.

Untuk batasan tambahan untuk setiap mesin DB, lihat bagian berikut:

- Batasan tambahan untuk Aurora MySQL
- Batasan tambahan untuk Aurora PostgreSQL

## Batasan tambahan untuk Aurora MySQL

Batasan tambahan berikut berlaku untuk Proksi RDS dengan basis data Aurora MySQL:

- Dukungan RDS Proxy untuk `caching_sha2_password` otentikasi memerlukan koneksi aman (TLS).
- Dukungan Proxy RDS `caching_sha2_password` diketahui memiliki masalah kompatibilitas dengan versi driver go-sql tertentu.
- Saat menggunakan driver MySQL 8.4 C, API mungkin membentuk paket `mysql_stmt_bind_named_param` yang salah bentuk jika jumlah parameter melebihi jumlah placeholder dalam pernyataan yang disiapkan. Ini menghasilkan tanggapan yang salah. Untuk informasi selengkapnya, lihat laporan bug MySQL.
- Saat ini, semua proksi mendengarkan di port 3306 untuk MySQL. Proksi ini masih terhubung ke basis data Anda menggunakan port yang sudah ditentukan dalam pengaturan basis data.
- Anda tidak dapat menggunakan RDS Proxy dengan database MySQL yang dikelola sendiri dalam instance. EC2

- Anda tidak dapat menggunakan Proksi RDS dengan instans DB RDS for MySQL dengan parameter `read_only` dalam grup parameter DB-nya diatur ke 1.
- Proksi RDS tidak mendukung mode terkompresi MySQL. Misalnya, Proksi RDS tidak mendukung kompresi yang digunakan oleh opsi `--compress` atau `-C` perintah `mysql`.
- Koneksi basis data yang memproses perintah `GET DIAGNOSTIC` mungkin menampilkan informasi yang tidak akurat saat Proksi RDS menggunakan kembali koneksi basis data yang sama untuk menjalankan kueri lain. Hal ini bisa terjadi ketika Proksi RDS me-multipleks koneksi basis data.
- Beberapa pernyataan dan fungsi SQL seperti `SET LOCAL` dapat mengubah status koneksi tanpa menyebabkan penyematan. Untuk perilaku penyematan terbaru, lihat Menghindari menyematkan Proxy RDS.
- Menggunakan `ROW_COUNT()` fungsi dalam kueri multi-pernyataan tidak didukung.
- RDS Proxy tidak mendukung aplikasi klien yang tidak dapat menangani beberapa pesan respons dalam satu catatan TLS.
- RDS Proxy tidak mendukung kata sandi ganda MySQL.

> **Important**
>
> Untuk proksi yang terkait dengan basis data MySQL, jangan atur parameter konfigurasi `sql_auto_is_null` ke `true` atau nilai bukan nol dalam kueri inisialisasi. Tindakan ini bisa menyebabkan perilaku aplikasi yang salah.

## Batasan tambahan untuk Aurora PostgreSQL

Batasan tambahan berikut berlaku untuk Proksi RDS dengan basis data Aurora PostgreSQL:

- Proksi RDS tidak mendukung filter penyematan sesi untuk PostgreSQL.
- Saat ini, semua proksi mendengarkan di port 5432 untuk PostgreSQL.
- Untuk PostgreSQL, Proxy RDS saat ini tidak mendukung pembatalan kueri dari klien dengan mengeluarkan `CancelRequest`. Misalnya, hal ini bisa terjadi ketika Anda membatalkan kueri yang berjalan lama dalam sesi psql interaktif dengan menggunakan Ctrl+C.
- Hasil dari fungsi lastval PostgreSQL tidak selalu akurat. Sebagai solusi, gunakan pernyataan INSERT dengan klausul `RETURNING`.
- Proksi RDS saat ini tidak mendukung mode replikasi streaming.

- `default`Database harus ada.
- Jika Anda menggunakan `ALTER ROLE` untuk mengubah peran penggunaSET `ROLE`, koneksi berikutnya sebagai pengguna tersebut ke proxy mungkin tidak menggunakan pengaturan peran ini, jika koneksi tersebut mengalami penyematan. Untuk mencegah hal ini, saat menggunakan proxy, gunakan `SET ROLE` dalam kueri inisialisasi proxy. Untuk informasi selengkapnya, lihat Kueri inisialisasi diMembuat proxy untuk .

> **Important**
>
> Untuk proksi yang ada dengan basis data PostgreSQL, jika Anda mengubah autentikasi basis data untuk menggunakan `SCRAM` saja, proksi akan menjadi tidak tersedia selama maksimal 60 detik. Untuk menghindari masalah ini, lakukan salah satu tindakan berikut:
>
> - Pastikan basis data mengizinkan autentikasi `SCRAM` dan `MD5`.
> - Untuk menggunakan autentikasi `SCRAM` saja, buat proksi baru, migrasi lalu lintas aplikasi ke proksi baru, lalu hapus proksi yang sebelumnya terkait dengan basis data.

## Merencanakan lokasi penggunaan Proksi RDS

Anda dapat menentukan instans, klaster, dan aplikasi DB mana yang mungkin paling banyak mendapatkan manfaat dari penggunaan Proksi RDS. Untuk melakukannya, pertimbangkan faktor-faktor berikut:

- klaster DB apa pun yang mengalami kesalahan "terlalu banyak koneksi" adalah kandidat yang baik untuk dikaitkan dengan proksi. Ini sering ditandai dengan nilai `ConnectionAttempts` CloudWatch metrik yang tinggi. Proksi tersebut memungkinkan aplikasi untuk membuka banyak koneksi klien sekaligus mengelola koneksi jangka panjang dalam jumlah lebih kecil ke instans DB.
- Untuk cluster DB yang menggunakan kelas AWS instans yang lebih kecil, seperti T2 atau T3, menggunakan proxy dapat membantu menghindari kondisi. out-of-memory Tindakan ini juga dapat membantu mengurangi overhead CPU untuk membangun koneksi. Kondisi ini dapat terjadi saat berurusan dengan koneksi dalam jumlah besar.
- Anda dapat memantau CloudWatch metrik Amazon tertentu untuk menentukan apakah cluster DB mendekati jenis batas tertentu. Batasan ini ditujukan untuk jumlah koneksi dan memori terkait dengan pengelolaan koneksi. Anda juga dapat memantau CloudWatch metrik tertentu untuk menentukan apakah cluster DB menangani banyak koneksi berumur pendek. Pembukaan dan

penutupan koneksi tersebut dapat membebani overhead performa pada basis data Anda. Untuk informasi tentang metrik yang akan dipantau, lihat Memantau metrik Proxy RDS dengan Amazon CloudWatch.

- Fungsi AWS Lambda juga bisa menjadi kandidat yang tepat untuk penggunaan proksi. Fungsi ini sering membuat koneksi basis data pendek yang mendapatkan manfaat dari kumpulan koneksi yang ditawarkan oleh Proksi RDS. Anda dapat memanfaatkan autentikasi IAM apa pun yang Anda miliki untuk fungsi Lambda, alih-alih mengelola kredensial basis data dalam kode aplikasi Lambda.
- Aplikasi yang biasanya membuka dan menutup koneksi basis data dalam jumlah besar dan tidak memiliki mekanisme pengumpulan koneksi bawaan adalah kandidat yang tepat untuk menggunakan proksi.
- Aplikasi yang dapat mempertahankan koneksi terbuka dalam jumlah besar selama jangka waktu yang lama biasanya merupakan kandidat yang tepat untuk penggunaan proksi. Aplikasi dalam industri seperti Perangkat Lunak sebagai Layanan (SaaS) atau e-niaga sering kali meminimalkan latensi untuk permintaan basis data dengan membiarkan koneksi terbuka. Dengan Proksi RDS, aplikasi dapat mempertahankan lebih banyak koneksi tetap terbuka daripada saat terhubung langsung ke instans DB.
- Mungkin Anda belum mengadopsi autentikasi IAM dan Secrets Manager karena kompleksitas penyiapan autentikasi tersebut untuk semua instans DB. Jika demikian, Anda dapat terus menerapkan metode autentikasi yang sudah ada dan mendelegasikan autentikasi ke proksi. Proksi dapat menerapkan kebijakan autentikasi koneksi klien untuk aplikasi tertentu. Anda dapat memanfaatkan autentikasi IAM apa pun yang Anda miliki untuk fungsi Lambda, alih-alih mengelola kredensial basis data dalam kode aplikasi Lambda.
- Proksi RDS dapat membantu membuat aplikasi lebih tangguh dan transparan terhadap kegagalan basis data. Proksi RDS melewati cache Sistem Nama Domain (DNS) untuk mengurangi waktu failover hingga 66% untuk instans DB basis data Aurora Multi-AZ. Proksi RDS juga secara otomatis merutekan lalu lintas ke instans basis data baru sekaligus mempertahankan koneksi aplikasi. Hal ini membuat failover lebih transparan untuk aplikasi.

## RDSKonsep dan terminologi proxy

Anda dapat menyederhanakan manajemen koneksi untuk Aurora DB Anda dengan menggunakan Proxy. RDS

RDSProxy menangani lalu lintas jaringan antara aplikasi klien dan database. Tindakan ini dilakukan secara aktif terlebih dahulu dengan memahami protokol basis data. Kemudian menyesuaikan perilakunya berdasarkan SQL operasi dari aplikasi Anda dan set hasil dari database.

RDSProxy mengurangi memori dan CPU overhead untuk manajemen koneksi pada database Anda. Basis data membutuhkan lebih sedikit memori dan CPU sumber daya ketika aplikasi membuka banyak koneksi simultan. Logika juga tidak dibutuhkan dalam aplikasi Anda untuk menutup dan membuka kembali koneksi yang idle dalam waktu yang lama. Demikian pula, logika aplikasi yang dibutuhkan untuk membangun kembali koneksi juga lebih sedikit jika terjadi masalah pada basis data.

Infrastruktur untuk RDS Proxy sangat tersedia dan digunakan di beberapa Availability Zones (AZs). Perhitungan, memori, dan penyimpanan untuk RDS Proxy tidak tergantung pada cluster Anda. Independensi ini membantu menurunkan overhead pada server basis data Anda, sehingga dapat mencurahkan sumber dayanya untuk melayani beban kerja basis data. Sumber daya komputasi RDS Proxy tanpa server, secara otomatis menskalakan berdasarkan beban kerja database Anda.

Topik


- Ikhtisar konsep RDS Proxy
- Pengumpulan koneksi
- RDSKeamanan proxy
- Failover
- Transaksi


## Ikhtisar konsep RDS Proxy

RDSProxy menangani infrastruktur untuk melakukan penyatuan koneksi dan fitur lain yang dijelaskan di bagian berikutnya. Anda melihat proxy yang diwakili di RDS konsol di halaman Proxy.

Setiap proxy menangani koneksi ke cluster Aurora DB tunggal. Proxy secara otomatis menentukan instance penulis saat ini untuk instans Aurora yang disediakan cluster.

Koneksi yang dibiarkan terbuka dan disediakan oleh proksi untuk digunakan oleh aplikasi basis data Anda akan membentuk kumpulan koneksi.

Secara default, RDS Proxy dapat menggunakan kembali koneksi setelah setiap transaksi dalam sesi Anda. Penggunaan kembali tingkat transaksi ini disebut multiplexing. Ketika RDS Proxy sementara menghapus koneksi dari kumpulan koneksi untuk menggunakannya kembali, operasi itu disebut meminjam koneksi. Ketika aman untuk melakukannya, RDS Proxy mengembalikan koneksi itu ke kumpulan koneksi.

Dalam beberapa kasus, RDS Proxy tidak dapat memastikan bahwa aman untuk menggunakan kembali koneksi database di luar sesi saat ini. Untuk kasus seperti ini, Proksi RDS akan tetap

mempertahankan sesi pada koneksi yang sama hingga sesi berakhir. Perilaku fallback ini disebut pinning.

Proksi memiliki titik akhir default. Anda tersambung ke titik akhir ini saat bekerja dengan cluster Aurora DB. Anda melakukannya alih-alih terhubung ke titik akhir baca/tulis yang terhubung langsung ke klaster. Titik akhir tujuan khusus untuk klaster Aurora tetap tersedia untuk Anda gunakan. Untuk cluster DB Aurora , Anda juga dapat membuat endpoint baca/tulis dan read-only tambahan. Untuk informasi selengkapnya, lihat Ikhtisar titik akhir proksi.

Misalnya, Anda masih dapat terhubung ke titik akhir klaster untuk koneksi baca/tulis tanpa pengumpulan koneksi. Anda masih dapat terhubung ke titik akhir pembaca untuk koneksi keseimbangan beban hanya-baca. Anda masih dapat terhubung ke titik akhir instans untuk melakukan diagnosis dan memecahkan masalah instans DB tertentu dengan klaster. Jika Anda menggunakan AWS layanan lain seperti AWS Lambda untuk terhubung ke RDS database, ubah pengaturan koneksi mereka untuk menggunakan titik akhir proxy. Misalnya, Anda menentukan titik akhir proxy untuk memungkinkan fungsi Lambda mengakses database Anda sambil memanfaatkan fungsionalitas ProxyRDS.

Setiap proksi berisi sebuah grup target. Grup target ini mewujudkan cluster Aurora DB yang dapat dihubungkan oleh proxy. Untuk klaster Aurora, grup target dikaitkan secara default dengan semua instans DB dalam klaster tersebut. Dengan demikian, proksi dapat terhubung ke instans DB Aurora mana pun yang dipromosikan menjadi instans penulis dalam klaster. Aurora DB cluster yang terkait dengan proxy disebut target proxy itu. Untuk kenyamanan, saat Anda membuat proxy melalui konsol, RDS Proxy juga membuat grup target yang sesuai dan mendaftarkan target terkait secara otomatis.

Keluarga mesin adalah serangkaian mesin basis data terkait yang menggunakan protokol DB yang sama. Anda dapat memilih keluarga mesin untuk setiap proksi yang Anda buat.

## Pengumpulan koneksi

Setiap proxy melakukan penyatuan koneksi secara terpisah untuk instance penulis dan pembaca dari database terkait DB. Pengumpulan koneksi adalah pengoptimalan yang menurunkan overhead yang terkait dengan pembukaan dan penutupan koneksi dan dengan menjaga banyak koneksi terbuka secara bersamaan. Overhead ini mencakup memori yang diperlukan untuk menangani setiap koneksi baru. Ini juga melibatkan CPU overhead untuk menutup setiap koneksi dan membuka yang baru. Contohnya termasuk Transport LayerSecurity/Secure Sockets Layer (TLS/SSL) jabat tangan, otentikasi, kemampuan negosiasi, dan sebagainya. Pengumpulan koneksi menyederhanakan logika aplikasi Anda. Anda tidak perlu menulis kode aplikasi untuk meminimalkan jumlah koneksi terbuka secara bersamaan.

Setiap proksi juga melakukan multipleks koneksi, yang juga dikenal sebagai penggunaan ulang koneksi. Dengan multiplexing, RDS Proxy melakukan semua operasi untuk transaksi menggunakan satu koneksi database yang mendasarinya. RDSkemudian dapat menggunakan koneksi yang berbeda untuk transaksi berikutnya. Anda dapat membuka banyak koneksi ke proksi secara bersamaan, dan proksi akan mempertahankan koneksi terbuka dalam jumlah yang lebih kecil ke instans atau klaster DB. Tindakan ini akan lebih meminimalkan overhead memori untuk koneksi di server basis data. Teknik ini juga mengurangi kemungkinan kesalahan "terlalu banyak koneksi".

## RDSKeamanan proxy

RDSProxy menggunakan mekanisme RDS keamanan yang ada sepertiTLS/SSLdan AWS Identity and Access Management (IAM). Untuk informasi umum tentang fitur keamanan tersebut, lihat Keamanan dalam Amazon Aurora. Juga, pastikan untuk membiasakan diri dengan cara kerja Aurora dengan otentikasi, otorisasi, dan bidang keamanan lainnya.

RDSProxy dapat bertindak sebagai lapisan keamanan tambahan antara aplikasi klien dan database yang mendasarinya. Misalnya, Anda dapat terhubung ke proxy menggunakan TLS 1.3, meskipun instans DB yang mendasarinya mendukung versi yang lebih lamaTLS. Anda dapat terhubung ke proxy menggunakan IAM peran. Hal ini terjadi bahkan jika proksi terhubung ke basis data menggunakan pengguna native dan metode autentikasi kata sandi. Dengan menggunakan teknik ini, Anda dapat menerapkan persyaratan autentikasi yang kuat untuk aplikasi basis data tanpa sebuah upaya migrasi yang mahal untuk instans DB itu sendiri.

Anda menyimpan kredensi database yang digunakan oleh RDS Proxy di. AWS Secrets Manager Setiap pengguna database untuk cluster Aurora DB yang diakses oleh proxy harus memiliki rahasia yang sesuai di Secrets Manager. Anda juga dapat mengatur IAM otentikasi untuk pengguna RDS Proxy. Dengan demikian, Anda dapat menerapkan IAM otentikasi untuk akses database bahkan jika database menggunakan otentikasi kata sandi asli. Sebaiknya gunakan fitur keamanan ini, alih-alih menyematkan kredensial basis data dalam kode aplikasi Anda.

### MenggunakanTLS/SSLdengan RDS Proxy

Anda dapat terhubung ke RDS Proxy menggunakan SSL protokolTLS/.

> **Note**
>
> RDSProxy menggunakan sertifikat dari AWS Certificate Manager (ACM). Jika Anda menggunakan RDS Proxy, Anda tidak perlu mengunduh RDS sertifikat Amazon atau memperbarui aplikasi yang menggunakan koneksi RDS Proxy.

TLSUntuk menerapkan semua koneksi antara proxy dan database Anda, Anda dapat menentukan pengaturan Memerlukan Transport Layer Security ketika Anda membuat atau memodifikasi proxy di AWS Management Console.

RDSProxy juga dapat memastikan bahwa sesi Anda menggunakanTLS/SSLantara klien Anda dan titik akhir RDS Proxy. Untuk meminta RDS Proxy melakukannya, tentukan persyaratan di sisi klien. SSLvariabel sesi tidak diatur untuk SSL koneksi ke database menggunakan RDS Proxy.

- Untuk Aurora My SQL, tentukan persyaratan di sisi klien dengan `--ssl-mode` parameter saat Anda menjalankan perintah. `mysql`
- Untuk dan SQL Aurora Postgre SQL, tentukan `sslmode=require` sebagai bagian dari `conninfo` string saat Anda menjalankan perintah. `psql`

RDSProxy mendukung TLS protokol versi 1.0, 1.1, 1.2, dan 1.3. Anda dapat terhubung ke proxy menggunakan versi yang lebih tinggi TLS dari yang Anda gunakan dalam database yang mendasarinya.

Secara default, program klien membuat koneksi terenkripsi dengan RDS Proxy, dengan kontrol lebih lanjut tersedia melalui opsi. `--ssl-mode` Dari sisi klien, RDS Proxy mendukung semua SSL mode.

Untuk klien, SSL modenya adalah sebagai berikut:

PREFERRED

SSLIni adalah pilihan pertama, tetapi tidak diperlukan.

DISABLED

Tidak SSL diperbolehkan.

REQUIRED

MenegakkanSSL.

VERIFY_CA

Menegakkan SSL dan memverifikasi otoritas sertifikat (CA).

VERIFY_IDENTITY

Menegakkan SSL dan memverifikasi nama host CA dan CA.

Saat menggunakan sebuah klien dengan `--ssl-mode VERIFY_CA` atau `VERIFY_IDENTITY`, tentukan opsi `--ssl-ca` yang menunjuk ke CA dalam format `.pem`. Untuk `.pem` file yang akan digunakan, unduh semua root CA PEMs dari Amazon Trust Services dan tempatkan ke dalam satu `.pem` file.

RDSProxy menggunakan sertifikat wildcard, yang berlaku untuk domain dan subdomainnya. Jika Anda menggunakan `mysql` klien untuk terhubung dengan SSL mode`VERIFY_IDENTITY`, saat ini Anda harus menggunakan perintah My SQL 8.0-compatible`mysql`.

## Failover

Failover adalah fitur ketersediaan tinggi yang menggantikan instans basis data dengan yang lain saat instans asli tidak tersedia. Failover dapat terjadi karena sebuah masalah pada instans basis data. Bisa juga bagian dari prosedur pemeliharaan normal, seperti saat peningkatan basis data. Failover berlaku untuk cluster Aurora DB dengan satu atau lebih instance pembaca selain instance penulis.

Terhubung melalui proksi membuat aplikasi Anda lebih tangguh menghadapi failover basis data. Ketika instans DB asli menjadi tidak tersedia, RDS Proxy terhubung ke database siaga tanpa menjatuhkan koneksi aplikasi idle. Hal ini dapat membantu mempercepat dan menyederhanakan proses failover. Dampaknya terhadap aplikasi juga lebih kecil dibandingkan masalah boot ulang atau basis data pada umumnya.

Tanpa RDS Proxy, failover melibatkan pemadaman singkat. Selama pemadaman, Anda tidak dapat melakukan operasi tulis pada basis data dengan failover. Koneksi semua basis data yang ada akan terganggu, dan aplikasi Anda harus membuka ulang koneksi tersebut. Basis data akan tersedia untuk koneksi dan operasi tulis baru saat instans DB hanya-baca dipromosikan menggantikan yang tidak tersedia.

Selama kegagalan DB, RDS Proxy terus menerima koneksi pada alamat IP yang sama dan secara otomatis mengarahkan koneksi ke instans DB primer yang baru. Klien yang terhubung melalui RDS Proxy tidak rentan terhadap hal-hal berikut:

- Domain Name System (DNS) penundaan propagasi pada failover.
- DNSCaching lokal.
- Waktu koneksi habis.
- Ketidakpastian tentang instans DB mana yang merupakan instans penulis saat ini.
- Menunggu respons kueri dari penulis sebelumnya yang menjadi tidak tersedia tanpa menutup koneksi.

Untuk aplikasi yang memelihara kumpulan koneksinya sendiri, melalui RDS Proxy berarti sebagian besar koneksi tetap hidup selama kegagalan atau gangguan lainnya. Hanya koneksi yang berada di tengah transaksi atau SQL pernyataan yang dibatalkan. RDS Proxy segera menerima koneksi baru. Ketika penulis database tidak tersedia, RDS Proxy mengantri permintaan yang masuk.

Untuk aplikasi yang tidak memelihara kumpulan koneksi mereka sendiri, RDS Proxy menawarkan tingkat koneksi yang lebih cepat dan koneksi yang lebih terbuka. Hal ini menyebabkan overhead yang mahal dikarenakan seringnya koneksi ulang dari basis data. Ia melakukannya dengan menggunakan kembali koneksi database yang dipelihara di kumpulan koneksi RDS Proxy. Pendekatan ini sangat penting untuk TLS koneksi, di mana biaya penyiapan signifikan.

## Transaksi

Semua pernyataan dalam satu transaksi selalu menggunakan koneksi basis data acuan yang sama. Koneksi akan dapat digunakan oleh sesi yang berbeda saat transaksi berakhir. Penggunaan transaksi sebagai unit granularitas memiliki konsekuensi sebagai berikut:

- Penggunaan kembali koneksi dapat terjadi setelah setiap pernyataan individu ketika pengaturan Saya diaktifkan.
- Sebaliknya, jika pengaturan `autocommit` dinonaktifkan, pernyataan pertama yang Anda terbitkan dalam sesi akan memulai transaksi baru. Misalnya, Anda memasukkan urutan`SELECT`,, `INSERT`UPDATE, dan pernyataan bahasa manipulasi data lainnya (DML). Dalam hal ini, penggunaan kembali koneksi tidak terjadi hingga Anda mengeluarkan `COMMIT`, `ROLLBACK`, atau mengakhiri transaksi.
- Memasukkan pernyataan bahasa definisi data (DDL) menyebabkan transaksi berakhir setelah pernyataan itu selesai.

RDSProxy mendeteksi ketika transaksi berakhir melalui protokol jaringan yang digunakan oleh aplikasi klien database. Deteksi transaksi tidak bergantung pada kata kunci seperti `COMMIT` atau `ROLLBACK` muncul dalam teks SQL pernyataan.

Dalam beberapa kasus, RDS Proxy mungkin mendeteksi permintaan database yang membuatnya tidak praktis untuk memindahkan sesi Anda ke koneksi yang berbeda. Dalam kasus ini, multiplexing dinonaktifkan untuk koneksi tersebut hingga sesi Anda berakhir. Aturan yang sama berlaku jika RDS Proxy tidak dapat memastikan bahwa multiplexing praktis untuk sesi tersebut. Operasi ini disebut pinning. Untuk mendeteksi dan meminimalkan pinning, lihat Menghindari menyematkan Proxy RDS.

# Memulai dengan RDS Proxy

Gunakan informasi di halaman berikut untuk mengatur Proksi Amazon RDS untuk Aurora dan mengelola serta mengatur opsi keamanan terkait. Opsi keamanan mengontrol siapa yang dapat mengakses setiap proxy dan bagaimana setiap proxy terhubung ke instans DB.

Jika Anda baru mengenal RDS Proxy, sebaiknya ikuti halaman sesuai urutan yang kami sajikan.

Topik

- Menyiapkan prasyarat jaringan untuk RDS Proxy
- Menyiapkan kredensi database AWS Secrets Manager untuk RDS Proxy
- Mengkonfigurasi otentikasi IAM untuk RDS Proxy
- Membuat proxy untuk
- Melihat proxy
- Terhubung ke basis data melalui Proksi RDS

## Menyiapkan prasyarat jaringan untuk RDS Proxy

Menggunakan Proksi RDS mengharuskan Anda memiliki cloud privat virtual (VPC) umum antara klaster DB Aurora dan Proksi RDS. VPC ini harus memiliki minimal dua subnet yang berada di Zona Ketersediaan yang berbeda. Akun Anda dapat memiliki subnet ini atau membagikannya dengan akun lain. Untuk informasi tentang berbagi VPC, lihat Bekerja dengan berbagi. VPCs

Sumber daya aplikasi klien Anda seperti Amazon EC2, Lambda, atau Amazon ECS dapat berada di VPC yang sama dengan proxy. Atau sumber daya ini bisa berada di VPC terpisah dari proksi. Jika Anda berhasil terhubung ke klaster DB Aurora apa pun, berarti Anda sudah memiliki sumber daya jaringan yang diperlukan.

Topik

- Mendapatkan informasi tentang subnet Anda
- Perencanaan untuk kapasitas alamat IP

## Mendapatkan informasi tentang subnet Anda

Jika Anda baru memulai dengan Aurora, Anda dapat mempelajari dasar-dasar menghubungkan ke database dengan mengikuti prosedur di. Menyiapkan lingkungan Anda untuk Amazon Aurora Anda juga dapat mengikuti tutorial dalam Memulai dengan Amazon Aurora.

Untuk membuat proxy, Anda harus menyediakan subnet dan VPC tempat proxy beroperasi di dalamnya. Contoh Linux berikut menunjukkan AWS CLI perintah yang memeriksa VPCs dan subnet yang dimiliki oleh Anda Akun AWS. Secara khusus, Anda meneruskan subnet IDs sebagai parameter saat Anda membuat proxy menggunakan CLI.

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

Contoh Linux berikut menunjukkan AWS CLI perintah untuk menentukan subnet yang IDs sesuai dengan instance cluster.

Untuk klaster Aurora, cari ID untuk salah satu instans DB terkait terlebih dahulu. Anda dapat mengekstrak subnet yang IDs digunakan oleh instans DB itu. Untuk melakukannya, periksa kolom bertingkat dalam atribut `DBSubnetGroup` dan `Subnets` dan di output deskripsi untuk instans DB. Anda menentukan beberapa atau semua subnet tersebut IDs saat menyiapkan proxy untuk server database tersebut.

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]||[0]||[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

Setelah menemukan ID instans DB, periksa VPC terkait untuk menemukan subnetnya. Contoh Linux berikut menunjukkan caranya.

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]||[0]||[0]||[Subnets]||[0]||[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

## Perencanaan untuk kapasitas alamat IP

Proksi RDS secara otomatis menyesuaikan kapasitasnya sesuai kebutuhan berdasarkan ukuran dan jumlah instans DB yang didaftarkan di proksi. Operasi tertentu mungkin juga memerlukan kapasitas proksi tambahan seperti menambah ukuran basis data yang terdaftar atau operasi pemeliharaan Proksi RDS internal. Selama operasi ini, proksi Anda mungkin memerlukan lebih banyak alamat IP untuk menyediakan kapasitas tambahan. Alamat tambahan ini memungkinkan proksi Anda diskalakan tanpa memengaruhi beban kerja Anda. Kurangnya alamat IP kosong di subnet Anda mencegah kenaikan skala proksi. Hal ini dapat menyebabkan latensi kueri yang lebih tinggi atau kegagalan koneksi klien. RDS memberi tahu Anda melalui peristiwa `RDS-EVENT-0243` ketika tidak ada cukup alamat IP kosong di subnet Anda. Untuk informasi tentang peristiwa ini, lihat Bekerja dengan acara RDS Proxy.

Cadangan jumlah minimum alamat IP gratis berikut di subnet Anda untuk proxy Anda, berdasarkan ukuran kelas instans DB.

| Kelas instans DB | Alamat IP kosong minimum |
|---|---|
| db.*.xlarge atau lebih kecil | 10 |
| db.*.2xlarge | 15 |
| db.*.4xlarge | 25 |
| db.*.8xlarge | 45 |
| db.*.12xlarge | 60 |
| db.*.16xlarge | 75 |
| db.*.24xlarge | 110 |

Jumlah alamat IP yang direkomendasikan ini adalah perkiraan untuk proksi dengan titik akhir default saja. Proksi dengan titik akhir tambahan atau replika baca mungkin memerlukan lebih banyak alamat IP kosong. Untuk setiap titik akhir tambahan, sebaiknya Anda mencadangkan tiga alamat IP lagi. Untuk setiap replika baca, sebaiknya Anda mencadangkan alamat IP tambahan seperti yang ditentukan dalam tabel berdasarkan ukuran replika baca tersebut.

> **Note**
>
> RDS Proxy tidak menggunakan lebih dari 215 alamat IP dalam VPC.

Misalnya, Anda ingin memperkirakan alamat IP yang diperlukan untuk proksi yang terkait dengan klaster DB Aurora.

Dalam kasus ini, asumsikan hal berikut:

- Klaster DB Aurora Anda memiliki 1 instans penulis berukuran db.r5.8xlarge dan 1 instans pembaca berukuran db.r5.2xlarge.
- Proksi yang disematkan ke klaster DB ini memiliki titik akhir default dan 1 titik akhir kustom dengan peran hanya-baca.

Dalam hal ini, proksi memerlukan kira-kira 63 alamat IP kosong (45 untuk instans penulis, 15 untuk instans pembaca, dan 3 untuk titik akhir kustom tambahan).

## Menyiapkan kredensi database AWS Secrets Manager untuk RDS Proxy

RDS Proxy di Amazon RDS digunakan AWS Secrets Manager untuk menyimpan dan mengelola kredenal database dengan aman. Alih-alih menyematkan kredensi dalam aplikasi Anda, Anda mengaitkan proxy dengan rahasia Secrets Manager yang berisi detail otentikasi yang diperlukan. Anda dapat membuat rahasia Secrets Manager terpisah untuk setiap akun pengguna basis data yang terhubung ke klaster DB Aurora.

Topik


- Membuat rahasia untuk digunakan dengan RDS Proxy


### Membuat rahasia untuk digunakan dengan RDS Proxy

Sebelum Anda membuat proxy, Anda harus terlebih dahulu membuat setidaknya satu rahasia yang menyimpan kredensi database Anda.

Konsol

Untuk membuat rahasia

1. Buka konsol Secrets Manager di https://console.aws.amazon.com/secretsmanager/.
2. Pilih Simpan rahasia baru.
3. Pilih Kredensial untuk database Amazon RDS.
4. Masukkan nama pengguna dan kata sandi. Kredensial yang Anda masukkan harus cocok dengan kredensi pengguna database yang ada di database RDS terkait. RDS Proxy menggunakan kredensil ini untuk mengautentikasi dan membangun koneksi ke database atas nama aplikasi.

   Jika ada ketidakcocokan, Anda dapat memperbarui rahasia agar sesuai dengan kata sandi database. Sampai Anda memperbarui rahasia, upaya untuk terhubung melalui proxy menggunakan rahasia itu gagal, tetapi koneksi menggunakan rahasia valid lainnya masih berfungsi.

> **Note**
>
> Untuk RDS untuk SQL Server, RDS Proxy memerlukan rahasia case-sensitive di Secrets Manager, terlepas dari pengaturan pengumpulan instans DB. Jika aplikasi Anda mengizinkan nama pengguna dengan kapitalisasi yang berbeda, seperti “Admin” dan “admin,” Anda harus membuat rahasia terpisah untuk masing-masing. RDS Proxy tidak mendukung otentikasi nama pengguna yang tidak peka huruf besar/kecil antara klien dan proxy.
> Untuk informasi selengkapnya tentang kolasi di SQL Server, lihat dokumentasi Microsoft SQL Server.

5. Untuk Database, pilih database Amazon RDS yang akan diakses rahasia.
6. Isi pengaturan lain untuk rahasia, lalu pilih Store. Untuk petunjuk komprehensif, lihat Membuat AWS Secrets Manager rahasia di Panduan AWS Secrets Manager Pengguna.

AWS CLI

Saat Anda membuat proxy melalui AWS CLI, Anda menentukan Amazon Resource Names (ARNs) dari rahasia terkait. Anda melakukannya untuk semua akun pengguna DB yang dapat diakses proksi. Dalam AWS Management Console, Anda memilih rahasia dengan nama deskriptif mereka.

- Untuk membuat rahasia Secrets Manager untuk digunakan dengan RDS Proxy, gunakan perintah create-secret:

```
aws secretsmanager create-secret \
  --name "secret_name" \
  --description "secret_description" \
  --region region_name \
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- Anda juga dapat membuat kunci khusus untuk mengenkripsi rahasia Secrets Manager Anda. Perintah berikut menciptakan sebuah contoh kunci.

```
aws kms create-key --description "test-key" --policy '{
  "Id":"kms-policy",
  "Version":"2012-10-17",
  "Statement":
    [
```

```
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
        "Resource":"*"
      }
    ]
}'
```

Misalnya, perintah berikut membuat rahasia Secrets Manager untuk dua pengguna database:

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

Untuk membuat rahasia ini dienkripsi dengan AWS KMS kunci kustom Anda, gunakan perintah berikut:

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

Untuk melihat rahasia yang dimiliki oleh AWS akun Anda, gunakan perintah daftar-rahasia:

```
aws secretsmanager list-secrets
```

Saat Anda membuat proxy menggunakan CLI, Anda meneruskan Amazon Resource Names (ARNs) dari satu atau beberapa rahasia ke parameter. `--auth` Contoh berikut menunjukkan cara menyiapkan laporan hanya dengan nama dan ARN dari setiap rahasia yang dimiliki oleh akun Anda AWS . Contoh ini menggunakan `--output table` parameter yang tersedia di AWS CLI versi 2. Jika Anda menggunakan AWS CLI versi 1, gunakan `--output text` sebagai gantinya.

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

Untuk mengonfirmasi bahwa rahasia berisi kredensil yang benar dalam format yang tepat, gunakan perintah. get-secret-value Ganti *your_secret_name* dengan nama pendek rahasia atau ARN.

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

Outputnya berisi garis dengan nilai yang dikodekan JSON mirip dengan yang berikut ini:

```
...
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
...
```

# Mengkonfigurasi otentikasi IAM untuk RDS Proxy

Untuk menyiapkan autentikasi AWS Identity and Access Management (IAM) untuk Proxy RDS di Amazon RDS, buat dan konfigurasikan kebijakan IAM yang memberikan izin yang diperlukan. RDS Proxy menggunakan AWS Secrets Manager untuk mengelola kredensional database dengan aman, yang memungkinkan aplikasi untuk mengautentikasi melalui proxy tanpa langsung menangani kredensi.

Topik ini menyediakan langkah-langkah untuk mengonfigurasi autentikasi IAM untuk RDS Proxy, termasuk membuat kebijakan IAM yang diperlukan dan melampirkannya ke peran IAM.

> **Tip**
>
> Prosedur ini hanya diperlukan jika Anda ingin membuat peran IAM Anda sendiri. Jika tidak, RDS dapat secara otomatis membuat peran yang diperlukan saat Anda mengatur proxy, sehingga Anda dapat melewati langkah-langkah ini.

## Prasyarat

Sebelum Anda mengatur autentikasi IAM untuk RDS Proxy, pastikan Anda memiliki yang berikut:

- AWS Secrets ManagerSetidaknya satu rahasia tersimpan yang berisi kredensi database. Untuk instruksi untuk membuat rahasia, lihatthe section called “Menyiapkan kredensial basis data”.
- Izin IAM — Peran IAM atau pengguna dengan izin untuk membuat dan mengelola kebijakan, peran, dan rahasia IAM. AWS Secrets Manager

## Membuat kebijakan IAM untuk akses Secrets Manager

Untuk mengizinkan RDS Proxy mengambil kredenal database dari Secrets Manager, buat peran IAM dengan kebijakan yang memberikan izin yang diperlukan.

Konsol

Untuk membuat peran untuk mengakses rahasia Anda untuk digunakan dengan proxy Anda

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Buat kebijakan izin untuk peran tersebut. Untuk langkah umum, lihat Membuat kebijakan IAM (konsol).

   Tempelkan kebijakan ini ke editor JSON dan buat perubahan berikut:

   - Ganti ID akun Anda sendiri.
   - Gantikan `us-east-2` dengan Wilayah tempat proxy akan berada.
   - Ganti nama rahasia dengan yang Anda buat. Untuk informasi selengkapnya, lihat Menentukan kunci KMS dalam pernyataan kebijakan IAM.
   - Ganti ID kunci KMS dengan yang Anda gunakan untuk mengenkripsi rahasia Secrets Manager, baik kunci default atau kunci Anda sendiri.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-
east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-
east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
```

```
                    "kms:ViaService": "secretsmanager.us-
east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. Buat peran dan lampirkan kebijakan izin padanya. Untuk langkah umum, lihat Membuat peran untuk mendelegasikan izin ke layanan. AWS

   Untuk jenis entitas Tepercaya, pilih AWS layanan. Di bawah Kasus penggunaan, pilih RDS dan pilih RDS - Tambahkan Peran ke Database untuk kasus penggunaan.

4. Untuk kebijakan Izin, pilih kebijakan yang Anda buat.

5. Untuk Pilih entitas tepercaya, masukkan kebijakan kepercayaan berikut untuk peran tersebut:

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

Untuk membuat peran menggunakan AWS CLI, kirim permintaan berikut:

```
aws iam create-role \
  --role-name my_role_name \
```

```
  --assume-role-policy-document '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

Kemudian, lampirkan kebijakan ke peran:

```
aws iam put-role-policy \
  --role-name my_role_name \
  --policy-name secret_reader_policy \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}'
```

Dengan peran dan izin IAM yang dikonfigurasi, Anda sekarang dapat membuat proxy dan mengaitkannya dengan peran ini. Hal ini memungkinkan proxy untuk mengambil kredensi database dengan aman dari AWS Secrets Manager dan mengaktifkan autentikasi IAM untuk aplikasi Anda. Untuk petunjuk, lihat the section called “Membuat proxy”.

# Membuat proxy untuk

Anda dapat menggunakan Amazon RDS Proxy untuk meningkatkan skalabilitas, ketersediaan, dan keamanan aplikasi database Anda dengan menggabungkan koneksi dan mengelola failover database secara lebih efisien. Topik ini memandu Anda melalui pembuatan proxy. Sebelum memulai, pastikan database Anda memenuhi prasyarat yang diperlukan, termasuk izin IAM dan konfigurasi VPC.

Anda dapat mengaitkan sebuah proksi dengan klaster DB Aurora MySQL atau Aurora PostgreSQL.

Konsol

Untuk membuat proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Pilih Buat proksi.
4. Konfigurasikan pengaturan berikut untuk proxy Anda.

| Pengaturan | Deskripsi |
| --- | --- |
| Keluarga mesin | Protokol jaringan database proxy mengenali ketika menafsirkan lalu lintas jaringan ke dan dari database.<br><br>**Note**<br>Untuk menggunakan Aurora PostgreSQL, pastikan untuk menyimpan database dalam instance Anda. `postgres` Lihat Pemecahan masalah database yang dihapus `postgres`. |
| Pengidentifikasi proxy | Nama yang unik dalam ID AWS akun Anda dan AWS Wilayah saat ini. |
| Batas waktu koneksi klien idle | Proxy menutup koneksi klien jika tetap menganggur selama periode tertentu. Secara default, ini adalah 1.800 detik (30 menit). Koneksi tidak digunakan saat aplikasi tidak mengirimk |

| Pengaturan | Deskripsi |
| --- | --- |
| | an permintaan baru dalam waktu yang ditentukan setelah menyelesaikan permintaan sebelumnya. Proxy menjaga koneksi database yang mendasarinya tetap terbuka dan mengembal ikannya ke kumpulan koneksi, membuatnya tersedia untuk koneksi klien baru.<br><br>Untuk secara proaktif menghapus koneksi basi, kurangi batas waktu koneksi klien idle. Untuk meminimalkan biaya koneksi selama lonjakan beban kerja, tingkatkan batas waktu. |
| Basis Data | Aurora DB cluster untuk mengakses melalui proxy ini. Daftar ini hanya mencakup instans dan klaster DB dengan mesin basis data, versi mesin, dan pengaturan lainnya yang kompatibe l. Jika daftar kosong, buat instans atau klaster DB baru yang kompatibel dengan Proksi RDS. Untuk melakukannya, ikuti prosedurnya di Membuat klaster DB Amazon Aurora. Kemudian, coba buat proxy lagi. |
| Koneksi kolam koneksi maksimum | Nilai antara 1 dan 100 untuk menentukan persentase `max_connections` batas yang dapat digunakan RDS Proxy. Jika Anda hanya bermaksud menggunakan satu proxy dengan instans atau cluster DB ini, tetapkan nilai ini ke 100. Untuk informasi selengkapnya tentang cara Proxy RDS menggunakan setelan ini, lihatthe section called “MaxConnectionsPercent”. |

| Pengaturan | Deskripsi |
| --- | --- |
| Filter penyematan sesi | Mencegah Proxy RDS menyematkan status sesi tertentu yang terdeteksi, yang melewati langkah-langkah keamanan default untuk koneksi multiplexing. Saat ini, PostgreSQL tidak mendukung pengaturan ini, dan satu-satunya opsi yang tersedia adalah. `EXCLUDE_VARIABLE_SETS` Mengaktifkannya dapat menyebabkan variabel sesi dari satu koneksi memengaruhi yang lain, yang menyebabkan kesalahan atau masalah kebenaran jika kueri bergantung pada variabel sesi yang ditetapkan di luar transaksi saat ini. Gunakan opsi ini hanya setelah mengonfirmasi bahwa aplikasi Anda dapat berbagi koneksi database dengan aman.<br><br>Pola-pola berikut dianggap aman:<br><br>• `SET`pernyataan di mana tidak ada perubahan pada nilai variabel sesi efektif. Dengan kata lain, tidak ada perubahan pada variabel sesi.<br>• Anda mengubah nilai variabel sesi dan mengeksekusi pernyataan dalam transaksi yang sama.<br><br>Untuk informasi selengkapnya, lihat Menghindari menyematkan Proxy RDS. |
| Koneksi meminjam batas waktu | Jika Anda mengharapkan proxy untuk menggunakan semua koneksi database yang tersedia, atur waktu tunggu sebelum mengembalikan kesalahan batas waktu. Anda dapat menentukan hingga lima menit. Pengaturan ini hanya berlaku ketika proxy telah mencapai jumlah maksimum koneksi dan semua sedang digunakan. |

| Pengaturan | Deskripsi |
| --- | --- |
| Kueri inisialisasi | (Opsional) Tambahkan kueri inisialisasi, atau ubah yang sekarang. Anda dapat menentukan satu atau beberapa pernyataan SQL untuk proksi yang akan dijalankan saat membuka setiap koneksi basis data baru. Pengaturan ini biasanya digunakan dengan SET pernyataan untuk memastikan bahwa setiap koneksi memiliki pengaturan yang identik. Pastikan kueri yang Anda tambahkan valid. Untuk memasukkan beberapa variabel dalam satu SET pernyataan, gunakan pemisah koma. Misalnya:<br>`SET variable1 =value1, variable2 =value2`<br>Untuk beberapa pernyataan, gunakan titik koma sebagai pemisah.<br>**Important**<br>Karena Anda dapat mengakses kueri inisialisasi sebagai bagian dari konfigurasi grup target, itu tidak dilindungi oleh otentikasi atau metode kriptografi. Siapa pun yang memiliki akses untuk melihat atau mengelola konfigurasi grup target proxy Anda dapat melihat kueri inisialisasi. Anda tidak boleh menambahkan data sensitif, seperti kata sandi atau kunci enkripsi berumur panjang, ke opsi ini. |
| AWS Identity and Access Management Peran (IAM) | Peran IAM dengan izin untuk mengakses rahasia Secrets Manager, yang mewakili kredensil untuk akun pengguna database yang dapat digunakan proxy. Atau, Anda dapat membuat peran IAM baru dari. AWS Management Console |
| Rahasia Secrets Manager | Pilih setidaknya satu rahasia Secrets Manager terpisah yang berisi kredensial pengguna basis data yang memungkinkan proksi mengakses klaster DB Aurora. |

| Pengaturan | Deskripsi |
| --- | --- |
| Jenis otentikasi klien | Jenis otentikasi yang digunakan proxy untuk koneksi dari klien. Pilihan Anda berlaku untuk semua rahasia Secrets Manager yang Anda kaitkan dengan proksi ini. Jika Anda perlu menentukan jenis otentikasi klien yang berbeda untuk setiap rahasia, buat proxy Anda dengan menggunakan AWS CLI atau API sebagai gantinya. |
| Autentikasi IAM | Apakah memerlukan atau melarang autentikasi IAM untuk koneksi ke proxy Anda. Pilihan Anda berlaku untuk semua rahasia Secrets Manager yang Anda kaitkan dengan proksi ini. Jika Anda perlu menentukan otentikasi IAM yang berbeda untuk setiap rahasia, buat proxy Anda dengan menggunakan AWS CLI atau API sebagai gantinya. |
| Memerlukan Keamanan Lapisan Transportasi | Menerapkan TLS/SSL untuk semua koneksi klien. Proxy menggunakan pengaturan enkripsi yang sama untuk koneksi ke database yang mendasarinya, apakah koneksi klien dienkripsi atau tidak terenkripsi. |
| Subnet | Bidang ini diisi sebelumnya dengan semua subnet yang terkait dengan VPC Anda. Anda dapat menghapus subnet apa pun yang tidak diperlukan untuk proxy, tetapi Anda harus meninggal kan setidaknya dua subnet. |

| Pengaturan | Deskripsi |
| --- | --- |
| Grup keamanan VPC | Pilih grup keamanan VPC yang ada atau buat yang baru dari. AWS Management Console Konfigurasikan aturan masuk untuk memungkinkan aplikasi Anda mengakses proxy dan aturan keluar untuk mengizinkan lalu lintas dari target database Anda.<br><br>**Note**<br>Grup keamanan harus mengizinkan koneksi dari proxy ke database. Ini berfungsi baik untuk masuknya dari aplikasi Anda ke proxy dan keluar dari proxy ke database. Misalnya, jika Anda menggunakan grup keamanan yang sama untuk database dan proxy, pastikan sumber daya dalam grup keamanan tersebut dapat berkomunikasi satu sama lain.<br>Saat Anda menggunakan VPC bersama, hindari menggunakan grup keamanan default untuk VPC atau yang terkait dengan akun lain. Sebagai gantinya, pilih grup keamanan milik akun Anda. Jika tidak ada, buat satu. Untuk informasi selengkapnya, lihat Bekerja dengan berbagi VPCs.<br><br>RDS menyebarkan proxy di beberapa Availability Zone untuk memastikan ketersediaan tinggi. Untuk mengaktifkan komunikasi lintas AZ, daftar kontrol akses jaringan (ACL) untuk subnet proxy Anda harus mengizinkan jalan keluar pada port engine dan masuknya pada semua port. Untuk informasi selengkapnya tentang jaringan ACLs, lihat Mengontrol lalu lintas ke subnet menggunakan jaringan ACLs. Jika ACL jaringan untuk proksi dan target Anda identik, Anda harus menambahkan aturan masuknya protokol TCP tempat Sumber diatur ke CIDR VPC. Anda juga harus menambahkan aturan keluar protokol TCP khusus port engine di mana Tujuan diatur ke CIDR VPC. |

| Pengaturan | Deskripsi |
| --- | --- |
| Aktifkan logging yang disempurnakan | Aktifkan pengaturan ini untuk memecahkan masalah kompatibi litas proxy atau masalah kinerja. Saat diaktifkan, Proxy RDS mencatat informasi kinerja terperinci untuk membantu Anda men-debug perilaku SQL atau kinerja dan skalabilitas koneksi proxy.<br><br>Hanya aktifkan pengaturan ini untuk debugging dan pastikan langkah-langkah keamanan yang tepat ada untuk melindungi informasi sensitif di log. Untuk meminimalkan overhead, RDS Proxy secara otomatis menonaktifkan pengaturan ini 24 jam setelah aktivasi. Gunakan sementara untuk memecahkan masalah tertentu. |

5. Pilih Buat proksi.

AWS CLI

Untuk membuat proxy dengan menggunakan AWS CLI, panggil create-db-proxyperintah dengan parameter yang diperlukan berikut:

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--auth`
- `--vpc-subnet-ids`

Nilai `--engine-family` ini bersifat peka huruf besar-kecil.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
```

```
    --auth ProxyAuthenticationConfig_JSON_string \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--tags comma_separated_list]
```

Untuk Windows:

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --auth ProxyAuthenticationConfig_JSON_string ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--tags comma_separated_list]
```

Berikut ini adalah contoh nilai JSON untuk opsi `--auth`. Contoh ini menerapkan jenis otentikasi klien yang berbeda untuk setiap rahasia.

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"
```

```
  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

> **Tip**
>
> Jika Anda belum mengetahui subnet yang akan digunakan IDs untuk `--vpc-subnet-ids` parameter, lihat Menyiapkan prasyarat jaringan untuk RDS Proxy contoh cara menemukannya.

> **Note**
>
> Grup keamanan ini harus mengizinkan akses ke basis data yang terhubung ke proksi. Grup keamanan yang sama digunakan sebagai jalur masuk dari aplikasi ke proksi, dan jalur keluar dari proksi ke basis data. Misalnya, anggap saja Anda menggunakan grup keamanan yang sama untuk basis data dan proksi Anda. Dalam kasus ini, pastikan sumber daya dalam grup keamanan tersebut dapat berkomunikasi dengan sumber daya lain dalam grup keamanan yang sama.
>
> Saat menggunakan VPC bersama, Anda tidak dapat menggunakan grup keamanan default untuk VPC, atau grup keamanan milik akun lain. Pilih grup keamanan milik akun Anda. Jika belum ada, buat. Untuk informasi selengkapnya tentang batasan ini, lihat Bekerja dengan berbagi VPCs.

Untuk membuat asosiasi yang tepat untuk proxy, Anda juga menggunakan register-db-proxy-targetsperintah. Tentukan nama grup target `default`. Proksi RDS secara otomatis membuat grup target dengan nama ini saat Anda membuat setiap proksi.

```
aws rds register-db-proxy-targets
```

```
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

API RDS

Untuk membuat proxy RDS, panggil Create operasi Amazon RDS API. DBProxy Anda melewati parameter dengan struktur AuthConfigdata.

Proksi RDS secara otomatis membuat grup target bernama `default` saat Anda membuat setiap proksi. Anda mengaitkan Aurora DB cluster dengan grup target dengan memanggil fungsi DBProxy Register Targets.

## Melihat proxy

Setelah Anda membuat satu atau beberapa proxy RDS, Anda dapat melihat dan mengelolanya di AWS Management Console, API AWS CLI, atau RDS. Anda dapat meninjau detail konfigurasi mereka, memantau kinerja, dan menentukan proxy mana yang akan dimodifikasi atau dihapus sesuai kebutuhan.

Untuk mengaktifkan aplikasi database untuk merutekan lalu lintas melalui proxy, Anda harus menentukan titik akhir proxy dalam string koneksi.

Konsol

Untuk melihat proxy di konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Pilih nama proxy untuk melihat detailnya.
4. Pada halaman detail, bagian Grup target menunjukkan bagaimana proxy ditautkan ke cluster Aurora tertentu. Anda dapat menavigasi ke halaman grup target default untuk tampilan yang lebih dalam dari asosiasi ini, termasuk pengaturan konfigurasi yang ditentukan selama pembuatan proxy. Pengaturan ini mencakup persentase koneksi maksimum, batas waktu pinjam koneksi, keluarga mesin, dan filter penyematan sesi.

CLI

Untuk melihat proxy Anda menggunakan CLI, gunakan perintah. describe-db-proxies Secara default, permintaan mengembalikan semua proxy yang dimiliki oleh akun Anda AWS . Untuk melihat detail dari satu proksi, masukkan namanya dengan parameter `--db-proxy-name`.

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

Untuk melihat informasi lain yang terkait dengan proxy, gunakan perintah berikut.

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

Gunakan urutan perintah berikut untuk melihat detail selengkapnya tentang hal-hal yang terkait dengan proksi:

1. Untuk mendapatkan daftar proxy, jalankan. describe-db-proxies
2. Untuk menampilkan parameter koneksi seperti persentase maksimum koneksi yang dapat digunakan proxy, jalankan describe-db-proxy-target-groups`--db-proxy-name`. Gunakan nama proksi sebagai nilai parameter.
3. Untuk melihat detail cluster Aurora  yang terkait dengan grup target yang dikembalikan, jalankan. describe-db-proxy-targets

API RDS

Untuk melihat proxy Anda menggunakan RDS API, gunakan operasi Deskripsikan. DBProxies Operasi ini akan menampilkan nilai dari jenis data DBProxy.

Untuk melihat detail pengaturan koneksi untuk proxy, gunakan pengidentifikasi proxy dari nilai pengembalian ini dengan DBProxy TargetGroups operasi Deskripsikan. Operasi ini akan menampilkan nilai dari jenis data DBProxyTargetGroup.

Untuk melihat instans RDS atau cluster Aurora DB yang terkait dengan proxy, gunakan operasi DBProxyDeskripsikan Target. Ia mengembalikan nilai-nilai tipe data DBProxyTarget.

# Terhubung ke basis data melalui Proksi RDS

Anda terhubung ke cluster Aurora DB atau cluster yang menggunakan Aurora Serverless v2 melalui proxy secara umum dengan cara yang sama seperti Anda terhubung langsung ke database. Perbedaan utamanya adalah Anda menentukan titik akhir proksi, bukan titik akhir klaster. Secara default, semua koneksi proksi memiliki kemampuan baca/tulis dan menggunakan instans penulis. Jika Anda biasanya menggunakan titik akhir pembaca untuk koneksi hanya-baca, Anda dapat membuat titik akhir hanya-baca tambahan untuk proksi. Anda dapat menggunakan titik akhir tersebut dengan cara yang sama. Untuk informasi selengkapnya, lihat Ikhtisar titik akhir proksi.

Topik

- Terhubung ke sebuah proksi menggunakan autentikasi native
- Terhubung ke sebuah proksi menggunakan autentikasi IAM
- Pertimbangan untuk terhubung ke sebuah proksi dengan PostgreSQL

## Terhubung ke sebuah proksi menggunakan autentikasi native

Gunakan langkah berikut untuk terhubung ke proksi menggunakan autentikasi native:

1. Temukan titik akhir proksi. Di AWS Management Console, Anda dapat menemukan titik akhir pada halaman detail untuk proxy yang sesuai. Dengan AWS CLI, Anda dapat menggunakan describe-db-proxiesperintah. Contoh berikut menunjukkan caranya.

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
```

```
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. Tentukan titik akhir sebagai parameter host dalam string koneksi untuk aplikasi klien Anda. Misalnya, tentukan titik akhir proksi sebagai nilai untuk opsi `mysql -h` atau opsi `psql -h`.
3. Masukkan nama dan kata sandi pengguna basis data yang sama seperti biasanya.

## Terhubung ke sebuah proksi menggunakan autentikasi IAM

Saat Anda menggunakan autentikasi IAM dengan Proksi RDS, siapkan pengguna basis data Anda untuk melakukan autentikasi dengan nama dan kata sandi pengguna reguler. Autentikasi IAM berlaku untuk Proksi RDS yang mengambil kredensial nama dan kata sandi pengguna dari Secrets Manager. Koneksi dari Proksi RDS ke basis data acuan tidak melewati IAM.

Untuk terhubung ke Proksi RDS menggunakan autentikasi IAM, gunakan prosedur koneksi umum yang sama seperti autentikasi IAM dengan klaster DB Aurora. Untuk informasi umum tentang cara menggunakan IAM, lihat Keamanan dalam Amazon Aurora.

Perbedaan utama dalam penggunaan IAM untuk Proksi RDS meliputi:

- Anda tidak dapat mengonfigurasi setiap pengguna basis data dengan plugin otorisasi. Pengguna basis data masih memiliki nama dan kata sandi pengguna reguler dalam basis data. Anda dapat menyiapkan rahasia Secrets Manager yang berisi nama dan kata sandi pengguna ini, dan mengotorisasi Proksi RDS untuk mengambil kredensial dari Secrets Manager.

  Autentikasi IAM berlaku untuk koneksi antara program klien Anda dan proksi. Proksi kemudian melakukan autentikasi ke basis data menggunakan kredensial nama dan kata sandi pengguna yang diambil dari Secrets Manager.
- Anda menentukan titik akhir proksi, bukan instans, klaster, atau titik akhir pembaca. Untuk detail tentang titik akhir proksi, lihat Menghubungkan ke klaster DB menggunakan autentikasi IAM.

- Dalam kasus autentikasi IAM basis data langsung, Anda secara selektif memilih pengguna basis data dan mengonfigurasinya untuk diidentifikasi dengan plugin autentikasi khusus. Anda kemudian dapat terhubung ke pengguna tersebut menggunakan autentikasi IAM.

  Dalam kasus penggunaan proksi, Anda memberi proksi Rahasia yang berisi nama pengguna dan kata sandi pengguna tertentu (autentikasi native). Anda kemudian terhubung ke proksi menggunakan autentikasi IAM. Di sini, Anda melakukannya dengan membuat token autentikasi dengan titik akhir proksi, bukan titik akhir basis data. Anda juga menggunakan nama pengguna yang cocok dengan salah satu nama pengguna untuk rahasia yang Anda berikan.
- Pastikan Anda menggunakan Keamanan Lapisan Pengangkutan (TLS)/Lapisan Soket Aman (SSL) saat terhubung ke sebuah proksi menggunakan autentikasi IAM.

Anda dapat memberi pengguna tertentu akses ke proksi dengan mengubah kebijakan IAM. Berikut contohnya.

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

## Pertimbangan untuk terhubung ke sebuah proksi dengan PostgreSQL

Jika Anda membuat pengguna database PostgreSQL baru untuk menghubungkan ke RDS Proxy, pastikan bahwa Anda memberikan hak istimewa pengguna pada database. `CONNECT` Tanpa ini, pengguna tidak dapat membuat koneksi. Untuk informasi selengkapnya, lihat the section called “Menambahkan pengguna database baru ke database Postgre saat menggunakan SQL Proxy RDS”.

Ketika klien memulai koneksi ke database PostgreSQL, ia mengirimkan pesan startup. Pesan ini berisi pasangan nama parameter dan string nilai. Untuk detailnya, lihat `StartupMessage` dalam PostgreSQL message formats dalam dokumentasi PostgreSQL.

Saat Anda terhubung melalui proxy RDS, pesan startup dapat menyertakan parameter berikut yang saat ini dikenali:

- `user`
- `database`

Pesan startup juga bisa menyertakan parameter runtime tambahan berikut:

- `application_name`

- client_encoding
- DateStyle
- TimeZone
- extra_float_digits
- search_path

Untuk informasi selengkapnya tentang pesan PostgreSQL, lihat Frontend/Backend protocol dalam dokumentasi PostgreSQL.

Untuk PostgreSQL, jika Anda menggunakan JDBC, sebaiknya lakukan tindakan berikut untuk menghindari penyematan:

- Atur parameter koneksi JDBC `assumeMinServerVersion` ke setidaknya `9.0` untuk menghindari penyematan. Tindakan ini dapat mencegah driver JDBC melakukan perjalanan roundtrip ekstra selama startup koneksi saat menjalankan `SET extra_float_digits = 3`.
- Atur parameter koneksi JDBC `ApplicationName` ke *`any/your-application-name`* untuk menghindari penyematan. Tindakan ini dapat mencegah driver JDBC melakukan roundtrip ekstra selama startup koneksi saat menjalankan `SET application_name = "PostgreSQL JDBC Driver"`. Perhatikan bahwa parameter JDBC adalah `ApplicationName`, tetapi parameter PostgreSQL `StartupMessage` adalah `application_name`.

Untuk informasi selengkapnya, lihat Menghindari menyematkan Proxy RDS. Untuk informasi selengkapnya tentang cara terhubung menggunakan JDBC, lihat Connecting to the database dalam dokumentasi PostgreSQL.

# Mengelola RDS Proxy

Bagian ini memberikan informasi tentang cara mengelola operasi dan konfigurasi RDS Proxy. Prosedur ini membantu aplikasi Anda memaksimalkan koneksi basis data dan mencapai penggunaan ulang koneksi maksimum. Semakin Anda dapat memanfaatkan penggunaan kembali koneksi, semakin banyak CPU dan overhead memori yang dapat Anda simpan. Pada akhirnya, tindakan ini dapat mengurangi latensi untuk aplikasi Anda dan memungkinkan basis data untuk mendedikasikan lebih banyak sumber dayanya untuk memproses permintaan aplikasi.

Topik

- Mengubah Proksi RDS
- Menambahkan pengguna database baru saat menggunakan RDS Proxy
- Pertimbangan koneksi RDS Proxy
- Menghindari menyematkan Proxy RDS
- Menghapus Proxy RDS


## Mengubah Proksi RDS

Anda dapat mengubah pengaturan spesifik yang terkait dengan proksi setelah Anda membuat proksi. Caranya adalah dengan mengubah proksi itu sendiri, grup target terkaitnya, atau keduanya. Setiap proksi memiliki satu grup target terkait.

AWS Management Console

> **Important**
>
> Nilai dalam kolom Jenis autentikasi klien dan Autentikasi IAM berlaku untuk semua rahasia Secrets Manager yang terkait dengan proksi ini. Untuk menentukan nilai yang berbeda untuk setiap rahasia, ubah proxy Anda dengan menggunakan AWS CLI atau API sebagai gantinya.

Untuk mengubah pengaturan proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Dalam daftar proksi, pilih proksi yang pengaturannya ingin diubah atau kunjungi halaman detailnya.
4. Untuk Tindakan, pilih Ubah.
5. Masukkan atau pilih properti yang akan diubah. Anda dapat mengubah opsi berikut:
   - ID proksi – Mengganti nama proksi dengan memasukkan ID baru.
   - Batas waktu koneksi klien idle – Masukkan periode waktu untuk batas waktu koneksi klien idle.
   - Peran IAM – Mengubah peran IAM yang digunakan untuk mengambil rahasia dari Secrets Manager.

- Rahasia Secrets Manager – Menambahkan atau membuang rahasia Secrets Manager. Rahasia ini sesuai dengan nama dan kata sandi pengguna basis data.
- Jenis autentikasi klien – (PostgreSQL saja) Mengubah jenis autentikasi untuk koneksi klien ke proksi.
- Autentikasi IAM – Mewajibkan atau melarang autentikasi IAM untuk koneksi ke proksi.
- Wajibkan Keamanan Lapisan Pengangkutan – Mengaktifkan atau menonaktifkan persyaratan untuk Keamanan Lapisan Pengangkutan (TLS).
- Grup keamanan VPC – Menambahkan atau menghapus grup keamanan VPC yang akan digunakan proksi.
- Aktifkan pencatatan log yang disempurnakan – Mengaktifkan atau menonaktifkan pencatatan log yang disempurnakan.

6. Pilih Ubah.

Jika pengaturan yang ingin diubah tidak tercantum, gunakan prosedur berikut untuk memperbarui grup target untuk proksi. Grup target yang terkait dengan proksi mengontrol pengaturan yang terkait dengan koneksi basis data fisik. Setiap proksi memiliki satu grup target terkait bernama `default`, yang dibuat secara otomatis dengan proksi. Anda tidak dapat mengganti nama grup target default.

Anda hanya dapat mengubah grup target dari halaman detail proksi, bukan dari daftar pada halaman Proksi.

Untuk mengubah pengaturan grup target proksi

1. Pada halaman Proksi, buka halaman detail proksi.
2. Untuk Grup target, pilih tautan `default`. Saat ini, semua proksi memiliki satu grup target bernama `default`.
3. Pada halaman detail grup target default, pilih Ubah.
4. Pilih pengaturan baru untuk properti yang dapat diubah:

   - Basis data — Pilih klaster Aurora.
   - Koneksi maksimum kumpulan koneksi – Sesuaikan persentase maksimum koneksi yang tersedia yang dapat digunakan proksi.
   - Filter penyematan sesi – (Opsional) Pilih filter penyematan sesi. Tindakan ini menghindari langkah-langkah keamanan default untuk me-multipleks koneksi basis data di seluruh koneksi

klien. Saat ini, pengaturan tidak didukung untuk PostgreSQL. Satu-satunya pilihan adalah `EXCLUDE_VARIABLE_SETS`.

Mengaktifkan pengaturan ini dapat menyebabkan variabel sesi dari satu koneksi memengaruhi koneksi lain. Hal ini dapat menyebabkan kesalahan atau masalah ketepatan jika kueri Anda bergantung pada nilai variabel sesi yang ditetapkan di luar transaksi saat ini. Pertimbangkan untuk menggunakan opsi ini setelah memastikan bahwa aplikasi Anda sudah bisa berbagi koneksi basis data dengan aman di seluruh koneksi klien.

Pola berikut bisa dianggap aman:

- Pernyataan `SET` di mana tidak ada perubahan pada nilai variabel sesi efektif, yaitu tidak ada perubahan pada variabel sesi.
- Anda mengubah nilai variabel sesi dan mengeksekusi pernyataan dalam transaksi yang sama.

Untuk informasi selengkapnya, lihat Menghindari menyematkan Proxy RDS.

- Batas waktu peminjaman koneksi – Sesuaikan interval batas waktu peminjaman koneksi. Pengaturan ini berlaku saat jumlah maksimum koneksi sudah digunakan semua untuk proksi. Pengaturan ini menentukan seberapa lama proksi harus menunggu koneksi menjadi tersedia sebelum menampilkan sebuah kesalahan batas waktu.
- Kueri inisialisasi. (Opsional) Tambahkan kueri inisialisasi, atau ubah yang sekarang. Anda dapat menentukan satu atau beberapa pernyataan SQL untuk proksi yang akan dijalankan saat membuka setiap koneksi basis data baru. Pengaturan ini biasanya digunakan dengan `SET` pernyataan untuk memastikan bahwa setiap koneksi memiliki pengaturan yang identik. Pastikan kueri yang Anda tambahkan valid. Untuk memasukkan beberapa variabel dalam satu `SET` pernyataan, gunakan pemisah koma. Sebagai contoh:

```
SET variable1=value1, variable2=value2
```

Untuk beberapa pernyataan, gunakan titik koma sebagai pemisah.

Anda tidak dapat mengubah properti tertentu, seperti ID grup target dan mesin basis data.

5. Pilih Ubah grup target.

AWS CLI

Untuk memodifikasi proxy menggunakan AWS CLI, gunakan perintah modify-db-proxy, modify-db-proxy-target-group deregister-db-proxy-targets, dan register-db-proxy-targets.

Dengan perintah `modify-db-proxy`, Anda dapat mengubah properti seperti berikut:

- Kumpulan rahasia Secrets Manager yang digunakan proksi.
- Apakah TLS diperlukan.
- Batas waktu klien idle.
- Apakah harus mencatat informasi tambahan dari pernyataan SQL untuk debugging.
- Peran IAM yang digunakan untuk mengambil rahasia Secrets Manager.
- Grup keamanan yang digunakan proksi.

Contoh berikut menunjukkan cara mengganti nama proksi yang sudah ada.

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

Untuk mengubah pengaturan terkait koneksi atau mengganti nama grup target, gunakan perintah `modify-db-proxy-target-group`. Saat ini, semua proksi memiliki satu grup target bernama `default`. Saat Anda bekerja dengan grup target ini, Anda menentukan nama proxy dan `default` untuk nama grup target. Anda tidak dapat mengganti nama grup target default.

Contoh berikut ini menunjukkan cara memeriksa pengaturan `MaxIdleConnectionsPercent` untuk proksi terlebih dahulu dan kemudian mengubahnya menggunakan grup target.

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
```

```
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

Dengan perintah `deregister-db-proxy-targets` dan `register-db-proxy-targets`, Anda dapat mengubah klaster DB Aurora mana yang dikaitkan dengan proksi melalui grup targetnya. Saat ini, setiap proksi dapat terhubung ke satu klaster DB Aurora. Grup target melacak detail koneksi untuk semua semua instans DB dalam klaster Aurora.

Contoh berikut dimulai dengan proksi yang dikaitkan dengan klaster Aurora MySQL bernama `cluster-56-2020-02-25-1399`. Contoh ini menunjukkan cara mengubah proksi sehingga dapat terhubung ke klaster lainnya yang bernama `provisioned-cluster`.

Saat menggunakan klaster DB Aurora, Anda dapat menentukan opsi `--db-cluster-identifier`.

Contoh berikut mengubah proksi Aurora MySQL. Proksi Aurora PostgreSQL memiliki port 5432.

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy
```

```
{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}
```

```
aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

API RDS

Untuk memodifikasi proxy menggunakan RDS API, Anda menggunakan operasi Modify, Modify DBProxy, Deregister DBProxy Targets DBProxyTargetGroup, dan Register Targets operations. DBProxy

Dengan `ModifyDBProxy`, Anda dapat mengubah properti seperti berikut:

- Kumpulan rahasia Secrets Manager yang digunakan proksi.
- Apakah TLS diperlukan.
- Batas waktu klien idle.
- Apakah harus mencatat informasi tambahan dari pernyataan SQL untuk debugging.
- Peran IAM yang digunakan untuk mengambil rahasia Secrets Manager.
- Grup keamanan yang digunakan proksi.

DenganModifyDBProxyTargetGroup, Anda dapat mengubah pengaturan terkait koneksi. Saat ini, semua proksi memiliki satu grup target bernama default. Saat Anda bekerja dengan grup target ini, Anda menentukan nama proxy dan default untuk nama grup target. Anda tidak dapat mengganti nama grup target default.

Dengan DeregisterDBProxyTargets dan RegisterDBProxyTargets, Anda dapat mengubah klaster Aurora mana yang dikaitkan dengan proksi melalui grup targetnya. Saat ini, setiap proksi dapat terhubung ke satu klaster Aurora. Grup target melacak detail koneksi untuk instans DB dalam klaster Aurora.

## Menambahkan pengguna database baru saat menggunakan RDS Proxy

Dalam beberapa kasus, Anda dapat menambahkan pengguna database baru ke Aurora cluster yang terkait dengan proxy. Jika demikian, tambahkan atau ganti tujuan sebuah rahasia Secrets Manager untuk menyimpan kredensial dari pengguna tersebut. Untuk melakukan ini, jalankan langkah-langkah berikut:

1. Buat rahasia Secrets Manager yang baru, dengan menggunakan prosedur yang dijelaskan dalam Menyiapkan kredensi database AWS Secrets Manager untuk RDS Proxy.
2. Perbarui IAM peran untuk memberikan akses RDS Proxy ke rahasia Secrets Manager yang baru. Untuk melakukannya, perbarui bagian sumber daya dari kebijakan IAM peran.
3. Ubah RDS Proxy untuk menambahkan rahasia Secrets Manager baru di bawah rahasia Secrets Manager.
4. Jika pengguna baru menggantikan yang sudah ada, perbarui kredensial yang tersimpan dalam rahasia Secrets Manager proksi untuk pengguna yang sudah ada.

### Menambahkan pengguna database baru ke database Postgre saat menggunakan SQL Proxy RDS

Saat menambahkan pengguna baru ke SQL database Postgre Anda, jika Anda telah menjalankan perintah berikut:

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

Berikan hak istimewa CONNECT kepada pengguna rdsproxyadmin sehingga pengguna dapat memantau koneksi pada basis data target.

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

Anda juga dapat mengizinkan pengguna basis data target lainnya untuk melakukan pemeriksaan kondisi dengan mengubah `rdsproxyadmin` ke pengguna basis data dalam perintah di atas.

### Mengubah kata sandi untuk pengguna database saat menggunakan RDS Proxy

Dalam beberapa kasus, Anda mungkin mengubah kata sandi untuk pengguna database di cluster Aurora yang terkait dengan proxy. Jika demikian, perbarui rahasia Secrets Manager yang sesuai dengan kata sandi baru.

## Pertimbangan koneksi RDS Proxy

### Mengonfigurasi pengaturan koneksi

Untuk menyesuaikan pengumpulan koneksi Proksi RDS, Anda dapat mengubah pengaturan berikut:

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

#### IdleClientTimeout

Anda dapat menentukan berapa lama koneksi klien bisa berada dalam status idle sebelum proksi menutupnya. Nilai default-nya adalah 1.800 detik (30 menit).

Koneksi klien dianggap idle jika aplikasi tidak mengirimkan permintaan baru dalam waktu yang ditentukan setelah permintaan sebelumnya selesai. Koneksi basis data yang mendasarinya akan tetap terbuka dan dikembalikan ke kumpulan koneksi. Oleh karena itu, koneksi dapat digunakan kembali untuk koneksi klien baru. Jika Anda ingin proksi secara proaktif menghapus koneksi yang sudah tidak terpakai, turunkan batas waktu koneksi klien yang idle. Jika beban kerja Anda sering membuat koneksi dengan proxy, maka naikkan batas waktu koneksi klien idle untuk menghemat biaya pembuatan koneksi.

Pengaturan ini diwakili oleh bidang batas waktu koneksi klien Idle di konsol RDS dan `IdleClientTimeout` pengaturan di AWS CLI dan API. Untuk mempelajari cara mengubah nilai kolom Batas waktu koneksi klien idle di konsol RDS, lihat AWS Management Console. Untuk

mempelajari cara mengubah nilai `IdleClientTimeout` setelan, lihat perintah CLI modify-db-proxyatau operasi API Modify. DBProxy

MaxConnectionsPercent

Anda dapat membatasi jumlah koneksi yang dapat dibuat oleh Proksi RDS dengan basis data target. Anda menentukan batas dalam bentuk persentase koneksi maksimum yang tersedia untuk basis data Anda. Pengaturan ini diwakili oleh bidang koneksi maksimum Connection pool di konsol RDS dan `MaxConnectionsPercent` pengaturan di AWS CLI dan API.

Nilai `MaxConnectionsPercent` dinyatakan sebagai persentase dari pengaturan `max_connections` untuk klaster DB Aurora yang digunakan oleh grup target. Proksi tidak membuat semua koneksi ini di depan. Pengaturan ini memungkinkan proksi membuat koneksi ini karena beban kerja membutuhkannya.

Misalnya, untuk target basis data terdaftar dengan `max_connections` diatur ke 1000, dan `MaxConnectionsPercent` diatur ke 95, Proksi RDS menetapkan 950 koneksi sebagai batas atas koneksi bersamaan ke target basis data tersebut.

Efek samping umum beban kerja yang mencapai jumlah maksimum koneksi basis data yang diizinkan adalah peningkatan latensi kueri secara keseluruhan, disertai peningkatan metrik `DatabaseConnectionsBorrowLatency`. Anda dapat memantau koneksi basis data yang saat ini digunakan dan total koneksi basis data yang diizinkan dengan membandingkan metrik `DatabaseConnections` dan `MaxDatabaseConnectionsAllowed`.

Saat mengatur parameter ini, perhatikan praktik terbaik berikut:

- Izinkan headroom koneksi yang cukup untuk perubahan pola beban kerja. Sebaiknya atur parameter ini setidaknya 30% di atas penggunaan maksimum yang dipantau baru-baru ini. Karena Proksi RDS mendistribusikan ulang kuota koneksi basis data di beberapa simpul, perubahan kapasitas internal mungkin memerlukan setidaknya 30% headroom untuk koneksi tambahan guna menghindari peningkatan latensi pinjaman.
- Proksi RDS mencadangkan jumlah koneksi tertentu untuk pemantauan aktif guna mendukung failover cepat, perutean lalu lintas, dan operasi internal. Metrik `MaxDatabaseConnectionsAllowed` tidak mencakup koneksi yang dicadangkan ini. Metrik ini mewakili jumlah koneksi yang tersedia untuk melayani beban kerja, dan bisa lebih rendah dari nilai yang berasal dari pengaturan `MaxConnectionsPercent`.

  Nilai `MaxConnectionsPercent` minimum yang direkomendasikan adalah:

- db.t3.small: 100
- db.t3.medium: 55
- db.t3.large: 35
- db.r3.large atau lebih: 20

Jika beberapa instans target didaftarkan di Proksi RDS seperti klaster Aurora dengan simpul pembaca, tetapkan nilai minimum berdasarkan instans terkecil yang terdaftar.

Untuk mempelajari cara mengubah nilai kolom Batas waktu maksimum kumpulan koneksi di konsol RDS, lihat AWS Management Console. Untuk mempelajari cara mengubah nilai `MaxConnectionsPercent` setelan, lihat perintah CLI modify-db-proxy-target-group atau operasi API Modify. DBProxy TargetGroup

> **Important**
>
> Jika klaster DB adalah bagian dari basis data global dengan penerusan tulis yang diaktifkan, kurangi nilai `MaxConnectionsPercent` proksi dengan kuota yang dialokasikan untuk penerusan tulis. Kuota penerusan tulis diatur dalam parameter klaster DB `aurora_fwd_writer_max_connections_pct`. Untuk informasi tentang penerusan tulis, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Untuk informasi tentang batas koneksi basis data, lihat Koneksi maksimum ke instans DB Aurora MySQL dan Koneksi maksimum ke instans DB Aurora PostgreSQL.

MaxIdleConnectionsPercent

Anda dapat mengontrol jumlah koneksi basis data idle yang dapat disimpan oleh Proksi RDS di kumpulan koneksi. Secara default, Proksi RDS menganggap koneksi basis data di kumpulannya menjadi idle jika tidak ada aktivitas pada koneksi selama lima menit.

`MaxIdleConnectionsPercent`Nilai dinyatakan sebagai persentase dari `max_connections` pengaturan untuk kelompok target instans RDS DB. Nilai default-nya adalah 50 persen dari `MaxConnectionsPercent`, dan batas atasnya adalah nilai `MaxConnectionsPercent`. Misalnya, jika`MaxConnectionsPercent`, adalah 80, maka nilai default `MaxIdleConnectionsPercent` adalah 40.

Dengan nilai tinggi, proksi membiarkan koneksi basis data idle dengan persentase yang tinggi tetap terbuka. Dengan nilai rendah, proksi menutup koneksi basis data idle dengan persentase yang tinggi. Jika beban kerja Anda tidak dapat diprediksi, pertimbangkan untuk mengatur nilai tinggi untuk `MaxIdleConnectionsPercent`. Jika Anda melakukannya, Proksi RDS dapat mengakomodasi lonjakan aktivitas tanpa membuka banyak koneksi basis data baru.

Pengaturan ini diwakili oleh `MaxIdleConnectionsPercent` pengaturan `DBProxyTargetGroup` di AWS CLI dan API. Untuk mempelajari cara mengubah nilai `MaxIdleConnectionsPercent` setelan, lihat perintah CLI modify-db-proxy-target-group atau operasi API Modify. DBProxy TargetGroup

Untuk informasi tentang batas koneksi basis data, lihat Koneksi maksimum ke instans DB Aurora MySQL dan Koneksi maksimum ke instans DB Aurora PostgreSQL.

ConnectionBorrowTimeout

Anda dapat memilih berapa lama Proksi RDS menunggu koneksi basis data dalam kumpulan koneksi menjadi tersedia untuk digunakan sebelum menampilkan eror batas waktu. Periode default-nya adalah 120 detik. Pengaturan ini berlaku saat jumlah koneksi mencapai titik maksimum, sehingga tidak ada koneksi yang tersedia dalam kumpulan koneksi. Pengaturan ini juga berlaku ketika tidak ada instans basis data yang tersedia untuk menangani permintaan, seperti saat operasi failover sedang berlangsung. Dengan pengaturan ini, Anda dapat mengatur periode tunggu terbaik untuk aplikasi Anda tanpa mengubah batas waktu kueri dalam kode aplikasi Anda.

Pengaturan ini diwakili oleh bidang batas waktu pinjam koneksi di konsol RDS atau `ConnectionBorrowTimeout` pengaturan `DBProxyTargetGroup` di API atau. AWS CLI Untuk mempelajari cara mengubah nilai kolom Batas waktu peminjaman koneksi di konsol RDS, lihat AWS Management Console. Untuk mempelajari cara mengubah nilai `ConnectionBorrowTimeout` setelan, lihat perintah CLI modify-db-proxy-target-group atau operasi API Modify. DBProxy TargetGroup

## Koneksi klien dan basis data

Koneksi dari aplikasi Anda ke Proksi RDS dikenal sebagai koneksi klien. Koneksi dari proxy ke basis data adalah koneksi basis data. Saat menggunakan Proksi RDS, koneksi klien berakhir di proksi sementara koneksi basis data dikelola dalam Proksi RDS.

Pengumpulan koneksi sisi aplikasi dapat memberikan manfaat untuk mengurangi pembuatan koneksi berulang antara aplikasi Anda dan Proksi RDS.

Pertimbangkan aspek konfigurasi berikut sebelum menerapkan kumpulan koneksi sisi aplikasi:

- Masa pakai maks koneksi klien: Proksi RDS menerapkan masa pakai maksimum koneksi klien selama 24 jam. Nilai ini tidak dapat dikonfigurasi. Konfigurasikan kumpulan Anda dengan masa pakai koneksi maksimum kurang dari 24 jam guna menghindari penurunan koneksi klien yang tidak terduga.
- Batas waktu idle koneksi klien: Proksi RDS menerapkan waktu idle maksimum untuk koneksi klien. Konfigurasikan kumpulan Anda dengan batas waktu koneksi idle dengan nilai yang lebih rendah dari pengaturan batas waktu idle koneksi klien untuk Proksi RDS guna menghindari penurunan koneksi yang tidak terduga.

Jumlah maksimum koneksi klien yang dikonfigurasi dalam kumpulan koneksi sisi aplikasi Anda tidak harus dibatasi pada pengaturan max_connections untuk Proksi RDS.

Pengumpulan koneksi klien menghasilkan masa pakai koneksi klien yang lebih lama. Jika koneksi Anda mengalami penyematan, pengumpulan koneksi klien dapat mengurangi efisiensi multiplexing. Koneksi klien yang disematkan tetapi idle dalam kumpulan koneksi sisi aplikasi terus berpegang pada koneksi basis data dan mencegah koneksi basis data digunakan kembali oleh koneksi klien lainnya. Tinjau log proksi untuk memeriksa apakah koneksi Anda mengalami penyematan.

> **Note**
>
> RDS Proxy menutup koneksi database beberapa saat setelah 24 jam ketika mereka tidak lagi digunakan. Proxy melakukan tindakan ini terlepas dari nilai pengaturan koneksi idle maksimum.

## Menghindari menyematkan Proxy RDS

Multiplexing akan lebih efisien saat permintaan basis data tidak bergantung pada informasi status dari permintaan sebelumnya. Dalam kasus ini, Proksi RDS dapat menggunakan kembali koneksi saat setiap transaksi selesai. Contoh informasi status tersebut mencakup sebagian besar variabel dan parameter konfigurasi yang dapat diubah melalui pernyataan `SET` atau `SELECT`. Secara default, transaksi SQL pada koneksi klien dapat bermultipleks antar-koneksi basis data acuan.

Koneksi ke proksi dapat memasukkan status yang disebut sebagai penyematan. Saat koneksi disematkan, setiap transaksi berikutnya akan menggunakan koneksi basis data acuan yang sama

hingga sesi berakhir. Koneksi klien lainnya juga tidak dapat menggunakan kembali koneksi basis data tersebut hingga sesi berakhir. Sesi berakhir saat koneksi klien terputus.

Proksi RDS secara otomatis menyematkan koneksi klien ke koneksi DB tertentu saat mendeteksi perubahan sebuah status sesi yang tidak sesuai untuk sesi lainnya. Penyematan mengurangi efektivitas penggunaan kembali koneksi. Jika semua atau hampir semua koneksi Anda mengalami penyematan, pertimbangkan untuk mengubah kode aplikasi atau beban kerja untuk mengurangi kondisi yang menyebabkan penyematan.

Misalnya, aplikasi Anda mengubah variabel sesi atau parameter konfigurasi. Dalam hal ini, pernyataan selanjutnya dapat mengandalkan variabel atau parameter baru yang berlaku. Oleh karena itu, saat Proksi RDS memproses permintaan untuk mengubah variabel sesi atau pengaturan konfigurasi, sesi ini akan disematkan ke koneksi DB. Dengan demikian, tahap sesi tetap berfungsi untuk semua transaksi berikutnya dalam sesi yang sama.

Untuk beberapa mesin basis data, aturan ini tidak berlaku untuk semua parameter yang dapat Anda atur. Proksi RDS melacak pernyataan dan variabel tertentu. Oleh karena itu, Proksi RDS tidak menyematkan sesi saat Anda mengubahnya. Dalam kasus ini, Proksi RDS hanya menggunakan kembali koneksi untuk sesi lain yang memiliki nilai yang sama untuk pengaturan tersebut. Untuk daftar pernyataan dan variabel yang dilacak untuk Aurora MySQL, lihat Apa yang dilacak Proksi RDS untuk basis data Aurora MySQL.

## Apa yang dilacak Proksi RDS untuk basis data Aurora MySQL

RDS Proxy melacak pernyataan MySQL berikut:

- DROP DATABASE
- DROP SCHEMA
- USE

RDS Proxy melacak variabel MySQL berikut:

- `AUTOCOMMIT`
- `AUTO_INCREMENT_INCREMENT`
- `CHARACTER SET (or CHAR SET)`
- `CHARACTER_SET_CLIENT`
- `CHARACTER_SET_DATABASE`

- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

> **Note**
>
> RDS Proxy melacak perubahan pada TRANSACTION_READ_ONLY variabel TRANSACTION_ISOLATION dan saat Anda mengaturnya di lingkup sesi. Namun, jika Anda mengaturnya pada cakupan transaksi berikutnya, RDS Proxy pin koneksi. Perilaku ini berlaku apakah Anda menggunakan SET pernyataan atau SET TRANSACTION pernyataan untuk mengonfigurasi nilai-nilai ini.

## Meminimalkan penyematan

Penyetelan performa untuk Proksi RDS meliputi upaya memaksimalkan penggunaan kembali koneksi tingkat transaksi (multiplexing) dengan meminimalkan penyematan.

Anda dapat meminimalkan penyematan dengan melakukan hal berikut:

- Hindari permintaan basis data yang tidak perlu yang dapat menyebabkan penyematan.
- Atur variabel dan pengaturan konfigurasi secara konsisten di semua koneksi. Dengan demikian, sesi berikutnya cenderung menggunakan kembali koneksi yang memiliki pengaturan tertentu tersebut.

  Akan tetapi, untuk pengaturan PostgreSQL, sebuah variabel bisa menimbulkan penyematan sesi.
- Untuk basis data keluarga MySQL, terapkan sebuah filter penyematan sesi ke proksi. Anda dapat mengecualikan jenis operasi tertentu dari penyematan sesi jika Anda mengetahui bahwa tindakan ini tidak memengaruhi operasi yang benar aplikasi Anda.
- Lihat seberapa sering penyematan terjadi dengan memantau CloudWatch metrik `DatabaseConnectionsCurrentlySessionPinned` Amazon. Untuk informasi tentang ini dan CloudWatch metrik lainnya, lihatMemantau metrik Proxy RDS dengan Amazon CloudWatch.
- Jika menggunakan pernyataan `SET` untuk melakukan inisialisasi yang identik untuk setiap koneksi klien, Anda dapat melakukannya sekaligus mempertahankan multiplexing tingkat transaksi. Dalam kasus ini, Anda memindahkan pernyataan yang menyiapkan status sesi awal ke dalam kueri inisialisasi yang digunakan oleh proksi. Properti ini adalah string yang berisi satu atau beberapa pernyataan SQL, yang dipisahkan oleh titik koma.

  Misalnya, Anda dapat menentukan kueri inisialisasi untuk proksi yang menetapkan parameter konfigurasi tertentu. Kemudian, Proksi RDS menerapkan pengaturan tersebut setiap kali koneksi baru untuk proksi itu disiapkan. Anda dapat menghapus pernyataan `SET` yang sesuai dari kode aplikasi, sehingga tidak mengganggu multiplexing tingkat transaksi.

  Untuk metrik tentang seberapa sering penyematan terjadi pada proksi, lihat Memantau metrik Proxy RDS dengan Amazon CloudWatch.

## Kondisi yang menyebabkan penyematan untuk semua keluarga mesin

Proksi menyematkan sesi ke koneksi saat ini dalam situasi berikut ketika multiplexing dapat menyebabkan perilaku yang tidak terduga:

- Pernyataan apa pun dengan ukuran teks lebih dari 16 KB bisa menyebabkan proksi menyematkan sesi.

## Kondisi yang menyebabkan penyematan untuk Aurora MySQL

Untuk MySQL, interaksi berikut juga menyebabkan penyematan:

- Pernyataan kunci tabel eksplisit `LOCK TABLE`, `LOCK TABLES`, atau `FLUSH TABLES WITH READ LOCK` menyebabkan proksi menyematkan sesi.
- Membuat kunci bernama dengan menggunakan `GET_LOCK` menyebabkan proksi menyematkan sesi.
- Menyetel variabel pengguna atau sistem (dengan beberapa pengecualian) menyematkan sesi ke proxy. Jika ini secara signifikan membatasi penggunaan kembali koneksi, Anda dapat mengonfigurasi `SET` operasi untuk menghindari penyematan. Untuk melakukan ini, sesuaikan properti filter penyematan sesi. Untuk informasi selengkapnya, silakan lihat Membuat proxy untuk dan Mengubah Proksi RDS.
- Membuat tabel sementara menyebabkan proksi menyematkan sesi. Dengan begitu, konten tabel sementara dipertahankan sepanjang sesi, terlepas dari batasan transaksi.
- Memanggil `ROW_COUNT` dan `FOUND_ROWS` fungsi terkadang menyebabkan penyematan.

  Keadaan ketika fungsi ini menyebabkan penyematan mungkin berbeda antara versi Aurora MySQL yang kompatibel dengan MySQL 5.7.
- Pernyataan yang disiapkan menyebabkan proksi menyematkan sesi. Aturan ini berlaku terlepas dari apakah pernyataan yang disiapkan menggunakan teks SQL maupun protokol biner.
- Proksi RDS tidak menyematkan koneksi saat Anda menggunakan SET LOCAL.
- Memanggil prosedur tersimpan dan fungsi tersimpan tidak menyebabkan pinning. Proksi RDS tidak mendeteksi perubahan status sesi apa pun yang terjadi akibat perintah tersebut. Pastikan aplikasi Anda tidak mengubah status sesi di dalam rutinitas tersimpan jika Anda mengandalkan status sesi tersebut untuk bertahan di seluruh transaksi. Misalnya, Proxy RDS saat ini tidak kompatibel dengan prosedur tersimpan yang membuat tabel sementara yang bertahan di semua transaksi.

Jika memiliki pengetahuan mendalam tentang perilaku aplikasi, Anda dapat menangani perilaku penyematan untuk pernyataan aplikasi tertentu. Untuk melakukannya, pilih opsi Filter penyematan sesi saat membuat proksi. Saat ini, Anda dapat memilih untuk tidak menggunakan penyematan sesi untuk pengaturan variabel sesi dan pengaturan konfigurasi.

### Kondisi yang menyebabkan penyematan untuk Aurora PostgreSQL

Untuk PostgreSQL, interaksi berikut juga menyebabkan penyematan:

- Menggunakan `SET` perintah.
- Menggunakan`PREPARE`,`DISCARD`,`DEALLOCATE`, atau `EXECUTE` perintah untuk mengelola pernyataan yang disiapkan.

- Membuat urutan sementara, tabel, atau tampilan.
- Mendeklarasikan kursor.
- Membuang status sesi.
- Mendengarkan di saluran notifikasi.
- Memuat modul perpustakaan sepertiauto_explain.
- Memanipulasi urutan menggunakan fungsi seperti nextval dan. setval
- Berinteraksi dengan kunci menggunakan fungsi seperti pg_advisory_lock danpg_try_advisory_lock.

> **Note**
>
> RDS Proxy tidak menyematkan kunci penasihat tingkat transaksi, khususnya,pg_advisory_xact_lock, pg_advisory_xact_lock_sharedpg_try_advisory_xact_lock, dan. pg_try_advisory_xact_lock_shared

- Mengatur parameter, atau mengatur ulang parameter ke defaultnya. Secara khusus, menggunakan SET dan set_config perintah untuk menetapkan nilai default ke variabel sesi.
- Memanggil prosedur tersimpan dan fungsi tersimpan tidak menyebabkan pinning. Proksi RDS tidak mendeteksi perubahan status sesi apa pun yang terjadi akibat perintah tersebut. Pastikan aplikasi Anda tidak mengubah status sesi di dalam rutinitas tersimpan jika Anda mengandalkan status sesi tersebut untuk bertahan di seluruh transaksi. Misalnya, Proxy RDS saat ini tidak kompatibel dengan prosedur tersimpan yang membuat tabel sementara yang bertahan di semua transaksi.

## Menghapus Proxy RDS

Anda dapat menghapus proksi saat tidak lagi membutuhkannya. Atau Anda dapat menghapus proksi jika Anda merasa instans atau klaster DB yang terkait dengannya sedang dalam perbaikan.

AWS Management Console

Untuk menghapus proksi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.

3. Pilih proksi yang akan dihapus dari daftar.
4. Pilih Hapus Proksi.

AWS CLI

Untuk menghapus proxy DB, gunakan AWS CLI perintah delete-db-proxy. Untuk menghapus asosiasi terkait, gunakan juga deregister-db-proxy-targetsperintah.

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS API

Untuk menghapus proxy DB, panggil RDS API fungsi Amazon D eleteDBProxy. Untuk menghapus item dan asosiasi terkait, Anda juga memanggil fungsi D eleteDBProxy TargetGroup dan D eregisterDBProxy Target.

# Bekerja dengan titik akhir Proksi Amazon RDS

Endpoint RDS Proxy menyediakan cara yang fleksibel dan efisien untuk mengelola koneksi database, yang meningkatkan skalabilitas, ketersediaan, dan keamanan. Dengan titik akhir proxy, Anda dapat:

- Sederhanakan pemantauan dan pemecahan masalah — Gunakan beberapa titik akhir untuk melacak dan mengelola koneksi dari berbagai aplikasi secara independen.
- Tingkatkan skalabilitas baca — Manfaatkan titik akhir pembaca dengan cluster Aurora DB untuk mendistribusikan lalu lintas baca secara efisien, yang mengoptimalkan kinerja untuk beban kerja yang berat kueri.
- Aktifkan akses lintas-VPC — Hubungkan database di seluruh VPCs menggunakan titik akhir lintas-VPC, yang memungkinkan sumber daya seperti instans Amazon EC2 di satu VPC mengakses database di VPC lainnya.

Topik

- Ikhtisar titik akhir proksi
- Batasan untuk titik akhir proksi
- Menggunakan titik akhir pembaca dengan klaster Aurora
- Mengakses database RDS di seluruh VPCs
- Membuat titik akhir proksi
- Melihat titik akhir proksi
- Mengubah titik akhir proksi
- Menghapus titik akhir proksi


## Ikhtisar titik akhir proksi

Bekerja dengan titik akhir Proksi RDS melibatkan jenis prosedur yang sama seperti dengan klaster DB Aurora dan titik akhir pembaca. Jika Anda belum terbiasa dengan titik akhir Aurora, temukan informasi selengkapnya di Koneksi titik akhir Amazon Aurora.

Saat Anda menggunakan Proxy RDS dengan cluster Aurora, titik akhir default mendukung operasi baca dan tulis. Ini berarti ia merutekan semua permintaan ke instance penulis cluster, berkontribusi pada `max_connections` batasnya. Untuk mendistribusikan lalu lintas dengan lebih baik, Anda dapat membuat endpoint baca/tulis atau read-only tambahan untuk proxy Anda, yang memungkinkan manajemen beban kerja yang lebih efisien dan peningkatan skalabilitas.

Gunakan endpoint read-only dengan proxy Anda untuk menangani kueri baca, seperti yang Anda lakukan dengan titik akhir pembaca untuk klaster yang disediakan Aurora. Pendekatan ini memaksimalkan skalabilitas baca Aurora dengan mendistribusikan kueri di satu atau lebih instance DB pembaca. Dengan menggunakan endpoint read-only dan menambahkan lebih banyak instance pembaca sesuai kebutuhan, Anda dapat meningkatkan jumlah kueri dan koneksi simultan yang dapat ditangani klaster Anda.

> **Tip**
>
> Saat Anda membuat proxy untuk cluster Aurora menggunakan AWS Management Console, Anda dapat meminta Proxy RDS secara otomatis membuat titik akhir pembaca. Untuk informasi tentang manfaat titik akhir pembaca, lihat Menggunakan titik akhir pembaca dengan klaster Aurora.

Saat Anda membuat titik akhir proxy, Anda dapat mengaitkannya dengan virtual private cloud (VPC) yang berbeda dari VPC proxy. Ini memungkinkan Anda untuk terhubung ke proxy dari VPC lain, seperti yang digunakan oleh aplikasi yang berbeda dalam organisasi Anda.

Untuk informasi tentang batas yang terkait dengan titik akhir proksi, lihat Batasan untuk titik akhir proksi.

RDS Proxy log awalan setiap entri dengan nama endpoint proxy terkait. Ini bisa berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna, atau nama khusus `default` untuk titik akhir baca/tulis default proxy.

Setiap titik akhir proxy memiliki kumpulan CloudWatch metriknya sendiri. Pantau metrik untuk semua titik akhir proxy, titik akhir tertentu, atau semua titik akhir baca/tulis atau baca-saja dari proxy. Untuk informasi selengkapnya, lihat Memantau metrik Proxy RDS dengan Amazon CloudWatch.

Titik akhir proksi menggunakan mekanisme autentikasi yang sama seperti proksi yang terkait. Proksi RDS secara otomatis menyiapkan izin dan otorisasi untuk titik akhir yang ditentukan pengguna, yang konsisten dengan properti proksi terkait.

Untuk mempelajari cara kerja titik akhir proksi untuk klaster DB dalam Aurora global basis data, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

## Batasan untuk titik akhir proksi

Titik akhir Proksi RDS memiliki batasan berikut:

- Titik akhir default proxy RDS tidak dapat dimodifikasi.
- Jumlah maksimum titik akhir yang ditentukan pengguna untuk proksi adalah 20. Dengan demikian, proksi dapat memiliki hingga 21 titik akhir: titik akhir default, ditambah 20 titik akhir yang Anda buat.
- Jika Anda mengaitkan titik akhir tambahan dengan proksi, Proksi RDS secara otomatis menentukan instans DB dalam klaster Anda yang digunakan untuk setiap titik akhir. Anda tidak dapat memilih instans tertentu seperti yang Anda lakukan dengan titik akhir kustom Aurora.

Saat Anda membuat proxy, RDS secara otomatis membuat titik akhir VPC untuk komunikasi yang aman antara aplikasi dan database. Titik akhir VPC terlihat dan dapat diakses dari Konsol VPC Amazon.

Menambahkan titik akhir proxy baru menyediakan titik akhir AWS PrivateLink antarmuka. Jika Anda menambahkan satu atau beberapa titik akhir ke proxy Anda, Anda akan dikenakan biaya tambahan. Untuk informasi selengkapnya, lihat Harga Proxy RDS.

## Menggunakan titik akhir pembaca dengan klaster Aurora

Anda dapat membuat dan menyambung ke titik akhir hanya-baca yang disebut titik akhir pembaca jika Anda menggunakan Proksi RDS dengan klaster Aurora. Titik akhir pembaca ini membantu meningkatkan skalabilitas baca aplikasi padat kueri. Titik akhir pembaca juga membantu meningkatkan ketersediaan koneksi Anda jika instans DB pembaca di klaster Anda menjadi tidak tersedia.

> **Note**
>
> Jika Anda menentukan bahwa titik akhir yang baru adalah titik akhir hanya-baca, Proksi RDS mengharuskan klaster Aurora memiliki satu atau beberapa instans DB pembaca. Dalam beberapa kasus, Anda mungkin mengubah target proxy ke cluster Aurora yang hanya berisi satu penulis. Jika Anda melakukannya, permintaan apa pun ke titik akhir pembaca akan gagal dengan kesalahan. Permintaan juga gagal jika target proksi adalah instans RDS, bukan klaster Aurora.
>
> Jika klaster Aurora memiliki instans pembaca, tetapi instans tersebut tidak tersedia, Proksi RDS akan menunggu untuk mengirim permintaan, bukan serta-merta menampilkan kesalahan. Jika tidak ada instans pembaca yang tersedia dalam periode batas waktu peminjaman koneksi, permintaan gagal akan dengan kesalahan.

### Cara titik akhir pembaca membantu ketersediaan aplikasi

Dalam beberapa kasus, satu atau beberapa instans pembaca dalam klaster Anda mungkin menjadi tidak tersedia. Jika demikian, koneksi yang menggunakan titik akhir pembaca proksi DB dapat pulih lebih cepat daripada koneksi yang menggunakan titik akhir pembaca Aurora. Proksi RDS hanya merutekan koneksi ke instans pembaca yang tersedia dalam klaster. Tidak ada penundaan dikarenakan caching DNS saat instans menjadi tidak tersedia.

Jika koneksi di-multipleks, Proksi RDS mengarahkan kueri berikutnya ke instans DB pembaca yang berbeda tanpa mengganggu aplikasi. Selama switchover otomatis ke instans pembaca baru, Proksi RDS memeriksa keterlambatan replikasi instans pembaca lama dan baru. Proksi RDS memastikan bahwa instans pembaca baru diperbarui dengan perubahan yang sama seperti instans pembaca

sebelumnya. Dengan begitu, aplikasi Anda tidak pernah melihat data usang ketika Proksi RDS beralih dari satu instans DB pembaca ke instans DB pembaca lainnya.

Jika koneksi disematkan, kueri berikutnya pada koneksi akan menampilkan kesalahan. Namun, aplikasi Anda dapat langsung terhubung kembali ke titik akhir yang sama. Proksi RDS merutekan koneksi ke instans DB pembaca yang berbeda yang ada dalam status `available`. Ketika Anda terhubung kembali secara manual, Proksi RDS tidak memeriksa keterlambatan replikasi antara instans pembaca lama dan baru.

Jika klaster Aurora Anda tidak memiliki instans pembaca yang tersedia, Proksi RDS akan memeriksa apakah kondisi ini sementara atau permanen. Perilaku dalam setiap kasusnya adalah sebagai berikut:

- Misalkan klaster Anda memiliki satu atau beberapa instans DB pembaca, tetapi tidak satu pun berada dalam status `Available`. Sebagai contoh, semua instans pembaca mungkin di-boot ulang atau menghadapi masalah. Dalam hal ini, upaya untuk terhubung ke titik akhir pembaca menunggu instans pembaca menjadi tersedia. Jika tidak ada instans pembaca yang tersedia dalam periode batas waktu peminjaman koneksi, upaya koneksi akan gagal. Jika instans pembaca menjadi tersedia, upaya koneksi berhasil.
- Misalkan klaster Anda tidak memiliki instans DB pembaca. Dalam hal ini, Proksi RDS segera menampilkan kesalahan jika Anda mencoba untuk terhubung ke titik akhir pembaca. Untuk mengatasi masalah ini, tambahkan satu atau beberapa instans pembaca ke klaster Anda sebelum terhubung ke titik akhir pembaca.

## Cara titik akhir pembaca membantu skalabilitas kueri

Titik akhir pembaca untuk proksi membantu skalabilitas kueri Aurora dengan cara berikut:

- Saat Anda menambahkan instans pembaca ke klaster Aurora, Proksi RDS dapat merutekan koneksi baru ke setiap titik akhir pembaca ke instans pembaca yang berbeda. Dengan begitu, kueri yang dilakukan dengan menggunakan satu koneksi titik akhir pembaca tidak memperlambat kueri yang dilakukan menggunakan koneksi titik akhir pembaca lain. Kueri berjalan pada instans DB terpisah. Setiap instans DB memiliki sumber daya komputasinya sendiri, buffer cache, dan sebagainya.
- Jika praktis, Proksi RDS menggunakan instans DB pembaca yang sama untuk semua masalah kueri yang menggunakan koneksi titik akhir pembaca tertentu. Dengan demikian, satu kumpulan kueri terkait pada tabel yang sama dapat memanfaatkan caching, optimasi rencana, dan sebagainya, pada instans DB tertentu.

- Jika instans DB pembaca tidak tersedia, pengaruh pada aplikasi Anda bergantung pada apakah sesi di-multipleks atau disematkan. Jika sesi di-multipleks, Proksi RDS merutekan setiap kueri berikutnya untuk instans DB pembaca yang berbeda tanpa campur tangan Anda. Jika sesi disematkan, aplikasi Anda mengalami kesalahan dan harus dihubungkan kembali. Anda dapat segera terhubung kembali ke titik akhir pembaca dan Proksi RDS merutekan koneksi ke instans DB pembaca yang tersedia. Untuk informasi selengkapnya tentang cara memultipleks dan menyematkan untuk sesi proksi, lihat Ikhtisar konsep RDS Proxy.
- Semakin banyak instans DB pembaca yang Anda miliki dalam klaster, semakin banyak koneksi simultan yang dapat Anda buat menggunakan titik akhir pembaca. Misalnya, anggaplah bahwa klaster Anda memiliki empat instans DB pembaca, masing-masing dikonfigurasi untuk mendukung 200 koneksi simultan. Misalkan proksi Anda dikonfigurasi untuk menggunakan 50% dari koneksi maksimum. Di sini, jumlah maksimum koneksi yang dapat Anda buat melalui titik akhir pembaca di proksi ini adalah 100 (50% dari 200) untuk pembaca 1. Juga 100 koneksi untuk pembaca 2, dan seterusnya, dengan total 400. Jika Anda menggandakan jumlah instans DB pembaca klaster menjadi delapan, maka jumlah maksimum koneksi melalui titik akhir pembaca juga berlipat ganda, menjadi 800.

## Contoh penggunaan titik akhir pembaca

Contoh Linux berikut menunjukkan cara mengonfirmasi bahwa Anda terhubung ke klaster Aurora MySQL melalui titik akhir pembaca. Pengaturan konfigurasi `innodb_read_only` diaktifkan. Upaya untuk melakukan operasi tulis seperti pernyataan `CREATE DATABASE` gagal dengan kesalahan. Dan Anda dapat mengonfirmasi bahwa Anda terhubung ke instans DB pembaca dengan memeriksa nama instans DB menggunakan variabel `aurora_server_id`.

> **Tip**
>
> Jangan hanya mengandalkan pemeriksaan nama instans DB untuk menentukan apakah koneksi itu baca/tulis atau hanya-baca. Ingat bahwa instans DB dalam klaster Aurora dapat mengubah peran antara penulis dan pembaca ketika failover terjadi.

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
...
mysql> select @@innodb_read_only;
+--------------------+
```

```
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
+---------------------------------------+
| @@aurora_server_id                    |
+---------------------------------------+
| proxy-reader-endpoint-demo-instance-3 |
+---------------------------------------+
```

Contoh berikut menunjukkan bagaimana koneksi Anda ke titik akhir pembaca proksi dapat tetap bekerja bahkan ketika instans DB pembaca dihapus. Dalam contoh ini, klaster Aurora memiliki dua instans pembaca, `instance-5507` dan `instance-7448`. Koneksi ke titik akhir pembaca mulai menggunakan salah satu instans pembaca. Dalam contoh, instans pembaca ini dihapus oleh perintah `delete-db-instance`. Proksi RDS beralih ke instans pembaca yang berbeda untuk kueri berikutnya.

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+

mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
```

```
+----------+
|      328 |
+----------+
```

Ketika sesi `mysql` masih berjalan, perintah berikut menghapus instans pembaca yang terhubung ke titik akhir pembaca.

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

Kueri dalam sesi `mysql` terus bekerja tanpa perlu menghubungkan kembali. Proksi RDS beralih otomatis ke instans DB pembaca yang berbeda.

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+

mysql> select count(*) from information_schema.TABLES;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

## Mengakses database  RDS di seluruh VPCs

Secara default, semua komponen tumpukan teknologi Aurora Anda berada dalam Amazon VPC yang sama. Misalnya, misalkan aplikasi yang berjalan pada EC2 instans Amazon terhubung ke instans DB Aurora DB Amazon DB. Dalam hal ini, server aplikasi dan basis data keduanya harus berada dalam VPC yang sama.

Dengan RDS Proxy, Anda dapat mengatur akses ke instans dalam satu VPC dari sumber daya di VPC lain, seperti instance. EC2 Misalnya, organisasi Anda mungkin memiliki beberapa aplikasi yang mengakses sumber daya basis data yang sama. Setiap aplikasi mungkin berada dalam VPC-nya sendiri.

Untuk mengaktifkan akses lintas-VPC, Anda membuat titik akhir baru untuk proksi tersebut. Proksi itu sendiri berada di VPC yang sama dengan  klaster DB Aurora. Namun, titik akhir lintas-VPC

berada di VPC lain, bersama dengan sumber daya lain seperti instance. EC2 Titik akhir lintas-VPC dikaitkan dengan subnet dan grup keamanan dari VPC yang sama dengan sumber daya dan sumber daya lainnya. EC2 Pengaitan ini memungkinkan Anda terhubung ke titik akhir dari aplikasi yang jika sebaliknya tidak dapat mengakses basis data dikarenakan pembatasan VPC.

Langkah-langkah berikut menjelaskan cara membuat dan mengakses titik akhir lintas-VPC melalui Proksi RDS:

1. Buat dua VPCs, atau pilih dua VPCs yang sudah Anda gunakan untuk pekerjaan Aurora . Setiap VPC harus memiliki sumber daya jaringan yang terkaitnya sendiri seperti gateway internet, tabel rute, subnet, dan grup keamanan. Jika Anda hanya memiliki satu VPC, Anda dapat melihat Memulai dengan Amazon Aurora untuk mengetahui langkah-langkah penyiapan VPC lain agar berhasil menggunakan Aurora. Anda juga dapat memeriksa VPC yang ada di EC2 konsol Amazon untuk melihat jenis sumber daya untuk dihubungkan bersama.
2. Buat proksi DB yang terkait dengan  klaster DB Aurora yang ingin Anda hubungkan. Ikuti prosedur di Membuat proxy untuk .
3. Pada halaman Detail untuk proksi Anda di konsol RDS, di bagian Titik akhir proksi, pilih Buat titik akhir. Ikuti prosedur di Membuat titik akhir proksi.
4. Pilih apakah lintas-VPC endpoint akan bersifat baca/tulis atau hanya-baca.
5. Alih-alih menerima pengaturan default VPC yang sama dengan  klaster DB Aurora, pilih VPC yang berbeda. VPC ini harus berada di AWS Wilayah yang sama dengan VPC tempat proxy berada.
6. Sekarang, alih-alih menerima pengaturan default untuk subnet dan grup keamanan dari VPC yang sama dengan  klaster DB Aurora, buatlah pilihan baru. Buat pilihan ini berdasarkan subnet dan grup keamanan dari VPC yang Anda pilih.
7. Anda tidak perlu mengubah pengaturan apa pun untuk rahasia Secrets Manager. Kredensial yang sama dapat digunakan untuk semua titik akhir proksi Anda, terlepas dari VPC tempat setiap titik akhir berada.
8. Tunggu sampai titik akhir baru untuk mencapai status Tersedia.
9. Buat catatan nama titik akhir lengkap. Ini adalah nilai yang berakhirkan *`Region_name`*`.rds.amazonaws.com` yang Anda berikan sebagai bagian dari string koneksi untuk aplikasi basis data Anda.
10. Akses titik akhir baru dari sumber daya di VPC yang sama dengan titik akhir. Cara sederhana untuk menguji proses ini adalah dengan membuat EC2 instance baru di VPC ini. Kemudian, masuk ke EC2 instance dan jalankan `psql` perintah `mysql` or untuk terhubung dengan menggunakan nilai endpoint dalam string koneksi Anda.

# Membuat titik akhir proksi

Untuk membuat titik akhir proxy, ikuti petunjuk ini:

Konsol

Untuk membuat titik akhir proksi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Klik nama proksi yang titik akhir barunya ingin dibuat.

   Halaman detail untuk proksi tersebut muncul.
4. Di bagian Titik akhir proksi, pilih Buat titik akhir proksi.

   Jendela Buat titik akhir proksi akan muncul.
5. Untuk Nama titik akhir proksi, masukkan nama deskriptif pilihan Anda.
6. Untuk Peran target, pilih apakah akan membuat titik akhir lintas-VPC bersifat baca/tulis atau hanya-baca.

   Koneksi yang menggunakan endpoint baca/tulis dapat melakukan segala jenis operasi, seperti pernyataan bahasa definisi data (DDL), pernyataan bahasa manipulasi data (DML), dan kueri. Titik akhir ini selalu terhubung ke instance utama cluster DB Aurora. Anda dapat menggunakan titik akhir baca/tulis untuk operasi basis data umum jika Anda hanya menggunakan titik akhir tunggal dalam aplikasi Anda. Anda juga dapat menggunakan endpoint baca/tulis untuk operasi administratif, pemrosesan transaksi online (OLTP) aplikasi, dan extract-transform-load (ETL) pekerjaan.

   Koneksi yang menggunakan titik akhir hanya-baca hanya dapat melakukan kueri. Ketika ada beberapa instance pembaca di cluster AuroraRDS, Proxy dapat menggunakan instance pembaca yang berbeda untuk setiap koneksi ke titik akhir. Dengan begitu, aplikasi intensif kueri dapat memanfaatkan kemampuan pengklasteran Aurora. Anda dapat menambahkan lebih banyak kapasitas kueri ke klaster dengan menambahkan lebih banyak instans pembaca DB. Koneksi hanya-baca ini tidak membebankan overhead apa pun pada instans primer klaster. Dengan begitu, kueri pelaporan dan analisis Anda tidak memperlambat operasi penulisan OLTP aplikasi Anda.

7. Untuk Virtual Private Cloud (VPC), pilih default untuk mengakses titik akhir dari EC2 instance yang sama atau sumber daya lain yang biasanya digunakan untuk mengakses proxy atau database terkait. Untuk mengatur VPC akses silang untuk proxy ini, pilih yang VPC lain selain default. Untuk informasi selengkapnya tentang VPC akses silang, lihatMengakses database RDS di seluruh VPCs.
8. Untuk Subnet, RDS Proxy mengisi subnet yang sama dengan proxy terkait secara default. Untuk membatasi akses ke titik akhir agar hanya sebagian dari rentang alamat yang VPC dapat terhubung dengannya, hapus satu atau beberapa subnet.
9. Untuk grup VPC keamanan, Anda dapat memilih grup keamanan yang ada atau membuat yang baru. RDSProxy mengisi grup atau grup keamanan yang sama dengan proxy terkait secara default. Jika aturan masuk dan keluar proksi sesuai untuk titik akhir ini, pertahankan pilihan default.

   Jika Anda memilih untuk membuat grup keamanan baru, tentukan nama grup keamanan tersebut di halaman ini. Kemudian edit pengaturan grup keamanan dari EC2 konsol nanti.
10. Pilih Buat titik akhir proksi.

AWS CLI

Untuk membuat titik akhir proxy, gunakan AWS CLI create-db-proxy-endpointperintah.

Sertakan parameter-parameter yang diperlukan berikut:

- `--db-proxy-name` *`value`*
- `--db-proxy-endpoint-name` *`value`*
- `--vpc-subnet-ids` *`list_of_ids`*. Pisahkan subnet IDs dengan spasi. Anda tidak menentukan ID itu VPC sendiri.

Anda juga dapat menentukan parameter opsional berikut:

- `--target-role { READ_WRITE | READ_ONLY }`. Pengaturan default parameter ini adalah `READ_WRITE`. Nilai `READ_ONLY` hanya memengaruhi klaster yang disediakan Aurora yang berisi satu atau beberapa instans DB pembaca. Ketika proksi dikaitkan dengan klaster Aurora yang hanya berisi instans DB penulis, Anda tidak dapat menentukan `READ_ONLY`. Untuk informasi selengkapnya tentang tujuan penggunaan titik akhir hanya-baca dengan klaster Aurora, lihat Menggunakan titik akhir pembaca dengan klaster Aurora .

- `--vpc-security-group-ids` *`value`*. Pisahkan grup keamanan IDs dengan spasi. Jika Anda menghilangkan parameter ini, RDS Proxy menggunakan grup keamanan default untuk. VPC RDSProxy menentukan VPC berdasarkan subnet IDs yang Anda tentukan untuk `--vpc-subnet-ids` parameter.

Example

Contoh berikut membuat titik akhir proksi bernama `my-endpoint`.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id ]
```

Untuk Windows:

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id
```

RDS API

Untuk membuat titik akhir proxy, gunakan tindakan RDS API C reateDBProxy Endpoint.

## Melihat titik akhir proksi

Untuk melihat titik akhir proxy yang ada, ikuti petunjuk ini:

Konsol

Untuk melihat detail titik akhir proksi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Proksi.
3. Dalam daftar, pilih proksi yang titik akhirnya ingin dilihat. Klik nama proksi untuk melihat halaman detailnya.
4. Di bagian Titik akhir proksi, pilih titik akhir yang ingin dilihat. Klik namanya untuk melihat halaman detailnya.
5. Periksa parameter yang nilainya Anda minati. Anda dapat mengubah properti seperti berikut ini:
   - Apakah titik akhir ini baca/tulis atau hanya-baca.
   - Alamat titik akhir yang Anda gunakan dalam string koneksi basis data.
   - SubnetVPC, dan kelompok keamanan yang terkait dengan titik akhir.

AWS CLI

Untuk melihat satu atau beberapa titik akhir proxy, gunakan AWS CLI describe-db-proxy-endpointsperintah.

Anda dapat menyertakan parameter opsional berikut:

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

Contoh berikut menjelaskan titik akhir proksi `my-endpoint`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

Untuk Windows:

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

Untuk mendeskripsikan satu atau lebih titik akhir proxy, gunakan operasi RDS API D escribeDBProxy Endpoints.

## Mengubah titik akhir proksi

Untuk mengubah titik akhir proxy Anda, ikuti petunjuk berikut:

Konsol

Untuk mengubah satu atau beberapa titik akhir proksi

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Dalam daftar, pilih proksi yang titik akhirnya ingin Anda ubah. Klik nama proksi untuk melihat halaman detailnya.
4. Di bagian Titik akhir proksi, pilih titik akhir yang ingin Anda ubah. Anda dapat memilihnya dalam daftar, atau mengeklik namanya untuk melihat halaman detail.
5. Di halaman detail proksi, di bagian Titik akhir proksi, pilih Edit. Atau, di halaman detail titik akhir proksi, untuk Tindakan, pilih Edit.
6. Ubah nilai parameter yang ingin dimodifikasi.
7. Pilih Simpan perubahan.

AWS CLI

Untuk memodifikasi titik akhir proxy, gunakan AWS CLI modify-db-proxy-endpointperintah dengan parameter yang diperlukan berikut:

- `--db-proxy-endpoint-name`

Tentukan perubahan ke properti titik akhir dengan menggunakan satu atau beberapa parameter berikut:

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`. Pisahkan grup keamanan IDs dengan spasi.

Contoh berikut mengganti nama titik akhir proksi `my-endpoint` menjadi `new-endpoint-name`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

Untuk Windows:

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

Untuk memodifikasi titik akhir proxy, gunakan operasi RDS API M odifyDBProxy Endpoint.

## Menghapus titik akhir proksi

Untuk menghapus titik akhir proxy Anda, ikuti petunjuk berikut:

> **Note**
>
> Anda tidak dapat menghapus titik akhir proxy default yang dibuat RDS Proxy secara otomatis untuk setiap proxy.
> Saat Anda menghapus proxy, RDS Proxy secara otomatis menghapus semua titik akhir terkait.

Konsol

Untuk menghapus titik akhir proxy menggunakan AWS Management Console

1. Di panel navigasi, pilih Proksi.
2. Dalam daftar, pilih proksi yang titik akhirnya ingin dihapus. Klik nama proksi untuk melihat halaman detailnya.

3. Di bagian Titik akhir proksi, pilih titik akhir yang ingin dihapus. Anda dapat memilih satu atau beberapa titik akhir dalam daftar, atau mengeklik nama titik akhir tunggal untuk melihat halaman detail.
4. Di halaman detail proksi, di bagian Titik akhir proksi, pilih Hapus. Atau, di halaman detail titik akhir proksi, untuk Tindakan, pilih Hapus.

AWS CLI

Untuk menghapus titik akhir proxy, jalankan delete-db-proxy-endpointperintah dengan parameter yang diperlukan berikut:

- `--db-proxy-endpoint-name`

Perintah berikut akan menghapus titik akhir proksi bernama `my-endpoint`.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

Untuk Windows:

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

Untuk menghapus titik akhir proxy dengan RDSAPI, jalankan operasi D eleteDBProxy Endpoint. Tentukan nama titik akhir proksi untuk parameter `DBProxyEndpointName`.

## Memantau metrik Proxy RDS dengan Amazon CloudWatch

Anda dapat memantau Proxy RDS dengan menggunakan Amazon CloudWatch. CloudWatch mengumpulkan dan memproses data mentah dari proxy menjadi metrik yang dapat dibaca. near-real-time Untuk menemukan metrik ini di CloudWatch konsol, pilih Metrik, lalu pilih RDS, dan pilih Metrik Per-Proxy. Untuk informasi selengkapnya, lihat Menggunakan CloudWatch metrik Amazon di Panduan CloudWatch Pengguna Amazon.

> **Note**
>
> RDS menerbitkan metrik ini untuk setiap EC2 instans Amazon yang mendasari yang terkait dengan proxy. Satu proxy dapat dilayani oleh lebih dari satu EC2 instance. Gunakan CloudWatch statistik untuk menggabungkan nilai untuk proxy di semua instance terkait. Beberapa metrik ini mungkin tidak terlihat hingga koneksi berhasil disambungkan terlebih dahulu oleh proksi.

Dalam log Proksi RDS, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini dapat berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna, atau nama khusus `default` untuk titik akhir default proxy yang melakukan permintaan. read/write

Semua metrik Proksi RDS berada di dalam grup `proxy`.

Setiap titik akhir proxy memiliki CloudWatch metriknya sendiri. Anda dapat memantau penggunaan setiap titik akhir proksi secara independen. Untuk informasi selengkapnya tentang titik akhir proksi, lihat Bekerja dengan titik akhir Proksi Amazon RDS.

Anda dapat menggabungkan beberapa nilai untuk setiap metrik menggunakan salah satu kumpulan dimensi berikut. Sebagai contoh, dengan menggunakan kumpulan dimensi `ProxyName`, Anda dapat menganalisis semua lalu lintas untuk proksi tertentu. Dengan menggunakan kumpulan dimensi lainnya, Anda dapat membagi beberapa metrik dengan cara yang berbeda. Anda dapat membagi metrik berdasarkan titik akhir atau basis data target yang berbeda dari setiap proxy, atau lalu lintas read/write dan read-only ke setiap database.

- Kumpulan dimensi 1: `ProxyName`
- Kumpulan dimensi 2: `ProxyName`, `EndpointName`
- Kumpulan dimensi 3: `ProxyName`, `TargetGroup`, `Target`
- Kumpulan dimensi 4: `ProxyName`, `TargetGroup`, `TargetRole`

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
| --- | --- | --- | --- |
| `Availabil ityPercentage` | Persentase waktu saat grup target tersedia dalam peran | 1 menit | Dimension set 4 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
| --- | --- | --- | --- |
| | yang diindikasikan oleh dimensi. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah Sum. | | |
| ClientConnections | Jumlah koneksi klien saat ini. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit | Dimension set 1, Dimension set 2 |
| ClientConnectionsClosed | Jumlah koneksi klien yang ditutup. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit ke atas | Dimension set 1, Dimension set 2 |
| ClientConnectionsInSetup | Jumlah koneksi klien saat ini terbuka tetapi belum menyelesaikan pengaturan. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit | Dimension set 1, Dimension set 2 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
| --- | --- | --- | --- |
| `ClientConnectionsNoTLS` | Jumlah koneksi klien saat ini tanpa Keamanan Lapisan Pengangkutan (TLS). Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit | Dimension set 1, Dimension set 2 |
| `ClientConnectionsReceived` | Jumlah permintaan koneksi klien yang diterima. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 2 |
| `ClientConnectionsSetupFailedAuth` | Jumlah upaya koneksi klien yang gagal karena konfigurasi autentikasi atau TLS yang salah. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 2 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
|---|---|---|---|
| `ClientConnectionsSetupSucceeded` | Jumlah koneksi klien yang berhasil dibangun dengan mekanisme autentikasi apa pun dengan atau tanpa TLS. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 2 |
| `ClientConnectionsTLS` | Jumlah koneksi klien saat ini dengan TLS. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit | Dimension set 1, Dimension set 2 |
| `DatabaseConnectionRequests` | Jumlah permintaan untuk membuat koneksi basis data. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 3, Dimension set 4 |
| `DatabaseConnectionRequestsWithTLS` | Jumlah permintaan untuk membuat koneksi basis data dengan TLS. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 3, Dimension set 4 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
| --- | --- | --- | --- |
| DatabaseConnections | Jumlah koneksi basis data saat ini. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit | Dimension set 1, Dimension set 3, Dimension set 4 |
| DatabaseConnectionsBorrowLatency | Waktu dalam mikrodetik yang dibutuhkan bagi proksi yang dipantau untuk mendapatkan koneksi basis data. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit ke atas | Dimension set 1, Dimension set 2 |
| DatabaseConnectionsCurrentlyBorrowed | Jumlah koneksi basis data saat ini yang berada dalam status meminjam. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit | Dimension set 1, Dimension set 3, Dimension set 4 |
| DatabaseConnectionsCurrentlyInTransaction | Jumlah koneksi basis data saat ini dalam transaksi. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit | Dimension set 1, Dimension set 3, Dimension set 4 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
| --- | --- | --- | --- |
| `DatabaseConnectionsCurrentlySessionPinned` | Jumlah koneksi basis data saat ini yang saat ini disematkan karena adanya operasi dalam permintaan klien yang mengubah status sesi. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit | Dimension set 1, Dimension set 3, Dimension set 4 |
| `DatabaseConnectionsSetupFailed` | Jumlah permintaan koneksi basis data yang gagal. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 3, Dimension set 4 |
| `DatabaseConnectionsSetupSucceeded` | Jumlah koneksi basis data yang berhasil dibangun dengan atau tanpa TLS. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 3, Dimension set 4 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
| --- | --- | --- | --- |
| `DatabaseConnectionsWithTLS` | Jumlah koneksi basis data saat ini dengan TLS. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit | Dimension set 1, Dimension set 3, Dimension set 4 |
| `MaxDatabaseConnectionsAllowed` | Jumlah maksimum koneksi basis data yang diperbolehkan. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit | Dimension set 1, Dimension set 3, Dimension set 4 |
| `QueryDatabaseResponseLatency` | Waktu dalam mikrodetik yang dibutuhkan basis data untuk merespons kueri. Statistik yang paling berguna untuk metrik ini adalah `Average`. | 1 menit ke atas | Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4 |
| `QueryRequests` | Jumlah kueri yang diterima. Kueri yang mencakup beberapa pernyataan yang dihitung sebagai satu kueri. Statistik yang paling berguna untuk metrik ini adalah `Sum`. | 1 menit ke atas | Dimension set 1, Dimension set 2 |

| Metrik | Deskripsi | Periode yang valid | CloudWatch set dimensi |
|---|---|---|---|
| QueryRequestsNoTLS | Jumlah kueri yang diterima dari koneksi non-TLS. Kueri yang mencakup beberapa pernyataan yang dihitung sebagai satu kueri. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit ke atas | Dimension set 1, Dimension set 2 |
| QueryRequestsTLS | Jumlah kueri yang diterima dari koneksi TLS. Kueri yang mencakup beberapa pernyataan yang dihitung sebagai satu kueri. Statistik yang paling berguna untuk metrik ini adalah Sum. | 1 menit ke atas | Dimension set 1, Dimension set 2 |
| QueryResponseLatency | Waktu dalam mikrodetik antara saat mendapatkan permintaan kueri dan saat proksi meresponsnya. Statistik yang paling berguna untuk metrik ini adalah Average. | 1 menit ke atas | Dimension set 1, Dimension set 2 |

Anda dapat menemukan log aktivitas Proxy RDS CloudWatch di bawah. AWS Management Console Setiap proksi memiliki entri di halaman Grup log.

> **Important**
>
> Log ini ditujukan untuk konsumsi manusia untuk tujuan pemecahan masalah dan bukan untuk akses terprogram. Format dan konten log dapat berubah sewaktu-waktu.
> Khususnya, log lama tidak berisi awalan yang menunjukkan titik akhir untuk setiap permintaan. Dalam log yang lebih baru, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini dapat berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna, atau nama khusus `default` untuk permintaan yang menggunakan titik akhir default proksi.

# Bekerja dengan acara RDS Proxy

Peristiwa menunjukkan perubahan dalam lingkungan seperti lingkungan AWS atau layanan atau aplikasi dari mitra perangkat lunak sebagai layanan (SaaS). Atau bisa berupa salah satu aplikasi atau layanan kustom Anda sendiri. Misalnya, Aurora menghasilkan peristiwa saat Anda membuat atau memodifikasi Proxy. RDS Amazon Aurora mengirimkan acara ke Amazon EventBridge dalam waktu hampir nyata. Berikut ini, Anda dapat menemukan daftar acara RDS Proxy yang dapat Anda berlangganan dan contoh acara RDS Proxy.

Berikut informasi selengkapnya tentang cara bekerja dengan peristiwa:

- Untuk petunjuk tentang cara melihat peristiwa dengan menggunakan AWS Management Console, AWS CLI, atau RDSAPI, lihat Melihat RDS acara Amazon.
- Untuk mempelajari cara mengonfigurasi Amazon Aurora untuk mengirim acara EventBridge, lihat. Membuat aturan yang dipicu pada peristiwa Amazon Aurora

## RDSAcara proxy

Tabel berikut menunjukkan kategori acara dan daftar peristiwa ketika RDS Proxy adalah tipe sumber.

| Kategori | RDSID acara | Pesan | Catatan |
| --- | --- | --- | --- |
| perubahan konfigurasi | RDS- EVENT -0204 | RDSproxy DB yang dimodifikasi *name*. | Tidak ada |

| Kategori | RDSID acara | Pesan | Catatan |
| --- | --- | --- | --- |
| perubahan konfigurasi | RDS- EVENT -0207 | RDSmemodifikasi titik akhir proxy DB *name*. | Tidak ada |
| perubahan konfigurasi | RDS- EVENT -0213 | RDSmendeteksi penambahan instans DB dan secara otomatis menambahkannya ke grup target proxy DB *name*. | Tidak ada |
| perubahan konfigurasi | RDS- EVENT -0214 | RDSterdeteksi penghapus an instans DB *name* dan secara otomatis menghapusnya dari grup target *name* dari proxy DB *name*. | Tidak ada |
| perubahan konfigurasi | RDS- EVENT -0215 | RDSterdeteksi penghapus an cluster DB *name* dan secara otomatis menghapusnya dari grup target *name* dari proxy DB *name*. | Tidak ada |
| pembuatan | RDS- EVENT -0203 | RDSdibuat proxy DB *name*. | Tidak ada |
| pembuatan | RDS- EVENT -0206 | RDStitik akhir yang dibuat *name* untuk proxy DB *name*. | Tidak ada |
| penghapusan | RDS- EVENT -0205 | RDSproxy DB dihapus *name*. | Tidak ada |
| penghapusan | RDS- EVENT -0208 | RDStitik akhir dihapus *name* untuk proxy DB *name*. | Tidak ada |

| Kategori | RDSID acara | Pesan | Catatan |
| --- | --- | --- | --- |
| kegagalan | RDS- EVENT -0243 | RDSgagal menyediakan kapasitas untuk proxy *name* karena tidak ada cukup alamat IP yang tersedia di subnet Anda: *name*. Untuk memperbaiki masalah ini, pastikan subnet Anda memiliki jumlah minimum alamat IP yang tidak digunakan seperti yang direkomendasikan dalam dokumentasi RDS Proxy. | Untuk menentukan jumlah yang direkomendasikan untuk kelas instans Anda, lihat Perencanaan untuk kapasitas alamat IP. |
| kegagalan | RDS- EVENT -0275 | RDSmembatasi beberapa koneksi ke proxy DB *name*. Jumlah permintaan koneksi simultan dari klien ke proxy telah melampaui batas. | Tidak ada |

Berikut ini adalah contoh peristiwa RDS Proxy dalam JSON format. Acara menunjukkan bahwa RDS memodifikasi titik akhir bernama `my-endpoint` RDS Proxy bernama`my-rds-proxy`. ID acara adalah RDS - EVENT -0207.

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
```

```
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

## RDSContoh baris perintah proxy

Untuk melihat bagaimana kombinasi perintah koneksi dan SQL pernyataan berinteraksi dengan RDS Proxy, lihat contoh berikut.

Contoh

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example Melestarikan koneksi ke SQL database Saya di seluruh failover

SQLContoh saya ini menunjukkan bagaimana koneksi terbuka terus bekerja selama failover. Contohnya adalah ketika Anda mem-boot ulang basis data atau basis data tidak tersedia karena terjadi masalah. Contoh ini menggunakan proksi bernama `the-proxy` dan klaster DB Aurora dengan instans DB `instance-8898` dan `instance-9814`. Saat Anda menjalankan perintah `failover-db-cluster` dari baris perintah Linux, instans penulis yang terhubung ke proksi berubah ke instans DB yang berbeda. Anda dapat melihat bahwa instans DB yang terkait dengan proksi ini berubah saat koneksi masih terbuka.

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
```

```
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
```

```
+---------------+---------------+
1 row in set (0.02 sec)
```

Example  Menyesuaikan pengaturan max_connections untuk klaster DB Aurora

Contoh ini menunjukkan bagaimana Anda dapat menyesuaikan `max_connections` pengaturan untuk cluster Aurora SQL My DB. Untuk melakukannya, Anda membuat grup parameter cluster DB Anda sendiri berdasarkan pengaturan parameter default untuk cluster yang kompatibel dengan My SQL 5.7. Tentukan nilai untuk pengaturan `max_connections`, dengan mengganti formula yang menetapkan nilai default. Kaitkan grup parameter klaster DB dengan klaster DB Anda.

```
export REGION=us-east-1
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"
```

```
echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"
```

# Pemecahan masalah untuk Proksi RDS

Setelah itu, Anda dapat menemukan ide pemecahan masalah untuk beberapa masalah Proxy RDS umum dan informasi tentang CloudWatch log untuk RDS Proxy.

Dalam log Proksi RDS, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini bisa berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna. Atau, bisa menjadi nama khusus `default` untuk titik akhir default proxy yang melakukan permintaan baca/tulis. Untuk informasi selengkapnya tentang titik akhir proksi, lihat Bekerja dengan titik akhir Proksi Amazon RDS.

Topik

- Memverifikasi konektivitas untuk proksi
- Masalah dan solusi umum
- Memecahkan masalah Proxy RDS dengan RDS untuk MySQL
- Memecahkan masalah Proxy RDS dengan RDS untuk PostgreSQL

## Memverifikasi konektivitas untuk proksi

Anda dapat menggunakan perintah berikut untuk memverifikasi bahwa semua komponen seperti proksi, basis data, dan instans komputasi dalam koneksi dapat berkomunikasi satu sama lain.

Periksa proxy itu sendiri menggunakan describe-db-proxiesperintah. Periksa juga kelompok target terkait menggunakan perintah describe-db-proxy-target-groups. Periksa apakah detail target cocok dengan klaster Aurora yang ingin dikaitkan dengan proksi. Gunakan perintah seperti berikut.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

Untuk mengonfirmasi bahwa proxy dapat terhubung ke database yang mendasarinya, periksa target yang ditentukan dalam grup target menggunakan describe-db-proxy-targetsperintah. Gunakan perintah seperti berikut.

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

Output dari describe-db-proxy-targetsperintah termasuk `TargetHealth` bidang. Anda dapat memeriksa kolom `State`, `Reason`, dan `Description` di dalam `TargetHealth` untuk memeriksa apakah proksi dapat berkomunikasi dengan instans DB acuan.

- Nilai `State` dari `AVAILABLE` mengindikasikan bahwa proksi dapat terhubung ke instans DB.
- Nilai `State` dari `UNAVAILABLE` mengindikasikan masalah koneksi sementara atau permanen. Dalam kasus ini, periksa kolom `Reason` dan `Description`. Misalnya, jika `Reason` memiliki nilai `PENDING_PROXY_CAPACITY`, coba hubungkan lagi setelah proksi menyelesaikan operasi penskalaan. Jika `Reason` memiliki nilai `UNREACHABLE`, `CONNECTION_FAILED`, atau `AUTH_FAILURE`, gunakan penjelasan dari kolom `Description` untuk membantu Anda mendiagnosis masalah.
- Kolom `State` mungkin memiliki nilai `REGISTERING` dalam waktu yang singkat sebelum berubah ke `AVAILABLE` atau `UNAVAILABLE`.

Jika perintah Netcat (`nc`) berikut berhasil, Anda dapat mengakses titik akhir proxy dari EC2 instance atau sistem lain tempat Anda masuk. Perintah ini melaporkan kegagalan jika Anda tidak berada di VPC yang sama dengan proksi dan basis data terkait. Anda mungkin dapat masuk secara langsung ke basis data tanpa berada di VPC yang sama. Akan tetapi, Anda tidak dapat masuk ke proksi kecuali Anda berada di VPC yang sama.

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

Anda dapat menggunakan perintah berikut untuk memastikan bahwa EC2 instance Anda memiliki properti yang diperlukan. Secara khusus, VPC untuk EC2 instance harus sama dengan VPC untuk cluster Aurora instance RDS DB yang terhubung dengan proxy.

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

Periksa rahasia Secrets Manager yang digunakan oleh proksi.

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

Pastikan kolom `SecretString` yang ditampilkan oleh `get-secret-value` dienkode sebagai string JSON yang mencakup kolom `username` dan `password`. Contoh berikut menunjukkan format kolom `SecretString`.

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

## Masalah dan solusi umum

Bagian ini menjelaskan beberapa masalah umum dan solusi potensial saat menggunakan Proksi RDS.

Setelah menjalankan perintah CLI `aws rds describe-db-proxy-targets`, jika deskripsi `TargetHealth` menyatakan `Proxy does not have any registered credentials`, verifikasi hal-hal berikut:

- Ada kredensial yang terdaftar bagi pengguna untuk mengakses proksi.
- Peran IAM untuk mengakses rahasia Secrets Manager yang digunakan oleh proksi valid.

Anda mungkin mengalami peristiwa RDS berikut saat membuat atau terhubung ke proksi DB.

| Kategori | ID peristiwa RDS | Deskripsi |
| --- | --- | --- |
| kegagalan | RDS-EVENT-0243 | RDS gagal menyediakan kapasitas untuk proksi karena tidak ada alamat IP yang cukup yang tersedia di subnet Anda. Untuk memperbai ki masalah ini, pastikan subnet Anda memiliki jumlah minimum alamat IP yang tidak digunakan. Untuk menentuka |

| Kategori | ID peristiwa RDS | Deskripsi |
| --- | --- | --- |
| | | n jumlah yang direkomen dasikan untuk kelas instans Anda, lihat Perencanaan untuk kapasitas alamat IP. |
| kegagalan | RDS-EVENT-0275 | RDS membatasi beberapa koneksi ke proxy DB. *name* Jumlah permintaan koneksi simultan dari klien ke proxy telah melampaui batas. |

Anda mungkin mengalami masalah berikut saat membuat proksi baru atau terhubung ke proksi.

| Kesalahan | Penyebab atau solusi |
| --- | --- |
| `403: The security token included in the request is invalid` | Pilih peran IAM yang sudah ada, bukan memilih untuk membuat peran baru. |

## Memecahkan masalah Proxy RDS dengan RDS untuk MySQL

Anda mungkin mengalami masalah berikut saat terhubung ke proksi MySQL.

| Kesalahan | Penyebab atau solusi |
| --- | --- |
| `ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )` | Tingkat permintaan koneksi dari klien ke proksi telah melampaui batas. |

| Kesalahan | Penyebab atau solusi |
| --- | --- |
| `ERROR 1040 (HY000): IAM authentication rate limit exceeded` | Jumlah permintaan simultan dengan autentikasi IAM dari klien ke proksi telah melampaui batas. |
| `ERROR 1040 (HY000): Number simultaneous connections exceeded (`*`limit_value`*` )` | Jumlah permintaan koneksi simultan dari klien ke proksi telah melampaui batas. |
| `ERROR 1045 (28000): Access denied for user '`*`DB_USER`*`'@'%' (using password: YES)` | Rahasia Secrets Manager yang digunakan oleh proksi tidak cocok dengan nama pengguna dan kata sandi dari pengguna basis data yang sudah ada. Perbarui kredensial dalam rahasia Secrets Manager, atau pastikan pengguna basis data ada dan memiliki kata sandi yang sama dengan yang ada dalam rahasia. |
| `ERROR 1105 (HY000): Unknown error` | Terjadi kesalahan yang tidak diketahui. |
| `ERROR 1231 (42000): Variable ''character_set_client'' can't be set to the value of `*`value`* | Nilai yang diatur untuk parameter `character_set_client` tidak valid. Misalnya, nilai `ucs2` tidak valid karena dapat merusak server MySQL Anda. |

| Kesalahan | Penyebab atau solusi |
| --- | --- |
| `ERROR 3159 (HY000): This RDS Proxy requires TLS connections.` | Anda mengaktifkan pengaturan Wajibkan Keamanan Lapisan Pengangkutan dalam proksi, tetapi koneksi Anda menyertakan parameter `ssl-mode=DISABLED` dalam klien MySQL. Lakukan salah satu dari langkah berikut:<br>• Nonaktifkan pengaturan Wajibkan Keamanan Lapisan Pengangkutan untuk proksi.<br>• Hubungkan ke basis data menggunakan pengaturan minimum `ssl-mode=REQUIRED` dalam klien MySQL. |
| `ERROR 2026 (HY000): SSL connection error: Internal Server Error` | Proses jabat tangan TLS dengan proksi gagal. Beberapa kemungkinan alasannya meliputi:<br>• SSL dibutuhkan, tetapi server tidak mendukungnya.<br>• Terjadi kesalahan server internal.<br>• Terjadi jabat tangan yang buruk. |
| `ERROR 9501 (HY000): Timed-out waiting to acquire database connection` | Waktu tunggu proksi untuk memperoleh koneksi basis data habis. Beberapa kemungkinan alasannya meliputi:<br>• Proksi tidak dapat membangun koneksi basis data karena koneksi maksimum sudah tercapai<br>• Proksi tidak dapat membangun koneksi basis data karena basis data tidak tersedia. |

## Memecahkan masalah Proxy RDS dengan RDS untuk PostgreSQL

Anda mungkin mengalami masalah berikut saat terhubung ke proksi PostgreSQL.

| Kesalahan | Penyebab | Solusi |
| --- | --- | --- |
| `ERROR 28000: IAM authentication is allowed only with SSL connections.` | Pengguna mencoba terhubung ke basis data menggunakan autentikasi IAM dengan pengaturan | Pengguna harus terhubung ke basis data menggunakan pengaturan minimum `sslmode=require` dalam |

| Kesalahan | Penyebab | Solusi |
| --- | --- | --- |
| | `sslmode=disable` dalam klien PostgreSQL. | klien PostgreSQL. Untuk informasi lebih lanjut, lihat dokumentasi Dukungan PostgreSQL SSL. |
| `ERROR 28000: This RDS proxy has no credentials for the role` *`role_name`* `. Check the credentials for this role and try again.` | Tidak ada rahasia Secrets Manager untuk peran ini. | Tambahkan rahasia Secrets Manager untuk peran ini. Untuk informasi selengkapnya, lihat Mengkonfigurasi otentikasi IAM untuk RDS Proxy. |
| `ERROR 28000: RDS supports only IAM, MD5, or SCRAM authentication.` | Klien basis data yang digunakan untuk terhubung ke proksi menggunakan mekanisme autentikasi yang saat ini tidak didukung oleh proksi. | Jika Anda tidak menggunakan otentikasi IAM, gunakan otentikasi kata sandi MD5 atau SCRAM. |
| `ERROR 28000: A user name is missing from the connection startup packet. Provide a user name for this connection.` | Klien basis data yang digunakan untuk terhubung ke proksi tidak mengirimkan nama pengguna saat mencoba membangun koneksi. | Pastikan untuk menentukan sebuah nama pengguna saat menyiapkan koneksi ke proksi menggunakan klien PostgreSQL pilihan Anda. |
| `ERROR 28000: IAM is allowed only with SSL connections.` | Klien mencoba untuk terhubung menggunakan autentikasi IAM, tetapi SSL tidak diaktifkan. | Aktifkan SSL dalam klien PostgreSQL. |

| Kesalahan | Penyebab | Solusi |
| --- | --- | --- |
| `ERROR 28000: This RDS Proxy requires TLS connections.` | Pengguna mengaktifkan opsi Wajibkan Keamanan Lapisan Pengangkutan, tetapi mencoba untuk terhubung dengan `sslmode=disable` dalam klien PostgreSQL. | Untuk memperbaiki kesalahan ini, lakukan salah satu tindakan berikut:<br>• Nonaktifkan opsi Wajibkan Keamanan Lapisan Pengangkutan proksi.<br>• Hubungkan ke basis data menggunakan pengaturan minimum `sslmode=allow` dalam klien PostgreSQL. |
| `ERROR 28P01: IAM authentication failed for user` *`user_name`* `. Check the IAM token for this user and try again.` | Kesalahan ini mungkin terjadi karena alasan berikut:<br>• Klien memberikan nama pengguna IAM yang salah.<br>• Klien memberikan token otorisasi IAM yang salah untuk pengguna.<br>• Klien menggunakan kebijakan IAM yang tidak memiliki izin yang dibutuhkan.<br>• Klien memberikan token otorisasi IAM yang kedaluwarsa untuk pengguna. | Untuk memperbaiki kesalahan ini, lakukan tindakan berikut:<br>1. Pastikan pengguna IAM yang disediakan sudah ada.<br>2. Pastikan token otorisasi IAM adalah milik pengguna IAM yang disediakan.<br>3. Pastikan kebijakan IAM memiliki izin yang memadai untuk RDS.<br>4. Periksa validitas token otorisasi IAM yang digunakan. |
| `ERROR 28P01: The password that was provided for the role` *`role_name`* `is wrong.` | Kata sandi untuk peran ini tidak cocok dengan rahasia Secrets Manager. | Periksa rahasia peran ini dalam Secrets Manager untuk melihat apakah kata sandi sama dengan yang digunakan dalam klien PostgreSQL Anda. |

| Kesalahan | Penyebab | Solusi |
| --- | --- | --- |
| `ERROR 28P01: The IAM authentication failed for the role role_name . Check the IAM token for this role and try again.` | Terjadi masalah dengan token IAM yang digunakan untuk autentikasi IAM. | Buat token autentikasi baru dan gunakan dalam koneksi baru. |
| `ERROR 0A000: Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.` | Klien PostgreSQL yang digunakan untuk terhubung ke proksi menggunakan protokol yang lebih lama dari 3.0. | Gunakan klien PostgreSQL yang lebih baru yang mendukung protokol olah pesan 3.0. Jika Anda menggunakan CLI `psql` PostgreSQL, gunakan versi yang lebih baru atau yang setara dengan 7.4. |
| `ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support streaming replication mode.` | Klien PostgreSQL yang digunakan untuk terhubung ke proksi mencoba menggunakan mode replikasi streaming , yang saat ini tidak didukung oleh Proksi RDS. | Nonaktifkan mode replikasi streaming dalam klien PostgreSQL yang digunakan untuk menghubungkan. |
| `ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support the option option_name .` | Melalui pesan startup, klien PostgreSQL yang digunakan untuk terhubung ke proksi meminta opsi yang saat ini tidak didukung oleh Proksi RDS. | Nonaktifkan opsi yang ditampilkan sebagai tidak didukung dari pesan di atas dalam klien PostgreSQL yang digunakan untuk menghubungkan. |
| `ERROR 53300: The IAM authentication failed because of too many competing requests.` | Jumlah permintaan simultan dengan autentikasi IAM dari klien ke proksi telah melampaui batas. | Kurangi tingkat pembangunan koneksi menggunakan autentikasi IAM dari klien PostgreSQL. |

| Kesalahan | Penyebab | Solusi |
| --- | --- | --- |
| `ERROR 53300: The maximum number of client connections to the proxy exceeded` *`number_value`* `.` | Jumlah permintaan koneksi simultan dari klien ke proksi telah melampaui batas. | Kurangi jumlah koneksi aktif dari klien PostgreSQL ke proksi RDS ini. |
| `ERROR 53300: Rate of connection to proxy exceeded` *`number_value`* `.` | Tingkat permintaan koneksi dari klien ke proksi telah melampaui batas. | Kurangi tingkat pembangunan koneksi dari klien PostgreSQL. |
| `ERROR XX000: Unknown error.` | Terjadi kesalahan yang tidak diketahui. | Hubungi AWS Support untuk menyelidiki masalah ini. |
| `ERROR 08000: Timed-out waiting to acquire database connection.` | Proxy timed-out menunggu untuk memperoleh koneksi database dalam durasi yang ditentukan oleh pengaturan. `ConnectionBorrowTimeout` Beberapa kemungkinan alasannya meliputi:<br>• Proksi tidak dapat membangun koneksi basis data karena koneksi maksimum sudah tercapai.<br>• Proxy tidak dapat membuat koneksi database karena database tidak tersedia atau jika pembentukan koneksi ke database membutuhkan waktu lebih lama dari yang dikonfigurasi`ConnectionBorrowTimeout` . | Kemungkinan solusinya sebagai berikut:<br>• Hindari menyematkan koneksi proxy. Lihat Menghindari menyematkan Proxy RDS.<br>• Tinjau `ConnectionBorrowTimeout` dan `MaxConnectionsPercent` pengaturan. Lihat Pertimbangan koneksi RDS Proxy.<br>• Tinjau ketersediaan target. Lihat `AvailabilityPercentage` diMemantau metrik Proxy RDS dengan Amazon CloudWatch. |

| Kesalahan | Penyebab | Solusi |
| --- | --- | --- |
| `ERROR XX000: Request returned an error: database_error .` | Koneksi basis data yang dibangun dari proksi menampilkan kesalahan. | Solusinya bergantung dari kesalahan basis data tertentu. Salah satu contohnya adalah: `Request returned an error: database "your-database-name" does not exist`. Ini berarti bahwa nama basis data yang ditentukan tidak ada di server basis data. Atau, nama pengguna yang digunakan sebagai sebuah nama basis data (jika nama basis data tidak ditentukan) tidak ada di server. |

### Pemecahan masalah database yang dihapus **`postgres`**

Jika Anda salah menghapus `postgres` database dari instance Anda, Anda perlu mengembalikan database untuk mengembalikan konektivitas ke instance Anda. Jalankan perintah berikut dalam instans DB Anda:

```
CREATE DATABASE postgres;
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

## Penggunaan Proksi RDS dengan AWS CloudFormation

Anda dapat menggunakan Proksi RDS dengan AWS CloudFormation. Hal ini membantu Anda membuat grup sumber daya terkait. Grup semacam ini bisa mencakup proksi yang dapat terhubung ke klaster DB Aurora. Dukungan Proksi RDS dalam AWS CloudFormation melibatkan dua jenis registri baru: `DBProxy` dan `DBProxyTargetGroup`.

Daftar berikut menunjukkan contoh templat AWS CloudFormation untuk Proksi RDS.

```
Resources:
```

```
DBProxy:
  Type: AWS::RDS::DBProxy
  Properties:
    DBProxyName: CanaryProxy
    EngineFamily: MYSQL
    RoleArn:
     Fn::ImportValue: SecretReaderRoleArn
    Auth:
      - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
    VpcSubnetIds:
      Fn::Split: [",", "Fn::ImportValue": SubnetIds]

ProxyTargetGroup:
  Type: AWS::RDS::DBProxyTargetGroup
  Properties:
    DBProxyName: CanaryProxy
    TargetGroupName: default
    DBInstanceIdentifiers:
      - Fn::ImportValue: DBInstanceName
  DependsOn: DBProxy
```

Untuk informasi selengkapnya tentang sumber daya dalam sampel ini, lihat `DBProxy`dan `DBProxyTargetGroup`.

Untuk informasi selengkapnya tentang sumber daya yang dapat dibuat menggunakan AWS CloudFormation, lihat Referensi jenis sumber daya RDS.

## Menggunakan Proksi RDS dengan basis data global Aurora

Basis data global Aurora adalah basis data tunggal yang mencakup beberapa Wilayah AWS, yang memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman tingkat Wilayah. Basis data ini menyediakan toleransi kesalahan bawaan untuk deployment karena instans DB tidak mengandalkan satu Wilayah AWS, tetapi mengandalkan beberapa Wilayah dan Zona Ketersediaan yang berbeda. Untuk informasi selengkapnya, lihat Menggunakan Database Global Amazon Aurora.

Anda dapat menggunakan Proksi RDS dengan klaster DB apa pun dalam basis data global Aurora. Sebelum mulai menggunakan fitur-fitur ini bersama-sama, pastikan Anda benar-benar memahami informasi berikut.

> **Important**
>
> Jika klaster DB adalah bagian dari basis data global dengan penerusan tulis yang diaktifkan, kurangi nilai `MaxConnectionsPercent` proksi dengan kuota yang dialokasikan untuk penerusan tulis. Kuota penerusan tulis diatur dalam parameter klaster DB `aurora_fwd_writer_max_connections_pct`. Untuk informasi tentang penerusan tulis, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

## Batasan untuk Proksi RDS dengan basis data global

Jika penerusan tulis klaster DB Aurora diaktifkan, Proksi RDS tidak mendukung nilai `SESSION` untuk variabel `aurora_replica_read_consistency`. Menetapkan nilai ini dapat menimbulkan perilaku tak terduga.

## Cara kerja titik akhir Proksi RDS dengan basis data global

Jika Anda memahami cara kerja titik akhir Proksi RDS dengan basis data global, Anda akan lebih mudah mengelola aplikasi yang menggunakan basis data Aurora dengan kedua fitur ini.

Untuk proksi dengan klaster primer basis data global sebagai target terdaftar, titik akhir proksi berfungsi seperti halnya dengan klaster DB Aurora apa pun. Titik akhir baca/tulis proksi mengirim semua permintaan ke instans penulis klaster. Titik akhir hanya baca proksi mengirim semua permintaan ke instans pembaca. Jika pembaca tidak tersedia saat koneksi terbuka, Proksi RDS akan mengalihkan kueri berikutnya pada koneksi ke instans pembaca lain. Untuk proksi dengan klaster sekunder sebagai target terdaftar, permintaan yang dikirim ke titik akhir hanya baca proksi juga dikirim ke instans pembaca. Karena klaster tidak memiliki instans penulis, permintaan yang dikirim ke titik akhir baca/tulis gagal dengan kesalahan "`The target group doesn't have any associated read/write instances`".

Operasi failover dan switchover basis data global melibatkan peralihan peran antara klaster DB primer dan salah satu klaster DB sekunder. Ketika klaster sekunder yang dipilih menjadi klaster primer baru, salah satu instans pembacanya ditingkatkan menjadi instans penulis. Instans DB ini kini menjadi instans penulis baru untuk klaster global. Pastikan untuk mengalihkan operasi tulis aplikasi ke titik akhir baca/tulis yang sesuai dari proksi yang terkait dengan klaster primer baru. Titik akhir proksi ini mungkin merupakan titik akhir default atau titik akhir baca/tulis kustom.

Proksi RDS mengantrekan semua permintaan melalui titik akhir baca/tulis dan mengirimkannya ke instans penulis dari klaster primer baru segera setelah tersedia. Ini dilakukan terlepas dari apakah

operasi switchover atau failover telah selesai. Selama switchover atau failover, titik akhir default proksi untuk klaster primer lama masih menerima operasi tulis. Namun, segera setelah klaster itu menjadi klaster sekunder, semua operasi tulis gagal. Untuk mempelajari bagaimana dan kapan harus melakukan tugas switchover atau failover global tertentu, lihat topik berikut:

- Switchover basis data global – Melakukan switchover untuk basis data global Amazon Aurora
- Failover basis data global – Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan

# Integrasi

Ini adalah solusi yang dikelola sepenuhnya untuk membuat data transaksional tersedia di tujuan analitik Anda setelah ditulis ke DB cluster. Extract, transform, and load (ETL) adalah proses menggabungkan data dari berbagai sumber menjadi gudang data sentral yang besar.

Integrasi nol-ETL membuat data dalam DB cluster tersedia di Amazon Redshift atau lakehouse dalam waktu dekat. Amazon SageMaker Setelah data tersebut berada di gudang data target atau data lake, Anda dapat memberi daya pada beban kerja analitik, ML, dan AI Anda menggunakan kemampuan bawaan, seperti pembelajaran mesin, tampilan terwujud, berbagi data, akses gabungan ke beberapa penyimpanan data dan data lake, dan integrasi dengan SageMaker Amazon AI, QuickSight, dan lainnya. Layanan AWS

Untuk membuat integrasi nol-ETL, Anda menentukan DB cluster sebagai sumber, dan gudang data atau lakehouse yang didukung sebagai target. Integrasi mereplikasi data dari database sumber ke gudang data target atau lakehouse.

Diagram berikut menggambarkan fungsi ini untuk integrasi nol-ETL dengan Amazon Redshift:

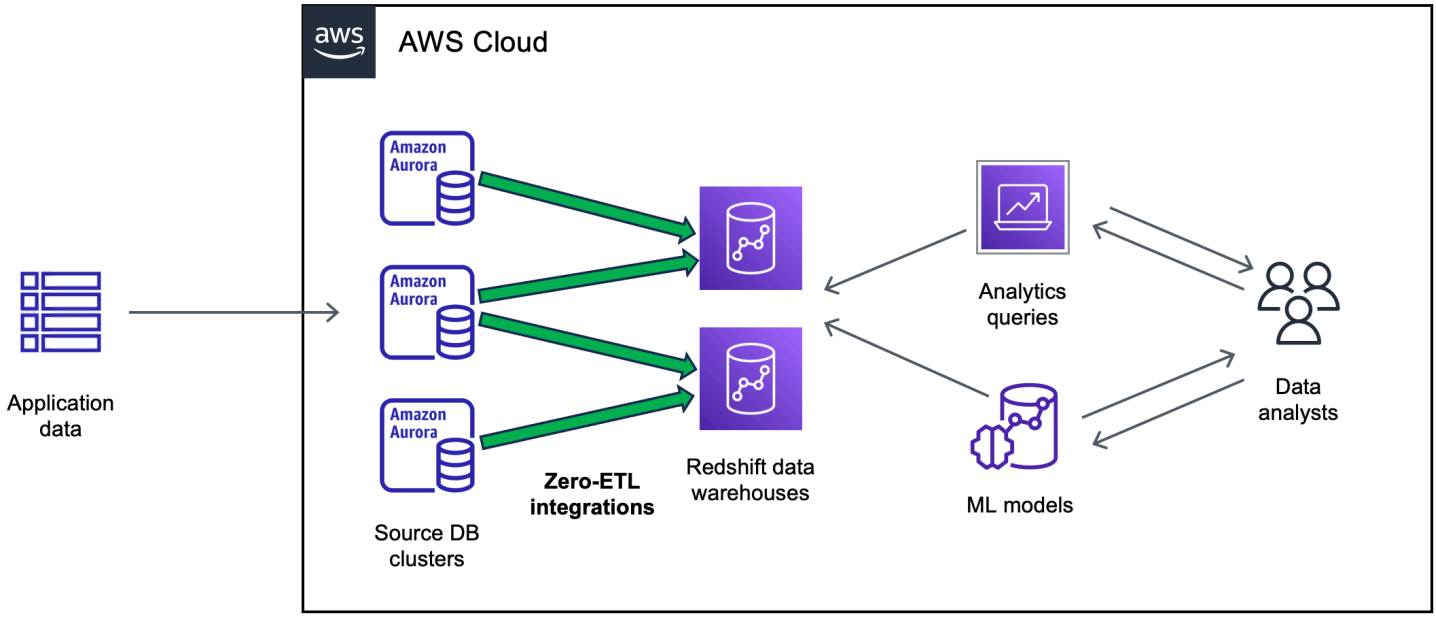
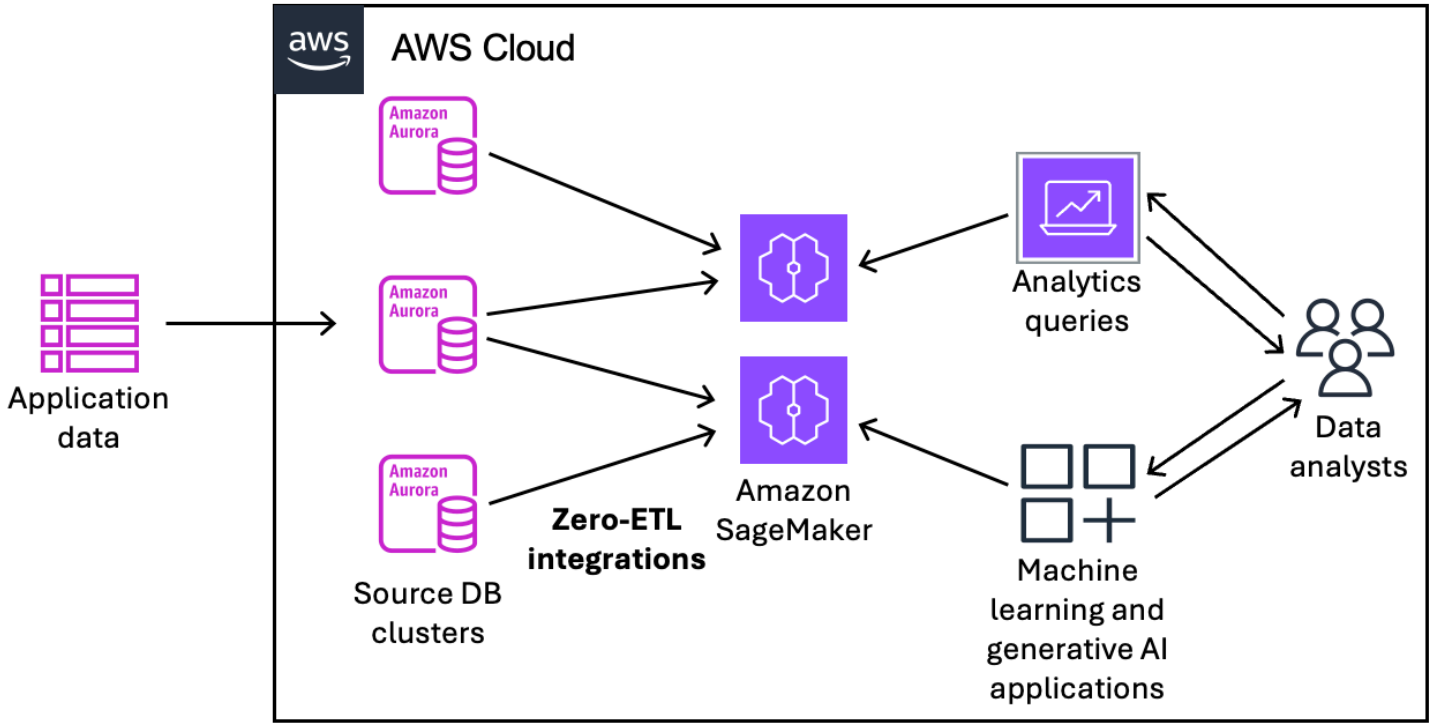



Diagram berikut menggambarkan fungsi ini untuk integrasi nol-ETL dengan rumah danau: Amazon SageMaker




Integrasi memantau kondisi pipeline data dan memulihkan dari masalah jika memungkinkan. Anda dapat membuat integrasi dari beberapa cluster Aurora DB ke dalam satu gudang data target tunggal atau lakehouse yang memungkinkan Anda memperoleh wawasan di beberapa aplikasi.

Untuk informasi tentang harga integrasi nol-ETL, lihat Harga Amazon Aurora dan Harga Amazon Redshift.

Topik


- Manfaat
- Konsep utama
- Batasan
- Kuota
- Wilayah yang Didukung
- Memulai dengan integrasi
- Membuat integrasi nol-ETL Aurora dengan Amazon Redshift
- Membuat integrasi Aurora Zero-ETL dengan rumah danau Amazon SageMaker
- Pemfilteran data untuk integrasi Zero-ETL
- Menambahkan data ke Aurora DB cluster dan menanyakannya
- Melihat dan memantau integrasi

- Memodifikasi integrasi Nol-ETL
- Menghapus integrasi Nol-ETL
- Memecahkan masalah integrasi

## Manfaat

Integrasi Aurora Zero-ETL memiliki manfaat sebagai berikut:

- Membantu Anda memperoleh wawasan menyeluruh dari berbagai sumber data.
- Menghilangkan kebutuhan untuk membangun dan memelihara pipeline data yang kompleks yang melakukan operasi extract, transform, and load (ETL). Integrasi nol-ETL menghilangkan tantangan yang muncul dalam membangun dan mengelola pipeline dengan menyediakan dan mengelolanya untuk Anda.
- Mengurangi beban dan biaya operasional, serta membantu Anda fokus pada peningkatan aplikasi Anda.
- Memungkinkan Anda memanfaatkan analitik tujuan target dan kemampuan ML untuk memperoleh wawasan dari data transaksional dan lainnya, untuk merespons secara efektif peristiwa kritis dan sensitif terhadap waktu.

## Konsep utama

Saat mulai menggunakan integrasi nol-ETL, pertimbangkan konsep berikut ini:

Integrasi

Pipa data yang dikelola sepenuhnya yang secara otomatis mereplikasi data dan skema transaksional dari DB cluster ke gudang data atau katalog.

sumber DB cluster

Aurora DB cluster tempat data direplikasi. Anda dapat menentukan cluster DB yang menggunakan instans DB atau instans DB yang disediakan sebagai Aurora Serverless v2 sumbernya.

Target

Gudang data atau lakehouse tempat data direplikasi. Ada dua jenis gudang data: gudang data klaster terprovisi dan gudang data nirserver. Gudang data klaster terprovisi adalah kumpulan

sumber daya komputasi yang disebut simpul, yang diatur ke dalam grup yang disebut klaster. Gudang data nirserver terdiri dari grup kerja yang menyimpan sumber daya komputasi, serta ruang nama yang menampung objek basis data dan pengguna. Kedua gudang data menjalankan mesin analitik dan berisi satu atau lebih database.

Target lakehouse terdiri dari katalog, database, tabel, dan tampilan. Untuk informasi lebih lanjut tentang arsitektur lakehouse, lihat Amazon SageMaker Lakehouse componentsdi Amazon SageMaker Unified StudioPanduan Pengguna.

Beberapa sumber DB cluster dapat menulis ke target yang sama.

Untuk informasi selengkapnya, lihat Arsitektur sistem gudang data dalam Panduan Developer Amazon Redshift.

# Batasan

Keterbatasan berikut berlaku untuk integrasi .

Topik

- Batasan umum
- Batasan Aurora MySQL
- Keterbatasan Aurora PostgreSQL
- Batasan Amazon Redshift
- Amazon SageMakerketerbatasan lakehouse

## Batasan umum

- Cluster DB sumber harus berada di Wilayah yang sama dengan target.
- Anda tidak dapat mengganti nama cluster DB atau instance-nya jika memiliki integrasi yang ada.
- Anda tidak dapat membuat beberapa integrasi antara database sumber dan target yang sama.
- Anda tidak dapat menghapus cluster DB yang memiliki integrasi yang ada. Anda harus menghapus semua integrasi yang terkait terlebih dahulu.
- Jika Anda menghentikan cluster DB sumber, beberapa transaksi terakhir mungkin tidak direplikasi ke target sampai Anda melanjutkan cluster .

- Jika kluster Anda adalah sumber penerapan biru/hijau, lingkungan biru dan hijau tidak dapat memiliki integrasi nol-ETL selama peralihan. Anda harus menghapus integrasi tersebut terlebih dahulu dan beralih, lalu membuat ulang integrasi.
- Cluster DB harus berisi setidaknya satu instans DB untuk menjadi sumber integrasi.
- Anda tidak dapat membuat integrasi untuk cluster DB sumber yang merupakan klon lintas akun, seperti yang dibagikan menggunakan AWS Resource Access Manager ()AWS RAM.
- Jika klaster sumber Anda adalah klaster DB primer dalam basis data global Aurora dan melakukan failover ke salah satu klaster sekundernya, integrasi menjadi tidak aktif. Anda harus menghapus dan membuat ulang integrasi.
- Anda tidak dapat membuat integrasi untuk database sumber yang memiliki integrasi lain yang sedang dibuat secara aktif.
- Saat Anda pertama kali membuat integrasi, atau ketika tabel sedang disinkronkan ulang, seeding data dari sumber ke target dapat memakan waktu 20-25 menit atau lebih tergantung ukuran basis data sumber. Penundaan ini dapat menyebabkan peningkatan lag replika.
- Beberapa jenis data tidak didukung. Untuk informasi selengkapnya, lihat the section called “Perbedaan jenis data”.
- Tabel sistem, tabel sementara, dan tampilan tidak direplikasi ke gudang target.
- `ALTER TABLE`operasi partisi menyebabkan tabel Anda melakukan sinkronisasi ulang untuk memuat ulang data dari Aurora ke tujuan analitik. Tabel tidak akan tersedia untuk kueri saat disinkronkan ulang. Untuk informasi selengkapnya, lihat the section called “Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang”.

## Batasan Aurora MySQL

- Cluster DB sumber Anda harus menjalankan versi Aurora MySQL yang didukung. Untuk daftar versi yang didukung, lihat the section called “Integrasi nol-ETL ”.
- Integrasi nol-ETL mengandalkan pencatatan log biner MySQL (binlog) untuk mengambil perubahan data yang sedang berlangsung. Jangan gunakan pemfilteran data berbasis binlog, karena dapat menyebabkan inkonsistensi data antara basis data sumber dan target.
- Integrasi nol-ETL didukung hanya untuk basis data yang dikonfigurasi untuk menggunakan mesin penyimpanan InnoDB.
- Referensi kunci asing dengan pembaruan tabel yang telah ditentukan sebelumnya tidak didukung. Secara khusus, `ON DELETE` dan `ON UPDATE` aturan tidak didukung dengan`CASCADE`,`SET NULL`,

dan `SET DEFAULT` tindakan. Mencoba membuat atau memperbarui tabel dengan referensi tersebut ke tabel lain akan menempatkan tabel ke dalam keadaan gagal.

- Transaksi XA yang dilakukan pada cluster DB sumber menyebabkan integrasi memasuki status. `Syncing`

## Keterbatasan Aurora PostgreSQL

- Cluster DB sumber Anda harus menjalankan versi Aurora PostgreSQL yang didukung. Untuk daftar versi yang didukung, lihat the section called “Integrasi nol-ETL ”.
- Jika Anda memilih cluster DB sumber PostgreSQL Aurora, Anda harus menentukan setidaknya satu pola filter data. Minimal, pola harus menyertakan database tunggal (*`database-name`*`.*.*`) untuk replikasi ke gudang target. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.
- Semua basis data yang dibuat dalam klaster DB Aurora PostgreSQL sumber harus menggunakan pengenkodean UTF-8.
- Jika Anda melakukan transaksi partisi deklaratif pada cluster DB sumber, semua tabel yang terpengaruh memasukkan status gagal dan tidak lagi dapat diakses.
- Transaksi dua fase tidak didukung.
- Jika Anda menghapus semua instance DB dari cluster DB yang merupakan sumber integrasi dan kemudian menambahkan kembali instance DB, replikasi akan terputus antara sumber dan kluster target.
- Cluster DB sumber tidak dapat menggunakan Aurora Limitless Database.

## Batasan Amazon Redshift

Untuk daftar batasan Amazon Redshift yang terkait dengan integrasi Nol-ETL, lihat Pertimbangan saat menggunakan integrasi Nol-ETL dengan Amazon Redshift di Panduan Manajemen Pergeseran Merah Amazon.

## Amazon SageMakerketerbatasan lakehouse

Berikut ini adalah batasan untuk integrasi Amazon SageMaker lakehouse zero-ETL.

- Nama katalog dibatasi hingga 19 karakter panjangnya.

# Kuota

Akun Anda memiliki kuota berikut yang terkait dengan integrasi Zero-ETL. Kecuali ditentukan lain, masing-masing kuota ditentukan untuk setiap Wilayah.

| Nama | Default | Deskripsi |
|---|---|---|
| Integrasi | 100 | Jumlah total integrasi dalam sebuah Akun AWS. |
| Integrasi per target | 50 | Jumlah integrasi yang mengirimkan data ke gudang data target tunggal atau lakehouse. |
| Integrasi per klaster sumber | 5 | Jumlah integrasi yang mengirimkan data dari klaster DB sumber tunggal. |

Selain itu, gudang target menempatkan batasan tertentu pada jumlah tabel yang diizinkan di setiap instance DB atau node cluster. Untuk informasi selengkapnya tentang kuota dan batas Amazon Redshift, lihat Kuota dan batas di Amazon Redshift di Panduan Manajemen Pergeseran Merah Amazon.

# Wilayah yang Didukung

Integrasi Aurora Zero-ETL tersedia dalam subset. Wilayah AWS Untuk mengetahui daftar Wilayah yang didukung, lihat the section called “Integrasi nol-ETL ”.

# Memulai dengan integrasi

Sebelum Anda membuat integrasi nol-ETL, konfigurasikan DB cluster dan gudang data Anda dengan parameter dan izin yang diperlukan. Selama pengaturan, Anda akan menyelesaikan langkah-langkah berikut:

1. Buat grup parameter klaster DB kustom.
2. Buat sumber DB cluster.
3. Buat gudang data target untuk Amazon Redshift atau Buat rumah danau target Amazon SageMaker.

Setelah Anda menyelesaikan tugas-tugas ini, lanjutkan ke the section called “Membuat integrasi Nol-ETL dengan Amazon Redshift” atauthe section called “Membuat integrasi nol-ETL dengan rumah danau Amazon SageMaker”.

Anda dapat menggunakan AWS SDKs untuk mengotomatiskan proses penyiapan untuk Anda. Untuk informasi selengkapnya, lihat the section called “Siapkan integrasi menggunakan AWS SDKs”.

> **Tip**
>
> Anda dapat meminta RDS menyelesaikan langkah-langkah pengaturan ini untuk Anda saat Anda membuat integrasi, daripada melakukannya secara manual. Untuk segera mulai membuat integrasi, lihatthe section called “Membuat integrasi Nol-ETL dengan Amazon Redshift”.

Untuk Langkah 3, Anda dapat memilih untuk membuat gudang data target (Langkah 3a) atau target lakehouse (Langkah 3b) tergantung pada kebutuhan Anda:

- Pilih gudang data jika Anda memerlukan kemampuan pergudangan data tradisional dengan analitik berbasis SQL.
- Pilih Amazon SageMaker rumah danau jika Anda membutuhkan kemampuan pembelajaran mesin dan ingin menggunakan fitur lakehouse untuk ilmu data dan alur kerja ML.

## Langkah 1: Buat grup parameter klaster DB kustom

Integrasi Aurora Zero-ETL memerlukan nilai khusus untuk parameter cluster DB yang mengontrol replikasi. Secara khusus, Aurora MySQL memerlukan binlog yang ditingkatkan (`aurora_enhanced_binlog`), dan Aurora PostgreSQL memerlukan replikasi logis yang ditingkatkan (`aurora.enhanced_logical_replication`).

Untuk mengonfigurasi pencatatan log biner atau replikasi logis, Anda harus membuat grup parameter klaster DB kustom terlebih dahulu, lalu mengaitkannya dengan klaster DB sumber.

Aurora MySQL (rangkaian aurora-mysql8.0):

- `aurora_enhanced_binlog=1`
- `binlog_backup=0`
- `binlog_format=ROW`

- `binlog_replication_globaldb=0`
- `binlog_row_image=full`
- `binlog_row_metadata=full`

Selain itu, pastikan bahwa parameter `binlog_transaction_compression` tidak ditetapkan ke `ON`, dan bahwa parameter `binlog_row_value_options` tidak diatur ke `PARTIAL_JSON`.

Untuk informasi selengkapnya tentang binlog yang ditingkatkan Aurora MySQL, lihat the section called “Menyiapkan binlog yang ditingkatkan”.

Aurora PostgreSQL (keluarga aurora-postgresql16):

- `rds.logical_replication=1`
- `aurora.enhanced_logical_replication=1`
- `aurora.logical_replication_backup=0`
- `aurora.logical_replication_globaldb=0`

Mengaktifkan replikasi logis yang disempurnakan (`aurora.enhanced_logical_replication`) akan selalu menulis semua nilai kolom ke log tulis depan (WAL) meskipun `REPLICA IDENTITY FULL` tidak diaktifkan. Ini mungkin meningkatkan IOPS untuk cluster DB sumber Anda.

> **Important**
>
> Jika Anda mengaktifkan atau menonaktifkan parameter cluster `aurora.enhanced_logical_replication` DB, instans DB primer membatalkan semua slot replikasi logis. Ini menghentikan replikasi dari sumber ke target, dan Anda harus membuat ulang slot replikasi pada instance DB utama. Untuk mencegah interupsi, jaga agar status parameter tetap konsisten selama replikasi.

## Langkah 2: Pilih atau buat cluster DB sumber

Setelah Anda membuat grup parameter cluster DB kustom, pilih atau buat Aurora DB cluster. Cluster ini akan menjadi sumber replikasi data ke gudang data target. Anda dapat menentukan cluster DB yang menggunakan instans DB atau instans DB yang disediakan sebagai Aurora Serverless v2 sumbernya. Untuk instruksi untuk membuat cluster , lihat the section called “Membuat klaster DB”atau. the section called “Membuat Aurora Serverless v2 klaster DB.”

Database harus menjalankan versi mesin DB yang didukung. Untuk daftar versi yang didukung, lihat the section called “Integrasi nol-ETL ”.

Saat Anda membuat database, di bawah Konfigurasi tambahan, ubah grup parameter cluster DB default ke grup parameter kustom yang Anda buat di langkah sebelumnya.

> **Note**
>
> Jika Anda mengaitkan grup parameter dengan cluster DB setelah cluster sudah dibuat, Anda harus me-reboot instance DB utama di cluster untuk menerapkan perubahan sebelum Anda dapat membuat integrasi nol-ETL. Untuk petunjuk, lihat the section called “Mem-boot ulang instans atau klaster DB Aurora”.

## Langkah 3a: Buat gudang data target

Setelah Anda membuat cluster DB sumber Anda, Anda harus membuat dan mengkonfigurasi gudang data target. Gudang data harus memenuhi persyaratan berikut:

- Menggunakan tipe RA3 node dengan setidaknya dua node, atau Redshift Serverless.
- Terenkripsi (jika menggunakan klaster yang disediakan). Untuk informasi selengkapnya, lihat Enkripsi basis data Amazon Redshift.

Untuk petunjuk cara membuat gudang data, lihat Membuat klaster untuk klaster terprovisi, atau Membuat grup kerja dengan ruang nama untuk Redshift Nirserver.

### Aktifkan kepekaan huruf besar/kecil di gudang data

Agar integrasi berhasil, parameter kepekaan huruf besar/kecil (`enable_case_sensitive_identifier`) harus diaktifkan untuk gudang data. Secara default, kepekaan huruf besar/kecil dinonaktifkan di semua klaster terprovisi dan grup kerja Redshift Nirserver.

Untuk mengaktifkan kepekaan huruf besar/kecil, lakukan langkah-langkah berikut bergantung pada jenis gudang data Anda:

- Klaster terprovisi – Untuk mengaktifkan kepekaan huruf besar/kecil pada klaster terprovisi, buat grup parameter kustom dengan parameter `enable_case_sensitive_identifier` diaktifkan.

Kemudian, hubungkan grup parameter dengan klaster. Untuk petunjuknya, lihat Mengelola grup parameter menggunakan konsol atau Mengonfigurasi nilai parameter menggunakan AWS CLI.

> **Note**
>
> Ingatlah untuk mem-boot ulang klaster setelah Anda mengaitkan grup parameter kustom dengannya.

- Grup kerja Nirserver – Untuk mengaktifkan kepekaan huruf besar/kecil di grup kerja Redshift Nirserver, Anda harus menggunakan AWS CLI. Konsol Amazon Redshift saat ini tidak mendukung modifikasi nilai parameter Redshift Nirserver. Kirim permintaan update-workgroup berikut:

```
aws redshift-serverless update-workgroup \
  --workgroup-name target-workgroup \
  --config-parameters
 parameterKey=enable_case_sensitive_identifier,parameterValue=true
```

Anda tidak perlu mem-boot ulang grup kerja setelah Anda mengubah nilai parameternya.

### Konfigurasikan otorisasi untuk gudang data

Setelah Anda membuat gudang data, Anda harus mengkonfigurasi sumber Aurora DB cluster sebagai sumber integrasi resmi. Untuk petunjuknya, lihat Mengonfigurasi otorisasi untuk gudang data Amazon Redshift Anda.

## Siapkan integrasi menggunakan AWS SDKs

Daripada menyiapkan setiap sumber daya secara manual, Anda dapat menjalankan skrip Python berikut untuk menyiapkan sumber daya yang diperlukan secara otomatis. Contoh kode menggunakan AWS SDK untuk Python (Boto3)untuk membuat sumber Amazon Aurora DB cluster dan gudang data target, masing-masing dengan nilai parameter yang diperlukan. Kemudian menunggu database tersedia sebelum membuat integrasi nol-ETL di antara mereka. Anda dapat menerapkan "comment out" pada berbagai fungsi tergantung pada sumber daya yang perlu Anda atur.

Untuk menginstal dependensi yang diperlukan, jalankan perintah berikut:

```
pip install boto3
pip install time
```

Dalam skrip, secara opsional, Anda dapat memodifikasi nama grup sumber, target, dan parameter. Fungsi akhir akan membuat integrasi bernama `my-integration` setelah sumber daya disiapkan.

Contoh kode Python

Aurora MySQL

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL binary logging'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
```

```
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']
['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)
```

```
    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
```

```
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']
```

```
    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

### Aurora PostgreSQL

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora PostgreSQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-postgresql16',
        Description='For Aurora PostgreSQL logical replication'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'rds.logical_replication',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.enhanced_logical_replication',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
```

```
            {
                'ParameterName': 'aurora.logical_replication_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.logical_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-postgresql',
        EngineVersion='16.4.aurora-postgresql',
        DatabaseName='mypostgresdb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']
['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-postgresql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora PostgreSQL zero-ETL integrations'
```

```
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
```

```
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
```

```
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

## Langkah 3b: Buat AWS Glue katalog untuk integrasi Amazon SageMaker nol-ETL

Saat membuat integrasi nol-ETL dengan Amazon SageMaker lakehouse, Anda harus membuat katalog terkelola di. AWS Glue AWS Lake Formation Katalog target harus berupa katalog terkelola Amazon Redshift. Untuk membuat katalog terkelola Amazon Redshift, pertama-tama buat peran terkait `AWSServiceRoleForRedshift` layanan. Di konsol Lake Formation, tambahkan `AWSServiceRoleForRedshift` sebagai administrator hanya-baca.

Untuk informasi selengkapnya tentang tugas sebelumnya, lihat topik berikut.

- Untuk informasi tentang membuat katalog terkelola Amazon Redshift, lihat Membuat katalog terkelola Amazon Redshift di Panduan AWS Glue Data Catalog Pengembang.AWS Lake Formation

- Untuk informasi tentang peran terkait layanan untuk Amazon Redshift, lihat Menggunakan peran terkait layanan untuk Amazon Redshift di Panduan Manajemen Pergeseran Merah Amazon.
- Untuk informasi tentang izin administrator hanya-baca untuk Lake Formation, lihat personas Lake Formation dan referensi izin IAM di Panduan Pengembang.AWS Lake Formation

## Konfigurasikan izin untuk katalog target AWS Glue

Sebelum membuat katalog target untuk integrasi nol-ETL, Anda harus membuat peran pembuatan target Lake Formation dan peran transfer data. AWS Glue Gunakan peran pembuatan target Lake Formation untuk membuat katalog target. Saat membuat katalog target, masukkan peran transfer data Glue di bidang peran IAM di bagian Access from engine.

Peran penciptaan target Lake Formation

Peran pembuatan target haruslah administrator Lake Formation dan memerlukan izin berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lakeformation:RegisterResource",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutEncryptionConfiguration",
                "iam:PassRole",
                "glue:CreateCatalog",
                "glue:GetCatalog",
                "s3:PutBucketTagging",
                "s3:PutLifecycleConfiguration",
                "s3:PutBucketPolicy",
                "s3:CreateBucket",
                "redshift-serverless:CreateNamespace",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "redshift-serverless:CreateWorkgroup"
            ],
```

```
            "Resource": [
                "arn:aws:glue:*:account-id:catalog",
                "arn:aws:glue:*:account-id:catalog/*",
                "arn:aws:s3:::*",
                "arn:aws:redshift-serverless:*:account-id:workgroup/*",
                "arn:aws:redshift-serverless:*:account-id:namespace/*",
                "arn:aws:iam::account-id:role/GlueDataCatalogDataTransferRole"
            ]
        }
    ]
}
```

Peran penciptaan target harus memiliki hubungan kepercayaan berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::account-id:user/Username"
          },
          "Action": "sts:AssumeRole"
        }
    ]
}
```

Glue peran transfer data

Peran transfer data Glue diperlukan untuk operasi katalog MySQL dan harus memiliki izin berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DataTransferRolePolicy",
```

```
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "glue:GetCatalog",
                "glue:GetDatabase"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Peran transfer data Glue harus memiliki hubungan kepercayaan berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "glue.amazonaws.com",
                    "redshift.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

## Langkah selanjutnya

Dengan sumber cluster Aurora DB dan gudang data target Amazon Redshift Amazon SageMaker atau lakehouse, Anda dapat membuat integrasi nol-ETL dan mereplikasi data. Untuk petunjuk, lihat the section called “Membuat integrasi Nol-ETL dengan Amazon Redshift”.

# Membuat integrasi nol-ETL Aurora dengan Amazon Redshift

Saat membuat integrasi Aurora Zero-ETL, Anda menentukan sumber Aurora DB cluster dan gudang data Amazon Redshift target. Anda juga dapat menyesuaikan pengaturan enkripsi dan

menambahkan tag. Aurora menciptakan integrasi antara cluster DB basis data dan targetnya. Setelah integrasi aktif, data apa pun yang Anda masukkan ke dalam cluster DB sumber akan direplikasi ke target Amazon Redshift yang dikonfigurasi.

## Prasyarat

Sebelum membuat integrasi nol-ETL, Anda harus membuat cluster DB sumber dan gudang data Amazon Redshift target. Anda juga harus mengizinkan replikasi ke gudang data dengan menambahkan cluster DB sebagai sumber integrasi resmi.

Untuk petunjuk cara menyelesaikan setiap langkah ini, lihat the section called “Mulai menggunakan integrasi nol-ETL”.

## Izin yang diperlukan

Izin IAM tertentu diperlukan untuk membuat integrasi nol-ETL. Setidaknya, Anda memerlukan izin untuk melakukan tindakan berikut:

- 
- Lihat dan hapus semua integrasi nol-ETL.
- Buat integrasi masuk ke gudang data target.

Contoh kebijakan berikut menunjukkan izin hak istimewa paling sedikit yang diperlukan untuk membuat dan mengelola integrasi. Anda mungkin tidak memerlukan izin persis ini jika pengguna atau peran Anda memiliki izin yang lebih luas, seperti kebijakan terkelola `AdministratorAccess`.

> **Note**
>
> Redshift Amazon Resource Names (ARNs) memiliki format berikut. Perhatikan penggunaan garis miring ke depan (`/`), bukan titik dua (`:`), sebelum UUID ruang nama nirserver.
>
> - Klaster terprovisi – `arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid`
> - Nirserver – `arn:aws:redshift-serverless:{region}:{account-id}:namespace/namespace-uuid`

Contoh kebijakan untuk target Redshift

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "rds:CreateIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:cluster:source-db",
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DescribeIntegrations"
        ],
        "Resource": ["*"]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DeleteIntegration",
            "rds:ModifyIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "redshift:CreateInboundIntegration"
        ],
        "Resource": [
            "arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid"
        ]
    }]
```

```
}
```

## Memilih gudang data target di akun yang berbeda

Jika Anda berencana untuk menentukan gudang data Amazon Redshift target yang ada di gudang lain Akun AWS, Anda harus membuat peran yang memungkinkan pengguna di akun saat ini mengakses sumber daya di akun target. Untuk informasi selengkapnya, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki.

Peran harus memiliki izin berikut, yang memungkinkan pengguna melihat klaster terprovisi Amazon Redshift dan ruang nama Redshift Nirserver di akun target.

Izin dan kebijakan kepercayaan yang diperlukan

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Peran harus memiliki kebijakan kepercayaan berikut ini, yang menentukan ID akun target.

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

Untuk petunjuk cara membuat peran, lihat Membuat peran menggunakan kebijakan kepercayaan kustom.

## Membuat integrasi nol-ETL

Anda dapat membuat integrasi nol-ETL menggunakan API AWS Management Console, the AWS CLI, atau RDS.

> **Important**
>
> Integrasi nol-ETL tidak mendukung operasi penyegaran atau sinkronisasi ulang. Jika Anda mengalami masalah dengan integrasi setelah pembuatan, Anda harus menghapus integrasi dan membuat yang baru.

Konsol RDS

Untuk membuat integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Integrasi nol-ETL.
3. Pilih Buat integrasi nol-ETL.
4. Untuk Pengidentifikasi integrasi, masukkan nama untuk integrasi. Nama dapat memiliki maksimal 63 karakter alfanumerik dan dapat menyertakan tanda hubung.

> **Important**
>
> Nama katalog dibatasi hingga 19 karakter panjangnya. Pastikan pengenal integrasi Anda memenuhi persyaratan ini jika akan digunakan sebagai nama katalog.

5. Pilih Berikutnya.
6. Untuk Sumber, pilih Aurora DB cluster tempat data akan berasal.

> **Note**
>
> RDS memberi tahu Anda jika parameter cluster DB tidak dikonfigurasi dengan benar. Jika Anda menerima pesan ini, Anda dapat memilih Perbaiki untuk saya, atau mengonfigurasinya secara manual. Untuk petunjuk cara memperbaikinya secara manual, lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”. Memodifikasi parameter klaster DB membutuhkan boot ulang. Sebelum Anda dapat membuat integrasi, reboot harus lengkap dan nilai parameter baru harus berhasil diterapkan ke cluster .

7. (Opsional) Pilih Sesuaikan opsi pemfilteran data dan tambahkan filter data ke integrasi Anda. Anda dapat menggunakan filter data untuk menentukan ruang lingkup replikasi ke gudang data target. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.
8. Setelah cluster DB sumber Anda berhasil dikonfigurasi, pilih Berikutnya.
9. Untuk Target, lakukan hal berikut:

    1. (Opsional) Untuk menggunakan target Amazon Redshift yang berbeda Akun AWS , pilih Tentukan akun yang berbeda. Kemudian, masukkan ARN peran IAM dengan izin untuk menampilkan gudang data Anda. Untuk petunjuk cara membuat peran IAM, lihat the section called “Memilih gudang data target di akun yang berbeda”.
    2. Untuk gudang data Amazon Redshift, pilih target untuk data yang direplikasi dari cluster DB sumber. Anda dapat memilih klaster Amazon Redshift terprovisi atau ruang nama Redshift Nirserver sebagai target.

> **Note**
>
> RDS akan memberi tahu Anda jika pengaturan kebijakan sumber daya atau kepekaan huruf besar/kecil untuk gudang data yang ditentukan tidak dikonfigurasi dengan benar. Jika Anda menerima pesan ini, Anda dapat memilih Perbaiki untuk saya, atau mengonfigurasinya secara manual. Untuk petunjuk cara memperbaikinya secara manual, lihat Mengaktifkan kepekaan huruf besar/kecil untuk gudang data Anda dan Mengonfigurasi otorisasi untuk gudang data Anda dalam Panduan Manajemen Amazon Redshift.
> Modifikasi kepekaan huruf besar/kecil untuk klaster Redshift terprovisi memerlukan boot ulang. Sebelum Anda dapat membuat integrasi, boot ulang harus selesai dan nilai parameter baru harus berhasil diterapkan ke klaster.
> Jika sumber dan target yang Anda pilih berada di Akun AWS yang berbeda, maka Amazon RDS tidak dapat memperbaiki pengaturan ini untuk Anda. Anda harus menavigasi ke akun lain dan memperbaikinya secara manual di Amazon Redshift.

10. Setelah gudang data target Anda dikonfigurasi dengan benar, pilih Berikutnya.
11. (Opsional) Untuk Tag, tambahkan satu atau beberapa tag ke integrasi. Untuk informasi selengkapnya, lihat the section called “Menandai RDSAurora dan sumber daya”.
12. Untuk Enkripsi, tentukan cara enkripsi integrasi Anda. Secara default, RDS mengenkripsi semua integrasi dengan file. Kunci milik AWS Untuk memilih kunci yang dikelola pelanggan, aktifkan Sesuaikan pengaturan enkripsi dan pilih kunci KMS yang akan digunakan untuk enkripsi. Untuk informasi selengkapnya, lihat the section called “Mengenkripsi sumber daya Amazon Aurora”.

    Secara opsional, tambahkan konteks enkripsi. Untuk informasi lebih lanjut, lihat Konteks enkripsi di Panduan Developer AWS Key Management Service .

    > **Note**
    >
    > Amazon RDS menambahkan pasangan konteks enkripsi berikut selain yang Anda tambahkan:
    >
    > - `aws:redshift:integration:arn` - `IntegrationArn`
    > - `aws:servicename:id` - `Redshift`

Ini mengurangi jumlah keseluruhan pasangan yang dapat Anda tambahkan dari 8 menjadi 6, dan berkontribusi pada batas karakter keseluruhan dari batasan hibah. Untuk informasi selengkapnya, lihat Menggunakan batasan hibah di Panduan Pengembang.AWS Key Management Service

13. Pilih Selanjutnya.
14. Tinjau pengaturan integrasi Anda dan pilih Buat integrasi nol-ETL.

    Jika pembuatan gagal, lihat the section called “Saya tidak dapat membuat integrasi nol-ETL” untuk langkah-langkah pemecahan masalah.

Integrasi memiliki status `Creating` ketika sedang dibuat dan gudang data Amazon Redshift target memiliki status `Modifying`. Selama waktu ini, Anda tidak dapat mengueri gudang data atau membuat perubahan konfigurasi apa pun di dalamnya.

Ketika integrasi berhasil dibuat, status integrasi dan gudang data Amazon Redshift target berubah menjadi `Active`.

AWS CLI

Untuk membuat integrasi nol-ETL menggunakan AWS CLI, gunakan perintah create-integration dengan opsi berikut:

> **Note**
>
> Ingat bahwa nama katalog dibatasi hingga 19 karakter. Pilih nama integrasi Anda sesuai jika akan digunakan sebagai nama katalog.

- `--integration-name` – Tentukan nama untuk integrasi.
- `--source-arn`— Tentukan ARN dari DB cluster yang akan menjadi sumber untuk integrasi.
- `--target-arn` – Tentukan ARN gudang data Amazon Redshift yang akan menjadi target integrasi.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

Untuk Windows:

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

API RDS

Untuk membuat integrasi nol-ETL menggunakan API Amazon RDS, gunakan operasi `CreateIntegration` dengan parameter berikut ini:

> **Note**
>
> Nama katalog dibatasi hingga 19 karakter. Pastikan IntegrationName parameter Anda memenuhi persyaratan ini jika akan digunakan sebagai nama katalog.

- `IntegrationName` – Tentukan nama untuk integrasi.
- `SourceArn`— Tentukan ARN dari DB cluster yang akan menjadi sumber untuk integrasi.
- `TargetArn` – Tentukan ARN gudang data Amazon Redshift yang akan menjadi target integrasi.

## Mengenkripsi integrasi dengan kunci yang dikelola pelanggan

Jika Anda menentukan kunci KMS kustom dan bukan Kunci milik AWS saat Anda membuat integrasi, kebijakan kunci harus menyediakan akses utama layanan Amazon Redshift ke `CreateGrant` tindakan tersebut. Selain itu, harus memungkinkan pengguna saat ini untuk melakukan `CreateGrant` tindakan `DescribeKey` dan tindakan.

Kebijakan contoh berikut menunjukkan cara memberikan izin yang diperlukan dalam kebijakan utama. Ini termasuk kunci konteks untuk lebih mengurangi ruang lingkup izin.

Contoh kebijakan kunci

JSON

```
{
    "Version": "2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift service principal to add a grant to a KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about
 a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

Untuk informasi selengkapnya, lihat Membuat kebijakan kunci di Panduan Developer AWS Key Management Service .

## Langkah selanjutnya

Setelah berhasil membuat integrasi nol-ETL, Anda harus membuat basis data tujuan dalam klaster atau grup kerja Amazon Redshift target Anda. Kemudian, Anda dapat mulai menambahkan data ke sumber cluster Aurora DB dan menanyakannya di Amazon Redshift. Untuk petunjuknya, lihat Membuat basis data tujuan di Amazon Redshift.

# Membuat integrasi Aurora Zero-ETL dengan rumah danau Amazon SageMaker

Saat Anda membuat integrasi Aurora Zero-ETL dengan Amazon SageMaker lakehouse, Anda menentukan sumber Kluster Aurora DB dan katalog terkelola target. AWS Glue Anda juga dapat menyesuaikan pengaturan enkripsi dan menambahkan tag. Aurora menciptakan integrasi antara cluster DB database dan targetnya. Setelah integrasi aktif, data apa pun yang Anda masukkan ke dalam cluster DB sumber akan direplikasi ke target yang dikonfigurasi.

## Prasyarat

Sebelum Anda membuat integrasi nol-ETL dengan Amazon SageMaker lakehouse, Anda harus membuat cluster DB sumber dan katalog terkelola target. AWS Glue Anda juga harus mengizinkan replikasi ke dalam katalog dengan menambahkan cluster DB sebagai sumber integrasi resmi.

Untuk petunjuk cara menyelesaikan setiap langkah ini, lihat the section called “Mulai menggunakan integrasi nol-ETL”.

## Izin yang diperlukan

Izin IAM tertentu diperlukan untuk membuat integrasi nol-ETL dengan lakehouse. Amazon SageMaker Setidaknya, Anda memerlukan izin untuk melakukan tindakan berikut:

- 
- Lihat dan hapus semua integrasi nol-ETL.
- Buat integrasi masuk ke dalam katalog AWS Glue terkelola target.
- Akses bucket Amazon S3 yang digunakan oleh katalog terkelola. AWS Glue
- Gunakan AWS KMS kunci untuk enkripsi jika enkripsi khusus dikonfigurasi.
- Daftarkan sumber daya dengan Lake Formation.
- Letakkan kebijakan sumber daya pada katalog AWS Glue terkelola untuk mengotorisasi integrasi masuk.

Kebijakan contoh berikut menunjukkan izin hak istimewa paling sedikit yang diperlukan untuk membuat dan mengelola integrasi dengan lakehouse. Amazon SageMaker Anda mungkin tidak memerlukan izin persis ini jika pengguna atau peran Anda memiliki izin yang lebih luas, seperti kebijakan terkelola `AdministratorAccess`.

Selain itu, Anda harus mengonfigurasi kebijakan sumber daya pada katalog AWS Glue terkelola target untuk mengotorisasi integrasi masuk. Gunakan AWS CLI perintah berikut untuk menerapkan kebijakan sumber daya.

Contoh AWS CLI perintah untuk mengotorisasi integrasi masuk pada katalog target

```
aws glue put-resource-policy \
      --policy-in-json  '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Action": [
            "glue:AuthorizeInboundIntegration"
        ],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"],
        "Condition": {
            "StringEquals": {
                "aws:SourceArn": "arn:aws:rds:region:account_id:db:source_name"
            }
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "account_id"
        },
        "Action": ["glue:CreateInboundIntegration"],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"]
    }
    ]
}' \
      --region region
```

> **Note**
>
> Glue catalog Amazon Resource Names (ARNs) memiliki format berikut:
>
> - Katalog Glue — `arn:aws:glue:{region}:{account-id}:catalog/catalog-name`

## Memilih katalog AWS Glue terkelola target di akun yang berbeda

Jika Anda berencana untuk menentukan katalog AWS Glue terkelola target yang ada di katalog lain Akun AWS, Anda harus membuat peran yang memungkinkan pengguna di akun saat ini mengakses sumber daya di akun target. Untuk informasi selengkapnya, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki.

Peran harus memiliki izin berikut, yang memungkinkan pengguna untuk melihat AWS Glue katalog yang tersedia di akun target.

Izin dan kebijakan kepercayaan yang diperlukan

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:GetCatalog"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Peran harus memiliki kebijakan kepercayaan berikut ini, yang menentukan ID akun target.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

Untuk petunjuk cara membuat peran, lihat Membuat peran menggunakan kebijakan kepercayaan kustom.

## Membuat integrasi nol-ETL dengan rumah danau Amazon SageMaker

Anda dapat membuat integrasi nol-ETL dengan Amazon SageMaker lakehouse menggunakan AWS Management Console,, atau RDS API. AWS CLI

> **Important**
>
> Integrasi nol-ETL dengan Amazon SageMaker lakehouse tidak mendukung operasi penyegaran atau sinkronisasi ulang. Jika Anda mengalami masalah dengan integrasi setelah pembuatan, Anda harus menghapus integrasi dan membuat yang baru.

Konsol RDS

Untuk membuat integrasi nol-ETL dengan rumah danau Amazon SageMaker

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Integrasi nol-ETL.
3. Pilih Buat integrasi nol-ETL.
4. Untuk Pengidentifikasi integrasi, masukkan nama untuk integrasi. Nama dapat memiliki maksimal 63 karakter alfanumerik dan dapat menyertakan tanda hubung.
5. Pilih Selanjutnya.
6. Untuk Sumber, pilih Aurora DB cluster tempat data akan berasal.

> **Note**
>
> RDS memberi tahu Anda jika parameter cluster DB tidak dikonfigurasi dengan benar. Jika Anda menerima pesan ini, Anda dapat memilih Perbaiki untuk saya, atau mengonfigurasinya secara manual. Untuk petunjuk cara memperbaikinya secara manual, lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”. Memodifikasi parameter klaster DB membutuhkan boot ulang. Sebelum Anda dapat membuat integrasi, reboot harus lengkap dan nilai parameter baru harus berhasil diterapkan ke cluster .

7. (Opsional) Pilih Sesuaikan opsi pemfilteran data dan tambahkan filter data ke integrasi Anda. Anda dapat menggunakan filter data untuk menentukan ruang lingkup replikasi ke Amazon SageMaker lakehouse target. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.
8. Setelah cluster DB sumber Anda berhasil dikonfigurasi, pilih Berikutnya.
9. Untuk Target, lakukan hal berikut:

    1. (Opsional) Untuk menggunakan target Amazon SageMaker lakehouse yang berbeda Akun AWS , pilih Tentukan akun yang berbeda. Kemudian, masukkan ARN peran IAM dengan izin untuk menampilkan katalog Anda. AWS Glue Untuk petunjuk cara membuat peran IAM, lihat the section called “Memilih katalog AWS Glue terkelola target di akun yang berbeda”.
    2. Untuk AWS Glue katalog, pilih target untuk data yang direplikasi dari cluster DB sumber. Anda dapat memilih katalog AWS Glue terkelola yang ada sebagai target.
    3. Peran IAM target perlu menjelaskan izin pada katalog target dan harus memiliki izin berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "glue:GetCatalog",
            "Resource": [
                "arn:aws:glue:region:account-id:catalog/*",
                "arn:aws:glue:region:account-id:catalog"
            ]
        }
    ]
}
```

    Peran IAM target harus memiliki hubungan kepercayaan berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4. Anda harus memberikan izin deskripsi peran IAM target untuk katalog AWS Glue terkelola target dengan peran administrator Lake Formation yang dibuat. Langkah 3b: Buat AWS Glue katalog untuk integrasi Amazon SageMaker nol-ETL

> **Note**
>
> RDS memberi tahu Anda jika kebijakan sumber daya atau pengaturan konfigurasi untuk katalog AWS Glue terkelola yang ditentukan tidak dikonfigurasi dengan benar. Jika Anda menerima pesan ini, Anda dapat memilih Perbaiki untuk saya, atau mengonfigurasinya secara manual.
> Jika sumber dan target yang Anda pilih berada di Akun AWS yang berbeda, maka Amazon RDS tidak dapat memperbaiki pengaturan ini untuk Anda. Anda harus menavigasi ke akun lain dan memperbaikinya secara manualSageMaker Unified Studio.

10. Setelah katalog AWS Glue terkelola target Anda dikonfigurasi dengan benar, pilih Berikutnya.
11. (Opsional) Untuk Tag, tambahkan satu atau beberapa tag ke integrasi. Untuk informasi selengkapnya, lihat the section called “Menandai RDSAurora dan sumber daya”.
12. Untuk Enkripsi, tentukan cara enkripsi integrasi Anda. Secara default, RDS mengenkripsi semua integrasi dengan file. Kunci milik AWS Untuk memilih kunci yang dikelola pelanggan, aktifkan Sesuaikan pengaturan enkripsi dan pilih kunci KMS yang akan digunakan untuk enkripsi. Untuk informasi selengkapnya, lihat the section called “Mengenkripsi sumber daya Amazon Aurora”.

    Secara opsional, tambahkan konteks enkripsi. Untuk informasi lebih lanjut, lihat Konteks enkripsi di Panduan Developer AWS Key Management Service .

> **Note**
>
> Amazon RDS menambahkan pasangan konteks enkripsi berikut selain yang Anda tambahkan:
>
> - `aws:glue:integration:arn` - `IntegrationArn`
> - `aws:servicename:id` - `glue`

Ini mengurangi jumlah keseluruhan pasangan yang dapat Anda tambahkan dari 8 menjadi 6, dan berkontribusi pada batas karakter keseluruhan dari batasan hibah. Untuk informasi selengkapnya, lihat Menggunakan batasan hibah di Panduan Pengembang.AWS Key Management Service

13. Pilih Selanjutnya.
14. Tinjau pengaturan integrasi Anda dan pilih Buat integrasi nol-ETL.

    Jika pembuatan gagal, lihat the section called “Memecahkan masalah integrasi nol-ETL” untuk langkah-langkah pemecahan masalah.

Integrasi memiliki status `Creating` saat sedang dibuat, dan target Amazon SageMaker lakehouse memiliki status. `Modifying` Selama waktu ini, Anda tidak dapat menanyakan katalog atau membuat perubahan konfigurasi apa pun di dalamnya.

Ketika integrasi berhasil dibuat, status integrasi dan target Amazon SageMaker lakehouse keduanya berubah menjadi. `Active`

AWS CLI

Untuk menyiapkan katalog AWS Glue terkelola target untuk integrasi nol-ETL menggunakan AWS CLI, Anda harus terlebih dahulu menggunakan create-integration-resource-propertyperintah dengan opsi berikut:

- `--resource-arn`— Tentukan ARN dari katalog AWS Glue terkelola yang akan menjadi target integrasi.
- `--target-processing-properties`— Tentukan ARN peran IAM untuk mengakses katalog terkelola target AWS Glue

```
aws glue create-integration-resource-property --region us-east-1
 --resource-arn arn:aws:glue:region:account_id:catalog/catalog_name \
 --target-processing-properties '{"RoleArn" : "arn:aws:iam::account_id:role/
TargetIamRole"}'
```

Untuk membuat integrasi nol-ETL dengan Amazon SageMaker lakehouse menggunakan AWS CLI, gunakan perintah create-integration dengan opsi berikut:

- `--integration-name` – Tentukan nama untuk integrasi.
- `--source-arn`— Tentukan ARN dari DB cluster yang akan menjadi sumber untuk integrasi.
- `--target-arn`— Tentukan ARN dari katalog AWS Glue terkelola yang akan menjadi target integrasi.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-integration \
    --integration-name my-sagemaker-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

Untuk Windows:

```
aws rds create-integration ^
    --integration-name my-sagemaker-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

API RDS

Untuk membuat integrasi nol-ETL dengan Amazon SageMaker menggunakan Amazon RDS API, gunakan `CreateIntegration`operasi dengan parameter berikut:

> **Note**
>
> Nama katalog dibatasi hingga 19 karakter. Pastikan IntegrationName parameter Anda memenuhi persyaratan ini jika akan digunakan sebagai nama katalog.

- `IntegrationName` – Tentukan nama untuk integrasi.
- `SourceArn`— Tentukan ARN dari DB cluster yang akan menjadi sumber untuk integrasi.
- `TargetArn`— Tentukan ARN dari katalog AWS Glue terkelola yang akan menjadi target integrasi.

# Mengenkripsi integrasi dengan kunci yang dikelola pelanggan

Jika Anda menentukan kunci KMS kustom dan bukan Kunci milik AWS saat Anda membuat integrasi denganAmazon SageMaker, kebijakan kunci harus menyediakan akses utama SageMaker Unified Studio layanan ke `CreateGrant` tindakan tersebut. Selain itu, harus memungkinkan pengguna saat ini untuk melakukan `CreateGrant` tindakan `DescribeKey` dan tindakan.

Kebijakan contoh berikut menunjukkan cara memberikan izin yang diperlukan dalam kebijakan utama. Ini termasuk kunci konteks untuk lebih mengurangi ruang lingkup izin.

Contoh kebijakan kunci

```
{
    "Version": "2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Glue service principal to add a grant to an AWS KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
```

```
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about a KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

Untuk informasi selengkapnya, lihat Membuat kebijakan kunci di Panduan Developer AWS Key Management Service .

## Langkah selanjutnya

Setelah Anda berhasil membuat integrasi nol-ETLAmazon SageMaker, Anda dapat mulai menambahkan data ke DB dan menanyakannya di lakehouse Anda. Amazon SageMaker Data akan direplikasi secara otomatis dan tersedia untuk beban kerja analitik dan pembelajaran mesin.

# Pemfilteran data untuk integrasi Zero-ETL

Integrasi Aurora Zero-ETL mendukung pemfilteran data, yang memungkinkan Anda mengontrol data mana yang direplikasi dari basis data sumber Anda Kluster Aurora DB ke gudang data target Anda. Alih-alih mereplikasi seluruh database, Anda dapat menerapkan satu atau beberapa filter untuk secara selektif menyertakan atau mengecualikan tabel tertentu. Ini membantu Anda mengoptimalkan kinerja penyimpanan dan kueri dengan memastikan bahwa hanya data yang relevan yang ditransfer. Saat ini, penyaringan terbatas pada tingkat database dan tabel. Pemfilteran tingkat kolom dan baris tidak didukung.

Pemfilteran data dapat berguna ketika Anda ingin:

- Bergabunglah dengan tabel tertentu dari dua atau lebih cluster sumber yang berbeda, dan Anda tidak memerlukan data lengkap dari salah satu cluster .
- Menghemat biaya dengan melakukan analitik hanya menggunakan subset tabel daripada seluruh armada database.
- Saring informasi sensitif—seperti nomor telepon, alamat, atau detail kartu kredit—dari tabel tertentu.

Anda dapat menambahkan filter data ke integrasi nol-ETL menggunakan, AWS Command Line Interface (AWS CLI) AWS Management Console, atau Amazon RDS API.

Jika integrasi memiliki cluster yang disediakan sebagai targetnya, cluster harus berada di patch 180 atau lebih tinggi untuk menggunakan pemfilteran data.

Topik

- Format filter data
- Filter logika
- Filter prioritas
- 
- Contoh PostgreSQL Aurora

- Menambahkan filter data ke integrasi
- Menghapus filter data dari integrasi

## Format filter data

Anda dapat menentukan beberapa filter untuk satu integrasi. Setiap filter menyertakan atau mengecualikan tabel database yang ada dan yang akan datang yang cocok dengan salah satu pola dalam ekspresi filter. Integrasi Aurora Zero-ETL menggunakan sintaks filter Maxwell untuk pemfilteran data.

Setiap filter memiliki elemen-elemen berikut:

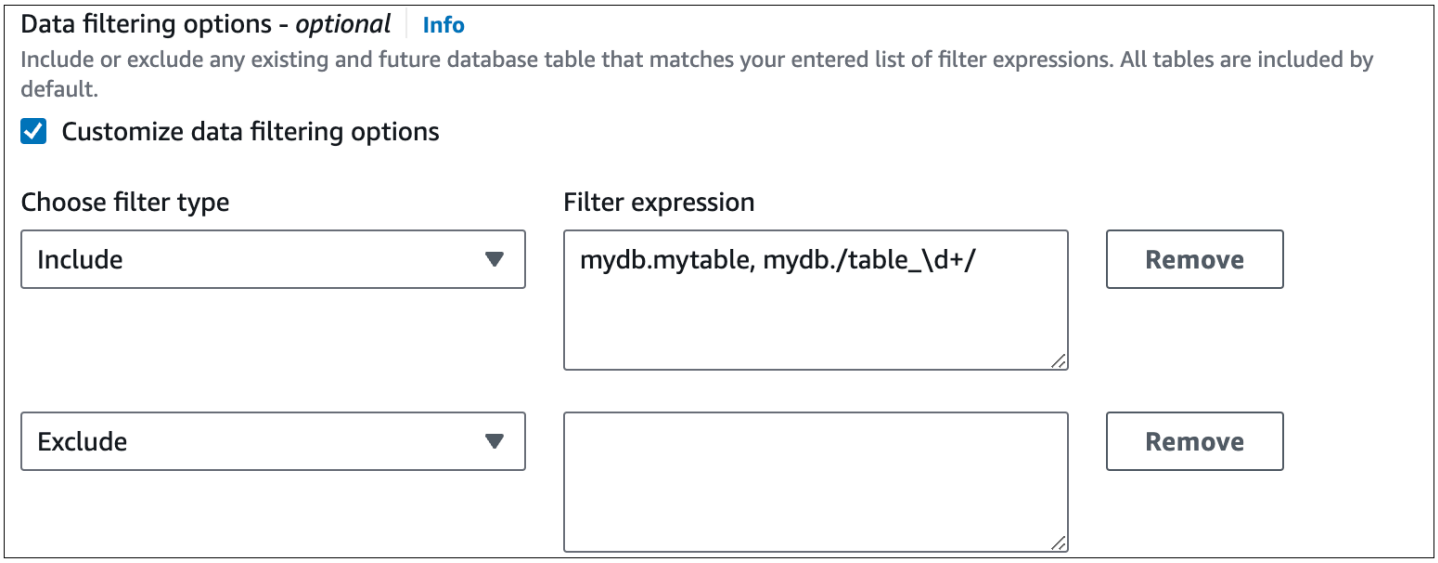
| Elemen | Deskripsi |
| --- | --- |
| Jenis filter | Jenis `Include` filter mencakup semua tabel yang cocok dengan salah satu pola dalam ekspresi filter. Jenis `Exclude` filter mengecualikan semua tabel yang cocok dengan salah satu pola. |
| Ekspresi filter | Daftar pola yang dipisahkan koma. Ekspresi harus menggunakan sintaks filter Maxwell. |
| Pola | Pola filter dalam format *`database`*`.`*`table`* untuk Aurora MySQL, atau untuk Aurora PostgreSQL. *`database`*`.`*`schema`*`.`*`table`* Anda dapat menentukan nama literal, atau menentukan ekspresi reguler.<br><br>**Note**<br>Untuk Aurora MySQL, ekspresi reguler didukung dalam database dan nama tabel. Untuk Aurora PostgreSQL, ekspresi reguler hanya didukung dalam skema dan nama tabel, bukan dalam nama database. |

| Elemen | Deskripsi |
| --- | --- |
| | Anda tidak dapat menyertakan filter tingkat kolom atau daftar denylist.<br><br>Integrasi tunggal dapat memiliki maksimum 99 pola total. Di konsol, Anda dapat memasukkan pola dalam satu ekspresi filter, atau menyebarkannya di antara beberapa ekspresi. Pola tunggal tidak dapat melebihi 256 karakter panjangnya. |

**Important**

Jika Anda memilih cluster DB sumber PostgreSQL Aurora, Anda harus menentukan setidaknya satu pola filter data. Minimal, pola harus menyertakan database tunggal (*`database-name`*`.*.*`) untuk replikasi ke gudang data target.

Gambar berikut menunjukkan struktur filter data MySQL Aurora di konsol:




**Important**

Jangan sertakan informasi identitas pribadi, rahasia, atau sensitif dalam pola filter Anda.

### Filter data di AWS CLI

Saat menggunakan AWS CLI untuk menambahkan filter data, sintaksnya sedikit berbeda dari konsol. Anda harus menetapkan jenis filter (`Include`atau`Exclude`) untuk setiap pola satu per satu, sehingga Anda tidak dapat mengelompokkan beberapa pola di bawah satu jenis filter.

Misalnya, di konsol Anda dapat mengelompokkan pola dipisahkan koma berikut di bawah satu `Include` pernyataan:

Aurora MySQL

```
mydb.mytable, mydb./table_\d+/
```

Aurora PostgreSQL

```
mydb.myschema.mytable, mydb.myschema./table_\d+/
```

Namun, saat menggunakan AWS CLI, filter data yang sama harus dalam format berikut:

Aurora MySQL

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

Aurora PostgreSQL

```
'include: mydb.myschema.mytable, include: mydb.myschema./table_\d+/'
```

## Filter logika

Jika Anda tidak menentukan filter data apa pun dalam integrasi Anda, Aurora mengasumsikan filter `include:*.*` default, yang mereplikasi semua tabel ke gudang data target. Namun, jika Anda menambahkan setidaknya satu filter, logika default beralih ke`exclude:*.*`, yang mengecualikan semua tabel secara default. Ini memungkinkan Anda secara eksplisit menentukan database dan tabel mana yang akan disertakan dalam replikasi.

Misalnya, jika Anda menentukan filter berikut:

```
'include: db.table1, include: db.table2'
```

Aurora mengevaluasi filter sebagai berikut:

```
'exclude:*.*, include: db.table1, include: db.table2'
```

Oleh karena itu, Aurora hanya `table1` mereplikasi `table2` dan dari database db bernama ke gudang data target.

## Filter prioritas

Aurora mengevaluasi filter data dalam urutan yang Anda tentukan. Dalam AWS Management Console, ia memproses ekspresi filter dari kiri ke kanan dan atas ke bawah. Filter kedua atau pola individual yang mengikuti yang pertama dapat menimpanya.

Misalnya, jika filter pertama adalah`Include books.stephenking`, itu hanya mencakup `stephenking` tabel dari `books` database. Namun, jika Anda menambahkan filter kedua`Exclude books.*`, itu akan menggantikan filter pertama. Ini mencegah tabel apa pun dari `books` indeks direplikasi ke gudang data target.

Ketika Anda menentukan setidaknya satu filter, logika dimulai dengan mengasumsikan secara `exclude:*.*` default, yang secara otomatis mengecualikan semua tabel dari replikasi. Sebagai praktik terbaik, tentukan filter dari yang paling luas hingga yang paling spesifik. Mulailah dengan satu atau beberapa `Include` pernyataan untuk menentukan data yang akan direplikasi, lalu tambahkan `Exclude` filter untuk menghapus tabel tertentu secara selektif.

Prinsip yang sama berlaku untuk filter yang Anda definisikan menggunakan AWS CLI. Aurora mengevaluasi pola filter ini dalam urutan yang Anda tentukan, sehingga sebuah pola dapat menggantikan pola yang Anda tentukan sebelumnya.

Contoh berikut menunjukkan cara kerja penyaringan data untuk integrasi Aurora MySQL Zero-ETL:

- Sertakan semua database dan semua tabel:

  ```
  'include: *.*'
  ```

- Sertakan semua tabel dalam `books` database:

  ```
  'include: books.*'
  ```

- Kecualikan tabel apa pun bernama`mystery`:

  ```
  'include: *.*, exclude: *.mystery'
  ```

- Sertakan dua tabel spesifik dalam `books` database:

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- Sertakan semua tabel dalam `books` database, kecuali yang berisi substring`mystery`:

```
'include: books.*, exclude: books./.*mystery.*/'
```

- Sertakan semua tabel dalam `books` database, kecuali yang dimulai dengan`mystery`:

```
'include: books.*, exclude: books./mystery.*/'
```

- Sertakan semua tabel dalam `books` database, kecuali yang diakhiri dengan`mystery`:

```
'include: books.*, exclude: books./.*mystery/'
```

- Sertakan semua tabel dalam `books` database yang dimulai dengan`table_`, kecuali yang bernama`table_stephen_king`. Misalnya, `table_movies` atau `table_books` akan direplikasi, tetapi tidak`table_stephen_king`.

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

## Contoh PostgreSQL Aurora

Contoh berikut menunjukkan cara kerja penyaringan data untuk integrasi Aurora PostgreSQL Zero-ETL:

- Sertakan semua tabel dalam `books` database:

```
'include: books.*.*'
```

- Kecualikan tabel apa pun yang disebutkan `mystery` dalam `books` database:

```
'include: books.*.*, exclude: books.*.mystery'
```

- Sertakan satu tabel dalam `books` database dalam `mystery` skema, dan satu tabel dalam `employee` database dalam `finance` skema:

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- Sertakan semua tabel dalam `books` database dan `science_fiction` skema, kecuali yang berisi substring`king`:

```
'include: books.science_fiction.*, exclude: books.*./.*king.*/
```

- Sertakan semua tabel dalam `books` database, kecuali yang memiliki nama skema yang dimulai dengan`sci`:

```
'include: books.*.*, exclude: books./sci.*/.*'
```

- Sertakan semua tabel dalam `books` database, kecuali yang ada di `mystery` skema yang diakhiri dengan`king`:

```
'include: books.*.*, exclude: books.mystery./.*king/'
```

- Sertakan semua tabel dalam `books` database yang dimulai dengan`table_`, kecuali yang bernama`table_stephen_king`. Misalnya, `table_movies` dalam `fiction` skema dan `table_books` skema direplikasi, tetapi tidak `table_stephen_king` di salah satu `mystery` skema:

```
'include: books.*./table_.*/, exclude: books.*.table_stephen_king'
```

## Menambahkan filter data ke integrasi

Anda dapat mengonfigurasi pemfilteran data menggunakan AWS Management Console, API AWS CLI, atau Amazon RDS.

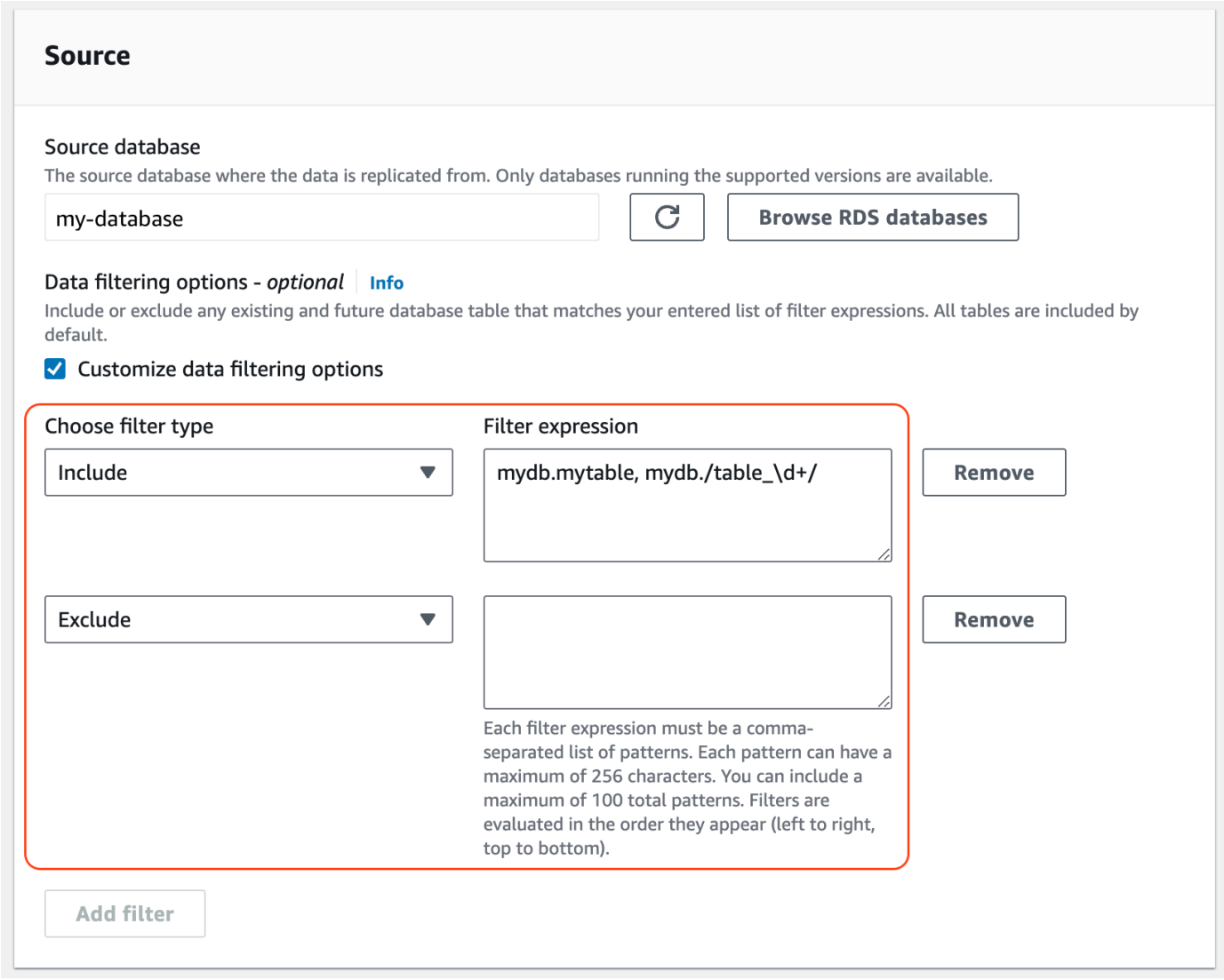
> **Important**
>
> Jika Anda menambahkan filter setelah membuat integrasi, Aurora memperlakukannya seolah-olah selalu ada. Ini menghapus data apa pun di gudang data target yang tidak cocok dengan kriteria pemfilteran baru dan menyinkronkan ulang semua tabel yang terpengaruh.

Konsol RDS

Untuk menambahkan filter data ke integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih integrasi nol-ETL. Pilih integrasi yang ingin Anda tambahkan filter data, lalu pilih Ubah.
3. Di bawah Sumber, tambahkan satu atau lebih `Include` dan `Exclude` pernyataan.

   Gambar berikut menunjukkan contoh filter data untuk integrasi MySQL:




4. Jika Anda puas dengan perubahan, pilih Lanjutkan dan Simpan perubahan.

AWS CLI

Untuk menambahkan filter data ke integrasi nol-ETL menggunakan AWS CLI, panggil perintah modify-integration. Selain pengidentifikasi integrasi, tentukan `--data-filter` parameter dengan daftar filter Maxwell yang dipisahkan koma. `Include` `Exclude`

Example

Contoh berikut menambahkan pola filter kemy`-integration`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-integration \
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

Untuk Windows:

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

API RDS

Untuk memodifikasi integrasi nol-ETL menggunakan RDS API, panggil operasi. ModifyIntegration Tentukan pengidentifikasi integrasi dan berikan daftar pola filter yang dipisahkan koma.

## Menghapus filter data dari integrasi

Saat Anda menghapus filter data dari integrasi, Aurora mengevaluasi kembali filter yang tersisa seolah-olah filter yang dihapus tidak pernah ada. Kemudian mereplikasi data yang sebelumnya dikecualikan yang sekarang memenuhi kriteria ke dalam gudang data target. Ini memicu sinkronisasi ulang semua tabel yang terpengaruh.

# Menambahkan data ke Aurora DB cluster dan menanyakannya

Untuk menyelesaikan pembuatan integrasi nol-ETL yang mereplikasi data dari Amazon Aurora ke Amazon Redshift, Anda harus membuat database di tujuan target.

Untuk koneksi dengan Amazon Redshift, sambungkan ke klaster atau grup kerja Amazon Redshift Anda dan buat database dengan referensi ke pengenal integrasi Anda. Kemudian, Anda dapat menambahkan data ke sumber Anda Aurora DB cluster dan melihatnya direplikasi di Amazon Redshift atau. Amazon SageMaker

Topik


- Membuat database target
- Menambahkan data ke sumber DB cluster
- Menanyakan data Anda di Amazon Redshift
- Perbedaan jenis data antara basis data Aurora dan Amazon Redshift
- Operasi DDL untuk Aurora PostgreSQL


## Membuat database target

Sebelum Anda dapat mulai mereplikasi data ke Amazon Redshift, setelah Anda membuat integrasi, Anda harus membuat database di gudang data target Anda. Database ini harus menyertakan referensi ke identifier integrasi. Anda dapat menggunakan konsol Amazon Redshift atau Editor kueri v2 untuk membuat basis data.

Untuk petunjuk cara membuat basis data tujuan, lihat Membuat basis data tujuan di Amazon Redshift.

## Menambahkan data ke sumber DB cluster

Setelah Anda mengkonfigurasi integrasi Anda, Anda dapat menambahkan beberapa data ke Aurora DB cluster yang ingin Anda replikasi ke gudang data Anda.

> **Note**
>
> Ada perbedaan antara tipe data di Aurora dan gudang analitik target. Untuk tabel pemetaan jenis data, lihat the section called “Perbedaan jenis data”.

Pertama, sambungkan ke cluster DB sumber menggunakan klien MySQL atau PostgreSQL pilihan Anda. Untuk petunjuk, lihat the section called “Menghubungkan ke klaster DB”.

Kemudian, buat tabel dan masukkan urutan data sampel.

> **Important**
>
> Pastikan tabel memiliki kunci primer. Jika tidak, tabel tidak dapat direplikasi ke gudang data target.

Utilitas pg_dump dan pg_restore PostgreSQL awalnya membuat tabel tanpa kunci utama dan kemudian menambahkannya setelahnya. Jika Anda menggunakan salah satu utilitas ini, sebaiknya buat skema terlebih dahulu, lalu muat data dalam perintah terpisah.

MySQL

Contoh berikut menggunakan utilitas MySQL Workbench.

```
CREATE DATABASE my_db;

USE my_db;

CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

Contoh berikut menggunakan terminal interaktif PostgreSQL `psql`. Saat menghubungkan ke klaster, sertakan basis data bernama yang Anda tentukan saat membuat integrasi.

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977,
 'Supernatural fiction');
```

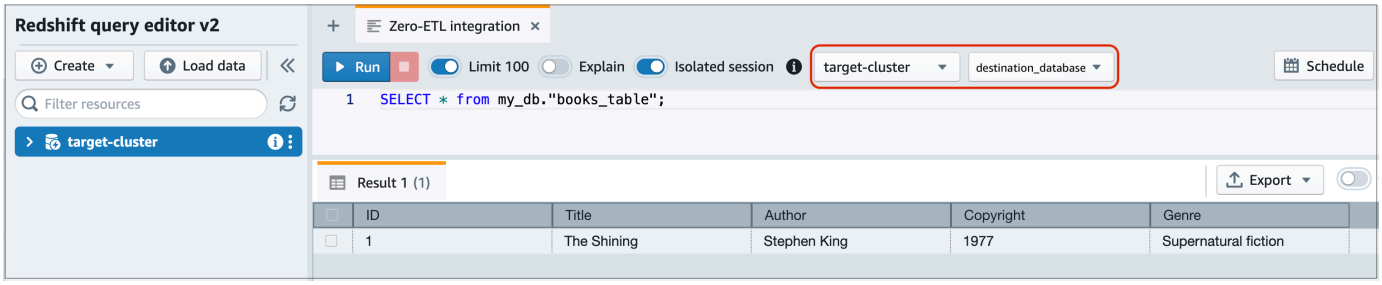
# Menanyakan data Anda di Amazon Redshift

Setelah Anda menambahkan data ke Aurora DB cluster, itu direplikasi ke database tujuan dan siap untuk ditanyakan.

Untuk mengueri data yang direplikasi

1. Buka konsol Amazon Redshift dan pilih Editor kueri v2 dari panel navigasi kiri.
2. Hubungkan ke klaster atau grup kerja Anda dan pilih basis data tujuan Anda (yang Anda buat dari integrasi) dari menu dropdown (destination_database dalam contoh ini). Untuk petunjuk cara membuat basis data tujuan, lihat Membuat basis data tujuan di Amazon Redshift.
3. Gunakan pernyataan SELECT untuk menanyakan data Anda. Dalam contoh ini, Anda dapat menjalankan perintah berikut untuk memilih semua data dari tabel yang Anda buat di sumber Aurora DB cluster:

```
SELECT * from my_db."books_table";
```




- *my_db* adalah nama skema basis data Aurora. Opsi ini hanya diperlukan untuk basis data MySQL.
- *books_table* adalah nama tabel Aurora.

Anda juga dapat menanyakan data menggunakan klien baris perintah. Misalnya:

```
destination_database=# select * from my_db."books_table";

 ID |       Title |        Author |   Copyright |                 Genre |  txn_seq |
 txn_id
----+-------------+---------------+-------------+-----------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> Untuk kepekaan huruf besar/kecil, gunakan tanda kutip ganda (" ") untuk nama skema, tabel, dan kolom. Untuk informasi selengkapnya, lihat enable_case_sensitive_identifier.

## Perbedaan jenis data antara basis data Aurora dan Amazon Redshift

atau Aurora PostgreSQL ke tipe data tujuan yang sesuai.  Amazon Aurora saat ini hanya mendukung jenis data ini untuk integrasi nol-ETL.

Jika tabel di cluster DB sumber Anda menyertakan tipe data yang tidak didukung, tabel tidak sinkron dan tidak dapat dikonsumsi oleh target tujuan. Streaming dari sumber ke target berlanjut, tetapi tabel dengan jenis data yang tidak didukung tidak tersedia. Untuk memperbaiki tabel dan membuatnya tersedia di tujuan target, Anda harus secara manual mengembalikan perubahan yang melanggar dan kemudian menyegarkan integrasi dengan menjalankan`ALTER DATABASE...INTEGRATION REFRESH`.

> **Note**
>
> Anda tidak dapat menyegarkan integrasi nol-ETL dengan lakehouse. Amazon SageMaker Sebagai gantinya, hapus dan coba buat integrasi lagi.

Topik

- Aurora MySQL
- Aurora PostgreSQL

### Aurora MySQL

| Jenis data Aurora MySQL | Tipe data target | Deskripsi | Batasan |
|---|---|---|---|
| INT | INTEGER | Bilangan bulat empat byte bertanda | Tidak ada |

| Jenis data Aurora MySQL | Tipe data target | Deskripsi | Batasan |
|---|---|---|---|
| SMALLINT | SMALLINT | Bilangan bulat dua byte bertanda | Tidak ada |
| TINYINT | SMALLINT | Bilangan bulat dua byte bertanda | Tidak ada |
| MEDIUMINT | INTEGER | Bilangan bulat empat byte bertanda | Tidak ada |
| BIGINT | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| INT UNSIGNED | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| TINYINT UNSIGNED | SMALLINT | Bilangan bulat dua byte bertanda | Tidak ada |
| MEDIUMINT UNSIGNED | INTEGER | Bilangan bulat empat byte bertanda | Tidak ada |
| BIGINT UNSIGNED | DECIMAL(20,0) | Numerik persis dari presisi yang dapat dipilih | Tidak ada |

| Jenis data Aurora MySQL | Tipe data target | Deskripsi | Batasan |
|---|---|---|---|
| DECIMAL(p,s) = NUMERIC(p,s) | DECIMAL(p,s) | Numerik persis dari presisi yang dapat dipilih | Presisi lebih besar dari 38 dan penskalaan lebih besar dari 37 tidak didukung |
| DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED | DECIMAL(p,s) | Numerik persis dari presisi yang dapat dipilih | Presisi lebih besar dari 38 dan penskalaan lebih besar dari 37 tidak didukung |
| FLOAT4/NYATA | REAL | Angka floating-point presisi tunggal | Tidak ada |
| FLOAT4/NYATA TIDAK DITANDATANGANI | REAL | Angka floating-point presisi tunggal | Tidak ada |
| DOUBLE/REAL/FLOAT8 | DOUBLE PRECISION | Angka floating-point presisi ganda | Tidak ada |
| DOUBLE/REAL/FLOAT8 TIDAK DITANDATANGANI | DOUBLE PRECISION | Angka floating-point presisi ganda | Tidak ada |
| BIT(n) | VARBYTE(8) | Nilai biner dengan panjang variabel | Tidak ada |

| Jenis data Aurora MySQL | Tipe data target | Deskripsi | Batasan |
|---|---|---|---|
| BINER (n) | VARBYTE(n) | Nilai biner dengan panjang variabel | Tidak ada |
| VARBINER (n) | VARBYTE(n) | Nilai biner dengan panjang variabel | Tidak ada |
| CHAR(n) | VARCHAR(n) | Nilai string dengan panjang variabel | Tidak ada |
| VARCHAR(n) | VARCHAR(n) | Nilai string dengan panjang variabel | Tidak ada |
| TEXT | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| TINYTEXT | VARCHAR(255) | Nilai string panjang variabel hingga 255 karakter | Tidak ada |
| MEDIUMTEXT | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| LONGTEXT | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |

| Jenis data Aurora MySQL | Tipe data target | Deskripsi | Batasan |
|---|---|---|---|
| ENUM | VARCHAR(1020) | Nilai string panjang variabel hingga 1.020 karakter | Tidak ada |
| SET | VARCHAR(1020) | Nilai string panjang variabel hingga 1.020 karakter | Tidak ada |
| DATE | DATE | Tanggal kalender (tahun, bulan, hari) | Tidak ada |
| DATETIME | TIMESTAMP | Tanggal dan waktu (tanpa zona waktu) | Tidak ada |
| TIMESTAMP(p) | TIMESTAMP | Tanggal dan waktu (tanpa zona waktu) | Tidak ada |
| TIME | VARCHAR(18) | Nilai string panjang variabel hingga 18 karakter | Tidak ada |
| YEAR | VARCHAR(4) | Nilai string panjang variabel hingga 4 karakter | Tidak ada |
| JSON | SUPER | Data atau dokumen semi-terstruktur sebagai nilai | Tidak ada |

## Aurora PostgreSQL

Integrasi nol-ETL untuk Aurora PostgreSQL tidak mendukung jenis data kustom atau jenis data yang dibuat oleh ekstensi.

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| array | SUPER | Data atau dokumen semi-terstruktur sebagai nilai | Tidak ada |
| bigint | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| bigserial | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| bit varying(n) | VARBYTE(n) | Nilai biner panjang variabel hingga 16.777.216 byte | Tidak ada |
| bit(n) | VARBYTE(n) | Nilai biner panjang variabel hingga 16.777.216 byte | Tidak ada |
| sedikit, sedikit bervariasi | VARBYTE (16777216) | Nilai biner panjang variabel hingga 16.777.216 byte | Tidak ada |
| boolean | BOOLEAN | Boolean logis (true/false) | Tidak ada |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| bytea | VARBYTE (16777216) | Nilai biner panjang variabel hingga 16.777.216 byte | Tidak ada |
| arang (n) | CHAR(n) | Nilai string karakter dengan panjang tetap hingga 65.535 byte | Tidak ada |
| char bervariasi (n) | VARCHAR(65535) | Nilai string karakter panjang variabel hingga 65.535 karakter | Tidak ada |
| cid | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| cidr | VARCHAR(19) | Nilai string panjang variabel hingga 19 karakter | Tidak ada |
| date | DATE | Tanggal kalender (tahun, bulan, hari) | Nilai lebih besar dari 294.276 A.D. tidak didukung |
| double precision | DOUBLE PRECISION | Angka floating-point presisi ganda | Nilai subnormal tidak sepenuhnya didukung |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| gtsvektor | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| inet | VARCHAR(19) | Nilai string panjang variabel hingga 19 karakter | Tidak ada |
| integer | INTEGER | Bilangan bulat empat byte bertanda | Tidak ada |
| int2vektor | SUPER | Data atau dokumen semi-terstruktur sebagai nilai. | Tidak ada |
| interval | INTERVAL | Durasi waktu | Hanya jenis INTERVAL yang menentukan tahun ke bulan atau hari ke kualifikasi kedua yang didukung. |
| json | SUPER | Data atau dokumen semi-terstruktur sebagai nilai | Tidak ada |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| jsonb | SUPER | Data atau dokumen semi-terstruktur sebagai nilai | Tidak ada |
| jsonpath | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| macaddr | VARCHAR (17) | Nilai string panjang variabel hingga 17 karakter | Tidak ada |
| macaddr8 | VARCHAR (23) | Nilai string panjang variabel hingga 23 karakter | Tidak ada |
| money | DECIMAL(20,3) | Jumlah mata uang | Tidak ada |
| name | VARCHAR (64) | Nilai string panjang variabel hingga 64 karakter | Tidak ada |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
| --- | --- | --- | --- |
| numeric(p,s) | DECIMAL(p,s) | Nilai presisi tetap yang ditentukan pengguna | • Nilai NaN tidak didukung<br>• Presisi dan skala harus didefinis ikan secara eksplisit dan tidak lebih besar dari 38 (presisi) dan 37 (skala)<br>• Skala negatif tidak didukung |
| oid | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| oidvektor | SUPER | Data atau dokumen semi-terstruktur sebagai nilai. | Tidak ada |
| pg_brin_bloom_summary | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| pg_dependencies | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| pg_lsn | VARCHAR (17) | Nilai string panjang variabel hingga 17 karakter | Tidak ada |
| pg_mcv_list | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| pg_ndistinct | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| pg_node_pohon | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| pg_snapshot | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| real | REAL | Angka floating-point presisi tunggal | Nilai subnormal tidak sepenuhnya didukung |
| refkursor | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| smallint | SMALLINT | Bilangan bulat dua byte bertanda | Tidak ada |
| smallserial | SMALLINT | Bilangan bulat dua byte bertanda | Tidak ada |
| serial | INTEGER | Bilangan bulat empat byte bertanda | Tidak ada |
| text | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| masa | VARCHAR (23) | Nilai string panjang variabel hingga 23 karakter | Tidak ada |
| waktu [(p)] tanpa zona waktu | VARCHAR(19) | Nilai string panjang variabel hingga 19 karakter | Nilai `Infinity` dan `-Infinity` tidak didukung |
| time [(p)] with time zone | VARCHAR(22) | Nilai string panjang variabel hingga 22 karakter | Nilai `Infinity` dan `-Infinity` tidak didukung |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
| --- | --- | --- | --- |
| stempel waktu [(p)] tanpa zona waktu | TIMESTAMP | Tanggal dan waktu (tanpa zona waktu) | • Nilai `Infinity` dan `-Infinity` tidak didukung<br>• Nilai yang lebih besar dari `9999-12-31` tidak didukung<br>• Nilai B.C. tidak didukung |
| timestamp [(p)] with time zone | TIMESTAMPTZ | Tanggal dan waktu (dengan zona waktu) | • Nilai `Infinity` dan `-Infinity` tidak didukung<br>• Nilai yang lebih besar dari `9999-12-31` tidak didukung<br>• Nilai B.C. tidak didukung |
| tsquery | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |

| Jenis data Aurora PostgreSQL | Jenis data Amazon Redshift | Deskripsi | Batasan |
|---|---|---|---|
| tsvektor | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| txid_snapshot | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |
| uuid | VARCHAR (36) | Variabel-panjang 36 string karakter | Tidak ada |
| xid | BIGINT | Bilangan bulat delapan byte bertanda | Tidak ada |
| xid8 | DESIMAL (20, 0) | Desimal presisi tetap | Tidak ada |
| xml | VARCHAR(65535) | Nilai string panjang variabel hingga 65.535 karakter | Tidak ada |

## Operasi DDL untuk Aurora PostgreSQL

Amazon Redshift berasal dari PostgreSQL, sehingga berbagi beberapa fitur dengan Aurora PostgreSQL karena arsitektur PostgreSQL yang umum. Integrasi nol-ETL memanfaatkan kesamaan ini untuk merampingkan replikasi data dari Aurora PostgreSQL ke Amazon Redshift, memetakan database berdasarkan nama dan memanfaatkan database, skema, dan struktur tabel bersama.

Pertimbangkan poin-poin berikut saat mengelola integrasi Aurora PostgreSQL Zero-ETL:

- Isolasi dikelola di tingkat database.
- Replikasi terjadi pada tingkat database.
- Database Aurora PostgreSQL dipetakan ke database Amazon Redshift berdasarkan nama, dengan data mengalir ke database Redshift berganti nama yang sesuai jika aslinya diganti namanya.

Terlepas dari kesamaan mereka, Amazon Redshift dan Aurora PostgreSQL memiliki perbedaan penting. Bagian berikut menguraikan respons sistem Amazon Redshift untuk operasi DDL umum.

Topik


- Operasi basis data
- Operasi skema
- Operasi tabel


## Operasi basis data

Tabel berikut menunjukkan respon sistem untuk operasi DDL database.

| Operasi DDL | Respon sistem Redshift |
| --- | --- |
| `CREATE DATABASE` | Tidak ada operasi |
| `DROP DATABASE` | Amazon Redshift menghapus semua data dalam database Redshift target. |
| `RENAME DATABASE` | Amazon Redshift menghapus semua data dalam database target asli dan menyinkronkan ulang data dalam database target baru. Jika database baru tidak ada, Anda harus membuatnya secara manual. Untuk petunjukn ya, lihat Membuat database tujuan di Amazon Redshift. |

## Operasi skema

Tabel berikut menunjukkan respon sistem untuk operasi skema DDL.

| Operasi DDL | Respon sistem Redshift |
| --- | --- |
| `CREATE SCHEMA` | Tidak ada operasi |
| `DROP SCHEMA` | Amazon Redshift menjatuhkan skema asli. |
| `RENAME SCHEMA` | Amazon Redshift menghapus skema asli lalu menyinkronkan ulang data dalam skema baru. |

## Operasi tabel

Tabel berikut menunjukkan respon sistem untuk operasi tabel DDL.

| Operasi DDL | Respon sistem Redshift |
| --- | --- |
| `CREATE TABLE` | Amazon Redshift membuat tabel.<br><br>Beberapa operasi menyebabkan pembuatan tabel gagal, seperti membuat tabel tanpa kunci utama atau melakukan partisi deklaratif. Untuk informasi selengkapnya, lihat the section called “Keterbatasan Aurora PostgreSQL” dan the section called “Memecahkan masalah integrasi nol-ETL”. |
| `DROP TABLE` | Amazon Redshift menjatuhkan meja. |
| `TRUNCATE TABLE` | Amazon Redshift memotong tabel. |
| `ALTER TABLE (RENAME... )` | Amazon Redshift mengganti nama tabel atau kolom. |
| `ALTER TABLE (SET SCHEMA)` | Amazon Redshift menjatuhkan tabel dalam skema asli dan menyinkronkan ulang tabel dalam skema baru. |
| `ALTER TABLE (ADD PRIMARY KEY)` | Amazon Redshift menambahkan kunci utama dan menyinkronkan ulang tabel. |

| Operasi DDL | Respon sistem Redshift |
| --- | --- |
| `ALTER TABLE (ADD COLUMN)` | Amazon Redshift menambahkan kolom ke tabel. |
| `ALTER TABLE (DROP COLUMN)` | Amazon Redshift menjatuhkan kolom jika bukan kolom kunci utama. Jika tidak, itu menyinkronkan ulang tabel. |
| `ALTER TABLE (SET LOGGED/UNLOGGED )` | Jika Anda mengubah tabel menjadi login, Amazon Redshift akan menyinkronkan ulang tabel. Jika Anda mengubah tabel menjadi tidak tercatat, Amazon Redshift akan menjatuhkan tabel. |

# Melihat dan memantau integrasi

Anda dapat melihat detail integrasi nol-ETL Amazon Aurora untuk melihat informasi konfigurasi dan statusnya saat ini. Anda juga dapat memantau status integrasi dengan mengueri tampilan sistem tertentu di Amazon Redshift. Selain itu, Amazon Redshift menerbitkan metrik terkait integrasi tertentu ke Amazon, yang dapat Anda lihat dalam konsol CloudWatch Amazon Redshift.

Topik

- Melihat integrasi
- Integrasi pemantauan menggunakan tabel sistem untuk Amazon Redshift
- Memantau integrasi dengan Amazon EventBridge untuk Amazon Redshift

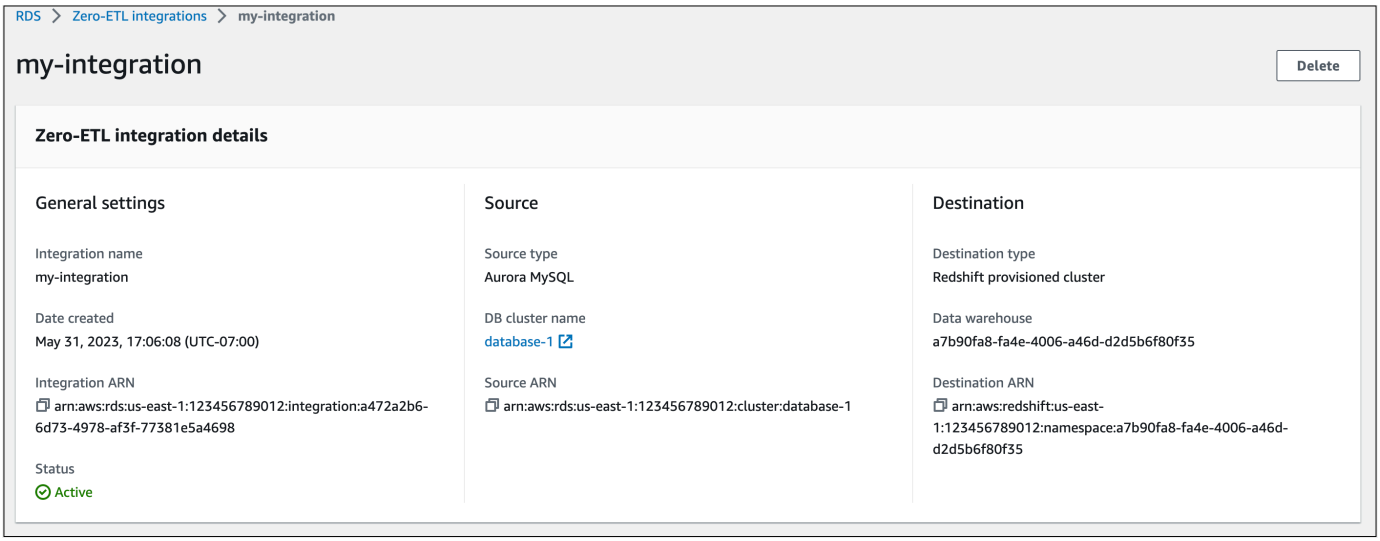
## Melihat integrasi

Anda dapat melihat integrasi Aurora Zero-ETL menggunakan, API, atau RDS AWS Management Console. AWS CLI

Konsol

Untuk melihat detail integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi sebelah kiri, pilih Integrasi nol-ETL.
3. Pilih integrasi untuk melihat detail lebih lanjut tentangnya, seperti cluster DB data sumber dan gudang data target.




Integrasi dapat memiliki status berikut:

- `Creating` – Integrasi sedang dibuat.
- `Active` – Integrasi sedang mengirimkan data transaksional ke gudang data target.
- `Syncing` – Integrasi telah mengalami kesalahan yang dapat dipulihkan dan sedang melakukan reseeding data. Tabel yang terpengaruh tidak tersedia untuk kueri sampai selesai disinkronkan ulang.
- `Needs attention` – Integrasi mengalami peristiwa atau kesalahan yang memerlukan intervensi manual untuk menyelesaikannya. Untuk memperbaiki masalah, ikuti petunjuk dalam pesan kesalahan di halaman detail integrasi.
- `Failed` – Integrasi mengalami peristiwa atau kesalahan yang tidak dapat dipulihkan dan tidak dapat diperbaiki. Anda harus menghapus dan membuat ulang integrasi.
- `Deleting` – Integrasi sedang dihapus.

AWS CLI

Untuk melihat semua integrasi nol-ETL di akun saat ini menggunakan AWS CLI, gunakan perintah deskripsi-integrasi dan tentukan opsi. `--integration-identifier`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Untuk Windows:

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

API RDS

Untuk melihat integrasi nol-ETL menggunakan API Amazon RDS, gunakan operasi `DescribeIntegrations` dengan parameter `IntegrationIdentifier`.

## Integrasi pemantauan menggunakan tabel sistem untuk Amazon Redshift

Amazon Redshift memiliki tabel dan tampilan sistem yang berisi informasi tentang bagaimana sistem berfungsi. Anda dapat mengueri tabel dan tampilan sistem ini dengan cara yang sama seperti Anda akan mengueri tabel basis data lainnya. Untuk informasi selengkapnya tentang tabel dan tampilan sistem di Amazon Redshift, lihat Referensi tabel dan tampilan sistem di Panduan Pengembang Database Amazon Redshift.

Anda dapat menanyakan tampilan dan tabel sistem berikut untuk mendapatkan informasi tentang integrasi Aurora Zero-ETL Anda:

- SVV_INTEGRATION – Menyediakan detail konfigurasi untuk integrasi Anda.
- SVV_INTEGRATION_TABLE_STATE – Menjelaskan status setiap tabel dalam integrasi.
- SYS_INTEGRATION_TABLE_STATE_CHANGE – Menampilkan log perubahan status tabel untuk integrasi.
- SYS_INTEGRATION_ACTIVITY – Menyediakan informasi tentang proses integrasi yang selesai.

Semua CloudWatch metrik Amazon terkait integrasi berasal dari Amazon Redshift. Untuk informasi selengkapnya, lihat Metrik untuk integrasi Nol-ETL di Panduan Manajemen Pergeseran Merah Amazon. Saat ini, Amazon Aurora tidak mempublikasikan metrik integrasi apa pun. CloudWatch

### Memantau integrasi dengan Amazon EventBridge untuk Amazon Redshift

Amazon Redshift mengirim peristiwa terkait integrasi ke Amazon. EventBridge Untuk daftar acara dan acara terkait IDs, lihat Pemberitahuan peristiwa integrasi nol-ETL dengan Amazon di Panduan Manajemen Pergeseran Merah EventBridge Amazon.

## Memodifikasi integrasi Nol-ETL

Anda hanya dapat memodifikasi nama, deskripsi, dan opsi pemfilteran data untuk integrasi nol-ETL di gudang data yang didukung. Anda tidak dapat memodifikasi AWS KMS kunci yang digunakan untuk mengenkripsi integrasi, atau basis data sumber atau target.

Jika Anda menambahkan filter data ke integrasi yang ada, Aurora mengevaluasi kembali filter seolah-olah selalu ada. Ini menghapus data apa pun yang saat ini ada di gudang data target yang tidak sesuai dengan kriteria penyaringan baru. Jika Anda menghapus filter data dari integrasi, itu mereplikasi data apa pun yang sebelumnya tidak cocok dengan kriteria pemfilteran (tetapi sekarang) ke gudang data target. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.

Anda dapat memodifikasi integrasi nol-ETL menggunakan AWS Management Console, API, AWS CLI atau Amazon RDS.

### Konsol RDS

Untuk memodifikasi integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih integrasi nol-ETL, lalu pilih integrasi yang ingin Anda ubah.
3. Pilih Ubah dan buat modifikasi pada pengaturan apa pun yang tersedia.
4. Ketika semua perubahan seperti yang Anda inginkan, pilih Ubah.

### AWS CLI

Untuk memodifikasi integrasi nol-ETL menggunakan AWS CLI, panggil perintah modify-integration. Seiring dengan`--integration-identifier`, tentukan salah satu opsi berikut:

- `--integration-name`— Tentukan nama baru untuk integrasi.
- `--description`— Tentukan deskripsi baru untuk integrasi.
- `--data-filter`— Tentukan opsi pemfilteran data untuk integrasi. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.

Example

Permintaan berikut memodifikasi integrasi yang ada.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

Untuk Windows:

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

### API RDS

Untuk memodifikasi integrasi nol-ETL menggunakan RDS API, panggil operasi. ModifyIntegration Tentukan pengidentifikasi integrasi, dan parameter yang ingin Anda modifikasi.

## Menghapus integrasi Nol-ETL

Data transaksional Anda tidak dihapus dari Amazon atau tujuan analitik, tetapi Amazon tidak mengirim data baru ke Amazon Redshift atau. Amazon SageMaker

Anda hanya dapat menghapus integrasi ketika integrasi ini memiliki status `Active`, `Failed`, `Syncing`, atau `Needs attention`.

Anda dapat menghapus integrasi nol-ETL menggunakan AWS Management Console,, atau RDS API AWS CLI.

### Konsol

Untuk menghapus integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi sebelah kiri, pilih Integrasi nol-ETL.
3. Pilih integrasi nol-ETL yang ingin Anda hapus.
4. Pilih Tindakan, Hapus, dan konfirmasi penghapusan.

### AWS CLI

Untuk menghapus integrasi nol-ETL, gunakan perintah delete-integration dan tentukan opsi `--integration-identifier`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Untuk Windows:

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

### API RDS

Untuk menghapus integrasi nol-ETL menggunakan API Amazon RDS, gunakan operasi `DeleteIntegration` dengan parameter `IntegrationIdentifier`.

## Memecahkan masalah integrasi

Anda dapat memeriksa status integrasi nol-ETL dengan menanyakan tabel sistem SVV_INTEGRATION di tujuan analitik. Jika kolom `state` memiliki nilai `ErrorState`, artinya

ada sesuatu yang salah. Untuk informasi selengkapnya, lihat the section called “Pemantauan menggunakan tabel sistem”.

Gunakan informasi berikut untuk memecahkan masalah umum dengan integrasi Aurora Zero-ETL.

> **Important**
>
> Operasi resync dan refresh tidak tersedia untuk integrasi nol-ETL dengan lakehouse. Amazon SageMaker Jika ada masalah dengan integrasi, Anda harus menghapus integrasi dan membuat integrasi baru. Anda tidak dapat menyegarkan atau menyinkronkan ulang integrasi yang ada.

Topik


- Saya tidak dapat membuat integrasi nol-ETL
- Integrasi saya terjebak dalam keadaan Syncing
- Tabel saya tidak bereplikasi ke Amazon Redshift
- Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang
- Masalah gagal integrasi untuk integrasi Amazon SageMaker lakehouse zero-ETL
- Perubahan DDL ada di Amazon Redshift sebelum transaksi DDL selesai untuk Aurora PostgreSQL


## Saya tidak dapat membuat integrasi nol-ETL

Jika Anda tidak dapat membuat integrasi nol-ETL, pastikan bahwa berikut ini benar untuk database sumber Anda:

- Database sumber Anda harus menjalankan versi mesin DB yang didukung. Untuk daftar versi yang didukung, lihat the section called “Integrasi nol-ETL ”.
- Anda mengkonfigurasi parameter DB dengan benar. Jika parameter yang diperlukan disetel secara tidak benar atau tidak terkait dengan database, pembuatan gagal. Lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”.

Selain itu, pastikan hal-hal berikut ini sudah benar untuk gudang data target Anda:

- Kepekaan huruf besar/kecil diaktifkan. Lihat Mengaktifkan kepekaan huruf besar/kecil untuk gudang data Anda.

- Anda menambahkan prinsipal yang diotorisasi dan sumber integrasi yang benar. Lihat Mengonfigurasi otorisasi untuk gudang data Amazon Redshift Anda.
- Gudang data dienkripsi (jika itu adalah cluster yang disediakan). Lihat enkripsi basis data Amazon Redshift.

## Integrasi saya terjebak dalam keadaan **`Syncing`**

Integrasi Anda mungkin secara konsisten menunjukkan status `Syncing` jika Anda mengubah nilai salah satu parameter DB yang diperlukan.

Untuk memperbaiki masalah ini, periksa nilai parameter dalam grup parameter yang terkait dengan cluster DB sumber, dan pastikan bahwa mereka cocok dengan nilai yang diperlukan. Untuk informasi selengkapnya, lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”.

Jika Anda memodifikasi parameter apa pun, pastikan untuk me-reboot cluster DB untuk menerapkan perubahan.

## Tabel saya tidak bereplikasi ke Amazon Redshift

Jika Anda tidak melihat satu atau beberapa tabel yang tercermin di Amazon Redshift, Anda dapat menjalankan perintah berikut untuk menyinkronkan ulang tabel tersebut:

```
ALTER DATABASE dbname INTEGRATION REFRESH TABLES table1, table2;
```

Untuk informasi selengkapnya, lihat ALTER DATABASE di referensi Amazon Redshift SQL.

Data Anda mungkin tidak mereplikasi karena satu atau beberapa tabel sumber Anda tidak memiliki kunci utama. Dasbor pemantauan di Amazon Redshift menampilkan status tabel ini sebagai`Failed`, dan status integrasi nol-ETL keseluruhan berubah menjadi. `Needs attention` Untuk mengatasi masalah ini, Anda dapat mengidentifikasi kunci yang ada di tabel Anda yang dapat menjadi kunci utama, atau Anda dapat menambahkan kunci primer sintetis. Untuk solusi mendetail, lihat dengan Amazon Redshift. sumber daya berikut:

- Menangani tabel tanpa kunci utama saat membuat Amazon Aurora MySQL atau Amazon RDS untuk integrasi MySQL Zero-ETL dengan Amazon Redshift
- Menangani tabel tanpa kunci utama saat membuat integrasi Amazon Aurora PostgreSQL Zero-ETL dengan Amazon Redshift

# Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang

Menjalankan perintah tertentu pada database sumber Anda mungkin memerlukan tabel Anda untuk disinkronkan ulang. Dalam kasus ini, tampilan sistem SVV_INTEGRATION_TABLE_STATE menunjukkan `table_state` dari `ResyncRequired`. Artinya, integrasi harus memuat ulang data sepenuhnya untuk tabel spesifik tersebut dari MySQL ke Amazon Redshift.

Ketika tabel mulai disinkronkan ulang, tabel ini akan beralih ke status `Syncing`. Anda tidak perlu mengambil tindakan manual apa pun untuk menyinkronkan ulang tabel. Saat data tabel disinkronkan ulang, Anda tidak dapat mengaksesnya di Amazon Redshift.

Berikut ini adalah beberapa contoh operasi yang dapat menempatkan tabel ke dalam status `ResyncRequired`, dan beberapa kemungkinan alternatif untuk dipertimbangkan.

| Operasi | Contoh | Alternatif |
| --- | --- | --- |
| Menambahkan kolom ke posisi tertentu | `ALTER TABLE table_name ADD COLUMN column_name INTEGER NOT NULL first;` | Amazon Redshift tidak mendukung penambahan kolom ke posisi tertentu menggunakan kata kunci `first` atau `after`. Jika urutan kolom dalam tabel target tidak penting, tambahkan kolom ke akhir tabel menggunakan perintah |

| Operasi | Contoh | Alternatif |
| --- | --- | --- |
| | | yang lebih sederhana:<br>`ALTER TABLE table_name ADD COLUMN column_name column_type ;` |

| Operasi | Contoh | Alternatif |
| --- | --- | --- |
| Menambahkan kolom stempel waktu dengan `CURRENT_TIMESTAMP` default | `ALTER TABLE table_name`<br>`  ADD COLUMN column_name  TIMESTAMP`<br>`  NOT NULL DEFAULT CURRENT_TIMESTAMP;` | Nilai `CURRENT_TIMESTAMP` untuk baris tabel yang ada dihitung oleh Aurora MySQL dan tidak dapat disimulasikan di Amazon Redshift tanpa sinkronisasi ulang data tabel penuh.<br><br>Jika memungkinkan, alihkan nilai default ke konstanta literal seperti `2023-01-01 00:00:15` untuk menghindari latensi dalam ketersediaan tabel. |

| Operasi | Contoh | Alternatif |
| --- | --- | --- |
| Melakukan beberapa operasi kolom dalam satu perintah | `ALTER TABLE table_name ADD COLUMN column_1, RENAME COLUMN column_2 TO column_3;` | Pertimbangkan untuk membagi perintah menjadi dua operasi terpisah, `ADD` dan `RENAME`, yang tidak memerlukan sinkronisasi ulang. |

## Masalah gagal integrasi untuk integrasi Amazon SageMaker lakehouse zero-ETL

Jika Anda mengalami masalah dengan integrasi nol-ETL yang ada dengan Amazon SageMaker lakehouse, satu-satunya resolusi adalah menghapus integrasi dan membuat yang baru. Tidak seperti AWS layanan lain, integrasi nol-ETL tidak mendukung operasi penyegaran atau sinkronisasi ulang.

Untuk mengatasi masalah integrasi:

1. Hapus integrasi nol-ETL yang bermasalah menggunakan konsol, CLI, atau API.
2. Verifikasi bahwa database sumber dan konfigurasi gudang data target sudah benar.
3. Buat integrasi nol-ETL baru dengan konfigurasi yang sama atau diperbarui.

Proses ini akan menghasilkan inisialisasi ulang lengkap dari pipeline data, yang mungkin memakan waktu tergantung pada ukuran database sumber Anda.

# Perubahan DDL ada di Amazon Redshift sebelum transaksi DDL selesai untuk Aurora PostgreSQL

Perubahan DDL dapat muncul di Amazon Redshift sebelum operasi DDL selesai dalam integrasi Aurora PostgreSQL Zero-ETL. Lihat informasi yang lebih lengkap di Operasi DDL untuk Aurora PostgreSQL.

# Menggunakan Aurora Serverless v2

Aurora Serverless v2 merupakan konfigurasi penskalaan otomatis sesuai permintaan untuk Amazon Aurora. Aurora Serverless v2 membantu mengotomatiskan proses pemantauan beban kerja dan menyesuaikan kapasitas database Anda. Kapasitas disesuaikan secara otomatis berdasarkan permintaan aplikasi. Anda hanya dikenai biaya untuk sumber daya yang menggunakan klaster DB Anda. Dengan demikian, Aurora Serverless v2 dapat membantu Anda tetap dalam anggaran dan menghindari membayar sumber daya komputer yang tidak Anda gunakan.

Jenis otomatisasi ini sangat berharga untuk basis data multi-penghuni, basis data terdistribusi, sistem pengembangan dan pengujian, serta lingkungan lain dengan beban kerja yang sangat bervariasi dan tidak dapat diprediksi.

Topik


- Aurora Serverless v2 kasus penggunaan
- Keuntungan dari Aurora Serverless v2
- Bagaimana Aurora Serverless v2 cara kerja
- Persyaratan dan batasan untuk Aurora Serverless v2
- Membuat cluster DB yang menggunakan Aurora Serverless v2
- Mengelola Aurora Serverless v2 Klaster DB
- Performa dan penskalaan untuk Aurora Serverless v2
- Penskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2
- Migrasi ke Aurora Serverless v2


## Aurora Serverless v2 kasus penggunaan

Aurora Serverless v2 mendukung banyak jenis beban kerja database. Hal ini berkisar dari lingkungan pengembangan dan pengujian, hingga situs web dan aplikasi yang memiliki beban kerja yang tidak terduga, hingga aplikasi bisnis yang paling menuntut dan kritis yang membutuhkan skala dan ketersediaan tinggi.

Aurora Serverless v2 sangat berguna untuk kasus penggunaan berikut:

- Beban kerja variabel – Anda menjalankan beban kerja yang mengalami peningkatan aktivitas secara tiba-tiba dan tidak dapat diprediksi. Contohnya adalah situs info lalu lintas yang mengalami

lonjakan aktivitas saat hujan mulai turun. Contoh yang lain adalah situs e-commerce yang mengalami peningkatan lalu lintas ketika Anda menawarkan obral atau promosi khusus. Dengan Aurora Serverless v2, database Anda secara otomatis menskalakan kapasitas untuk memenuhi kebutuhan beban puncak aplikasi dan menskalakan kembali saat lonjakan aktivitas selesai. Dengan Aurora Serverless v2, Anda tidak perlu lagi menyediakan kapasitas puncak atau rata-rata. Anda dapat menentukan batas kapasitas atas untuk menangani situasi terburuk, dan kapasitas tersebut tidak akan digunakan kecuali jika diperlukan.

Perincian penskalaan di Aurora Serverless v2 membantu Anda untuk mencocokkan kapasitas erat dengan kebutuhan database Anda. Untuk klaster terprovisi, penskalaan memerlukan penambahan instans DB yang sama sekali baru. Untuk Aurora Serverless v1 cluster, penskalaan membutuhkan penggandaan jumlah unit kapasitas Aurora (ACUs) untuk cluster, seperti dari 16 menjadi 32 atau 32 menjadi 64. Sebaliknya, Aurora Serverless v2 dapat menambahkan setengah ACU ketika hanya sedikit lebih banyak kapasitas yang dibutuhkan. Ini dapat menambahkan 0,5, 1, 1,5, 2, atau setengah tambahan ACUs berdasarkan kapasitas tambahan yang diperlukan untuk menangani peningkatan beban kerja. Dan itu dapat menghapus 0,5, 1, 1,5, 2, atau tambahan setengah- ACUs ketika beban kerja menurun dan kapasitas itu tidak lagi diperlukan.

- Aplikasi multi-penyewa - Dengan Aurora Serverless v2, Anda tidak perlu mengelola kapasitas basis data satu per satu untuk setiap aplikasi dalam armada Anda. Aurora Serverless v2 mengelola kapasitas basis data individu untuk Anda.

  Anda dapat membuat klaster untuk setiap penghuni. Dengan demikian, Anda dapat menggunakan fitur seperti kloning, pemulihan snapshot, dan basis data global Aurora untuk meningkatkan ketersediaan tinggi dan pemulihan bencana yang sesuai untuk setiap penghuni.

  Setiap penghuni mungkin memiliki periode sibuk dan idle tertentu tergantung waktu dalam sehari, periode dalam setahun, acara promosi, dan sebagainya. Setiap klaster dapat memiliki rentang kapasitas yang luas. Dengan demikian, klaster dengan aktivitas rendah dikenai biaya instans DB minimal. Setiap klaster dapat dengan cepat menaikkan skala untuk menangani periode aktivitas tinggi.
- Aplikasi baru – Anda men-deploy aplikasi baru dan Anda tidak yakin dengan ukuran instans DB yang Anda butuhkan. Dengan menggunakan Aurora Serverless v2, Anda dapat mengatur cluster dengan satu atau banyak instans DB dan memiliki database autoscale untuk persyaratan kapasitas aplikasi Anda.
- Aplikasi mixed-use — Misalkan Anda memiliki aplikasi pemrosesan transaksi online (OLTP), tetapi Anda secara berkala mengalami lonjakan lalu lintas kueri. Dengan menentukan tingkatan promosi untuk Aurora Serverless v2 Instans DB di cluster, Anda dapat mengonfigurasi cluster Anda

sehingga instans DB pembaca dapat menskalakan secara independen dari instans DB penulis untuk menangani beban tambahan. Saat lonjakan penggunaan mereda, instans DB pembaca menurunkan skala kembali agar sesuai dengan kapasitas instans DB penulis.

- Perencanaan kapasitas – Misalkan Anda biasanya menyesuaikan kapasitas basis data Anda, atau memverifikasi kapasitas basis data optimal untuk beban kerja Anda, dengan memodifikasi kelas instans DB untuk semua instans DB dalam sebuah klaster. Dengan Aurora Serverless v2, Anda dapat menghindari overhead administratif ini. Anda dapat menentukan kapasitas minimum dan maksimum yang sesuai dengan menjalankan beban kerja dan memeriksa seberapa besar skala instans DB sebenarnya.

  Anda dapat memodifikasi instans DB yang ada dari yang disediakan ke Aurora Serverless v2 atau dari Aurora Serverless v2 untuk disediakan. Anda tidak perlu membuat klaster baru atau instans DB baru dalam kasus seperti itu.

  Dengan basis data global Aurora, Anda mungkin tidak memerlukan kapasitas untuk klaster sekunder sebanyak untuk klaster primer. Anda dapat menggunakan Aurora Serverless v2 Instans DB di cluster sekunder. Dengan demikian, kapasitas klaster dapat dinaikkan skalanya jika wilayah sekunder dipromosikan dan mengambil alih beban kerja aplikasi Anda.
- Pengembangan dan pengujian — Selain menjalankan aplikasi Anda yang paling menuntut, Anda juga dapat menggunakan Aurora Serverless v2 untuk lingkungan pengembangan dan pengujian. Dengan Aurora Serverless v2, Anda dapat membuat instance DB dengan kapasitas minimum rendah alih-alih menggunakan kelas instans db.t* DB yang dapat diburstable. Anda dapat mengatur kapasitas maksimum cukup tinggi sehingga instans DB tersebut masih dapat menjalankan beban kerja yang substansial tanpa kehabisan memori. Ketika basis data tidak digunakan, semua instans DB diturunkan skalanya untuk menghindari biaya yang tidak perlu.

  > **Tip**
  >
  > Untuk membuatnya nyaman digunakan Aurora Serverless v2 di lingkungan pengembangan dan pengujian, AWS Management Console menyediakan pintasan Easy create saat Anda membuat cluster baru. Jika Anda memilih opsi Dev/Test, Aurora membuat cluster dengan Aurora Serverless v2 Instans DB dan rentang kapasitas yang khas untuk sistem pengembangan dan pengujian.

## Penggunaan Aurora Serverless v2 untuk beban kerja yang sudah ada

Misalkan Anda sudah memiliki aplikasi Aurora yang berjalan pada klaster terprovisi. Anda dapat memeriksa bagaimana aplikasi akan bekerja dengan Aurora Serverless v2 dengan menambahkan satu atau lebih Aurora Serverless v2 Instans DB ke cluster yang ada sebagai instans DB pembaca. Anda dapat memeriksa seberapa sering instans DB pembaca dinaikkan dan diturunkan skalanya. Anda dapat menggunakan mekanisme failover Aurora untuk mempromosikan Aurora Serverless v2 Instans DB untuk menjadi penulis dan memeriksa bagaimana menangani beban kerja baca/tulis. Dengan demikian, Anda dapat beralih dengan waktu henti minimal dan tanpa mengubah titik akhir yang digunakan aplikasi klien Anda. Untuk detail tentang prosedur untuk mengonversi cluster yang ada ke Aurora Serverless v2, lihat Migrasi ke Aurora Serverless v2.

# Keuntungan dari Aurora Serverless v2

Aurora Serverless v2 ditujukan untuk beban kerja variabel atau “runcing”. Dengan beban kerja yang tidak dapat diprediksi seperti itu, Anda mungkin mengalami kesulitan dalam merencanakan kapan harus mengubah kapasitas basis data Anda. Anda mungkin juga mengalami kesulitan dalam membuat perubahan kapasitas dengan cukup cepat menggunakan mekanisme yang sudah dikenal seperti menambahkan instance DB atau mengubah kelas instans DB. Aurora Serverless v2 memberikan keuntungan berikut untuk membantu kasus penggunaan seperti:

- Manajemen kapasitas yang lebih sederhana daripada yang disediakan - Aurora Serverless v2 mengurangi upaya untuk merencanakan ukuran instans DB dan mengubah ukuran instans DB saat beban kerja berubah. Layanan ini juga mengurangi upaya untuk mempertahankan kapasitas yang konsisten untuk semua instans DB dalam klaster.
- Penskalaan yang lebih cepat dan lebih mudah selama periode aktivitas tinggi — Aurora Serverless v2 skala kapasitas komputasi dan memori sesuai kebutuhan, tanpa gangguan pada transaksi klien atau beban kerja Anda secara keseluruhan. Kemampuan untuk menggunakan instans DB pembaca dengan Aurora Serverless v2 membantu Anda memanfaatkan penskalaan horizontal selain penskalaan vertikal. Kemampuan untuk menggunakan database global Aurora berarti Anda dapat menyebarkan Aurora Serverless v2 baca beban kerja di beberapa Wilayah AWS. Kemampuan ini lebih praktis daripada mekanisme penskalaan untuk klaster terprovisi. Ini juga lebih cepat dan lebih terperinci daripada kemampuan penskalaan di Aurora Serverless v1.
- Hemat biaya selama periode aktivitas rendah - Aurora Serverless v2 membantu Anda menghindari penyediaan instans DB Anda secara berlebihan. Aurora Serverless v2 menambahkan sumber daya secara bertahap saat instans DB meningkat. Anda hanya membayar untuk sumber daya database yang Anda konsumsi. Aurora Serverless v2 penggunaan sumber daya diukur berdasarkan per

detik. Dengan demikian, ketika instans DB menurunkan skala, penggunaan sumber daya yang berkurang segera didaftarkan.

- Paritas fitur yang lebih besar dengan disediakan - Anda dapat menggunakan banyak fitur Aurora dengan Aurora Serverless v2 yang tidak tersedia untuk Aurora Serverless v1. Misalnya, dengan Aurora Serverless v2 Anda dapat menggunakan instans DB pembaca, database global, AWS Identity and Access Management (IAM) otentikasi database, dan Performance Insights. Anda juga dapat menggunakan lebih banyak parameter konfigurasi daripada dengan Aurora Serverless v1.

  Secara khusus, dengan Aurora Serverless v2 Anda dapat memanfaatkan fitur-fitur berikut dari cluster yang disediakan:

  - Instans DB pembaca - Aurora Serverless v2 dapat memanfaatkan instans DB pembaca untuk menskalakan secara horizontal. Ketika klaster berisi satu atau beberapa instans DB pembaca, klaster dapat segera melakukan failover jika terjadi masalah dengan instans DB penulis. Ini adalah kemampuan yang tidak tersedia dengan Aurora Serverless v1.
  - Cluster multi-AZ — Anda dapat mendistribusikan Aurora Serverless v2 Instance DB dari sebuah cluster di beberapa Availability Zones (AZs). Menyiapkan klaster Multi-AZ akan membantu memastikan kelangsungan bisnis bahkan dalam kasus masalah yang jarang terjadi yang memengaruhi seluruh AZ. Ini adalah kemampuan yang tidak tersedia dengan Aurora Serverless v1.
  - Database global — Anda dapat menggunakan Aurora Serverless v2 dikombinasikan dengan database global Aurora untuk membuat salinan read-only tambahan dari cluster Anda di tempat lain Wilayah AWS untuk tujuan pemulihan bencana.
  - RDSProxy — Anda dapat menggunakan Amazon RDS Proxy untuk memungkinkan aplikasi Anda mengumpulkan dan berbagi koneksi database untuk meningkatkan kemampuan mereka dalam skala.
- Penskalaan yang lebih cepat, lebih granular, kurang mengganggu dibandingkan Aurora Serverless v1 – Aurora Serverless v2 dapat naik dan turun lebih cepat. Penskalaan dapat mengubah kapasitas hanya 0,5ACUs, alih-alih menggandakan atau mengurangi separuh jumlah. ACUs Penskalaan biasanya terjadi tanpa jeda dalam pemrosesan sama sekali. Penskalaan tidak melibatkan peristiwa yang harus Anda waspadai, seperti halnya Aurora Serverless v1. Penskalaan dapat terjadi saat SQL pernyataan berjalan dan transaksi terbuka, tanpa perlu menunggu titik tenang.

# Bagaimana Aurora Serverless v2 cara kerja

Ikhtisar berikut menjelaskan bagaimana Aurora Serverless v2 bekerja.

Topik


- Aurora Serverless v2 selayang pandang
- Konfigurasi untuk cluster Aurora DB
- Aurora Serverless v2 kapasitas
- Aurora Serverless v2 penskalaan
- Aurora Serverless v2 dan ketersediaan tinggi
- Aurora Serverless v2 dan penyimpanan
- Parameter konfigurasi untuk klaster Aurora


## Aurora Serverless v2 selayang pandang

Amazon Aurora Serverless v2 cocok untuk beban kerja yang paling menuntut dan sangat bervariasi. Misalnya, penggunaan basis data Anda mungkin berat untuk waktu yang singkat, diikuti dengan aktivitas ringan dalam waktu lama atau tidak ada aktivitas sama sekali. Beberapa contohnya adalah situs web retail, game, atau olahraga dengan peristiwa promosi berkala, dan basis data yang menghasilkan laporan jika diperlukan. Contoh lainnya adalah lingkungan pengembangan dan pengujian, serta aplikasi baru yang penggunaannya dapat meningkat dengan cepat. Untuk kasus seperti ini dan banyak kasus lainnya, mengonfigurasi kapasitas dengan benar sebelumnya tidak selalu dapat dilakukan dengan model terprovisi. Hal ini juga dapat mengakibatkan biaya yang lebih tinggi jika Anda menetapkan penyediaan yang berlebih dan memiliki kapasitas yang tidak Anda gunakan.

Sebaliknya, klaster terprovisi Aurora cocok untuk beban kerja yang stabil. Dengan klaster terprovisi, Anda memilih kelas instans DB yang memiliki jumlah standar memori, daya CPU, bandwidth I/O, dan sebagainya. Jika beban kerja Anda berubah, Anda secara manual memodifikasi kelas instans penulis dan pembaca Anda. Model terprovisi berfungsi dengan baik saat Anda dapat menyesuaikan kapasitas sebelum pola konsumsi yang diharapkan dan Anda sanggup mengalami pemadaman singkat saat mengubah kelas instans penulis dan pembaca di klaster Anda.

Aurora Serverless v2 dirancang dari bawah ke atas untuk mendukung cluster DB tanpa server yang dapat diskalakan secara instan. Aurora Serverless v2 direkayasa untuk memberikan tingkat keamanan dan isolasi yang sama seperti penulis dan pembaca yang disediakan. Aspek-aspek ini sangat penting dalam lingkungan cloud nirserver multi-penghuni. Mekanisme penskalaan dinamis memiliki overhead yang sangat sedikit sehingga dapat merespons dengan cepat perubahan beban kerja basis data. Mekanisme ini juga cukup kuat untuk memenuhi permintaan memproses permintaan yang meningkat secara drastis.

Dengan menggunakan Aurora Serverless v2, Anda dapat membuat cluster Aurora DB tanpa dikunci ke dalam kapasitas database tertentu untuk setiap penulis dan pembaca. Anda menentukan rentang kapasitas minimum dan maksimum. Timbangan Aurora masing-masing Aurora Serverless v2 penulis atau pembaca dalam cluster dalam rentang kapasitas itu. Dengan menggunakan klaster Multi-AZ yang memungkinkan setiap penulis atau pembaca diskalakan secara dinamis, Anda dapat memanfaatkan penskalaan dinamis dan ketersediaan tinggi.

Aurora Serverless v2 menskalakan sumber daya basis data secara otomatis berdasarkan spesifikasi kapasitas minimum dan maksimum Anda. Penskalaan dilakukan dengan cepat karena sebagian besar operasi peristiwa penskalaan mempertahankan penulis atau pembaca di host yang sama. Dalam kasus yang jarang terjadi, sebuah Aurora Serverless v2 penulis atau pembaca dipindahkan dari satu host ke host lainnya, Aurora Serverless v2 mengelola koneksi secara otomatis. Anda tidak perlu mengubah kode aplikasi klien basis data Anda atau string koneksi basis data Anda.

Dengan Aurora Serverless v2, seperti halnya cluster yang disediakan, kapasitas penyimpanan dan kapasitas komputasi terpisah. Ketika kita mengacu pada Aurora Serverless v2 kapasitas dan penskalaan, selalu kapasitas komputasi yang meningkat atau menurun. Dengan demikian, klaster Anda dapat berisi banyak terabyte data bahkan ketika kapasitas CPU dan memori diturunkan skalanya ke tingkat rendah.

Alih-alih menyediakan dan mengelola server basis data, Anda menentukan kapasitas basis data. Untuk detail tentang Aurora Serverless v2 kapasitas, lihatAurora Serverless v2 kapasitas. Kapasitas sebenarnya dari masing-masing Aurora Serverless v2 penulis atau pembaca bervariasi dari waktu ke waktu, tergantung pada beban kerja Anda. Untuk detail tentang mekanisme tersebut, lihat Aurora Serverless v2 penskalaan.

> **Important**
>
> Dengan Aurora Serverless v1, cluster Anda memiliki satu ukuran kapasitas komputasi yang dapat menskalakan antara nilai kapasitas minimum dan maksimum. Dengan Aurora Serverless v2, cluster Anda dapat berisi pembaca selain penulis. Masing-masing Aurora Serverless v2 penulis dan pembaca dapat menskalakan antara nilai kapasitas minimum dan maksimum. Dengan demikian, total kapasitas Anda Aurora Serverless v2 cluster tergantung pada rentang kapasitas yang Anda tentukan untuk cluster DB Anda dan jumlah penulis dan pembaca di cluster. Pada waktu tertentu, Anda hanya dikenakan biaya untuk Aurora Serverless v2 kapasitas yang sedang digunakan secara aktif di cluster Aurora DB Anda.

# Konfigurasi untuk cluster Aurora DB

Untuk setiap cluster Aurora DB Anda, Anda dapat memilih kombinasi Aurora Serverless v2 kapasitas, kapasitas yang disediakan, atau keduanya.

Anda dapat mengatur cluster yang berisi keduanya Aurora Serverless v2 dan kapasitas yang disediakan, yang disebut cluster konfigurasi campuran. Misalnya, anggaplah Anda membutuhkan lebih banyak kapasitas baca/tulis daripada yang tersedia untuk Aurora Serverless v2 penulis. Dalam hal ini, Anda dapat mengatur klaster dengan penulis terprovisi yang sangat besar. Dalam hal ini, Anda masih dapat menggunakan Aurora Serverless v2 untuk para pembaca. Atau anggaplah beban kerja tulis untuk klaster Anda bervariasi, tetapi beban kerja baca stabil. Dalam hal ini, Anda dapat mengatur cluster Anda dengan Aurora Serverless v2 penulis dan satu atau lebih pembaca yang disediakan.

Anda juga dapat mengatur cluster DB di mana semua kapasitas dikelola oleh Aurora Serverless v2. Untuk melakukan ini, Anda dapat membuat cluster baru dan menggunakan Aurora Serverless v2 dari awal. Atau Anda dapat mengganti semua kapasitas yang disediakan di cluster yang ada dengan Aurora Serverless v2. Misalnya, beberapa jalur pemutakhiran dari versi mesin yang lebih lama memerlukan dimulai dengan penulis yang disediakan dan menggantinya dengan Aurora Serverless v2 penulis. Untuk prosedur membuat cluster DB baru dengan Aurora Serverless v2 atau untuk mengganti cluster DB yang ada ke Aurora Serverless v2, lihat Membuat Aurora Serverless v2 klaster DB. danBeralih dari klaster yang disediakan ke Aurora Serverless v2.

Jika Anda tidak menggunakan Aurora Serverless v2 sama sekali dalam cluster DB, semua penulis dan pembaca di cluster DB disediakan. Ini adalah jenis klaster DB terlama dan paling umum yang sudah dikenal oleh sebagian besar pengguna. Bahkan, sebelumnya Aurora Serverless, tidak ada nama khusus untuk cluster Aurora DB semacam ini. Kapasitas terprovisi bersifat konstan. Biayanya relatif mudah untuk diperkirakan. Namun, Anda harus memprediksi sebelumnya berapa banyak kapasitas yang Anda butuhkan. Dalam beberapa kasus, prediksi Anda mungkin tidak akurat atau kebutuhan kapasitas Anda mungkin berubah. Dalam kasus ini, klaster DB Anda dapat mengalami kekurangan penyediaan (lebih lambat dari yang Anda inginkan) atau kelebihan penyediaan (lebih mahal dari yang Anda inginkan).

## Aurora Serverless v2 kapasitas

Satuan ukuran untuk Aurora Serverless v2 adalah unit kapasitas Aurora (ACU). Aurora Serverless v2 kapasitas tidak terkait dengan kelas instans DB yang Anda gunakan untuk kluster yang disediakan.

Setiap ACU adalah kombinasi dari sekitar 2 gibibyte (GiB) memori, CPU yang sesuai, dan jaringan. Anda menentukan rentang kapasitas basis data menggunakan unit ukur ini. Metrik `ACUUtilization` dan `ServerlessDatabaseCapacity` membantu Anda menentukan berapa banyak kapasitas yang sebenarnya digunakan basis data Anda dan di mana kapasitas tersebut berada dalam rentang yang ditentukan.

Setiap saat, setiap Aurora Serverless v2 Penulis atau pembaca DB memiliki kapasitas. Kapasitas direpresentasikan sebagai angka floating-point yang mewakili. ACUs Kapasitas ini meningkat atau menurun setiap kali penulis atau pembaca diskalakan. Nilai ini diukur setiap detik. Untuk setiap cluster DB tempat Anda ingin menggunakan Aurora Serverless v2, Anda menentukan rentang kapasitas: nilai kapasitas minimum dan maksimum yang masing-masing Aurora Serverless v2 penulis atau pembaca dapat menskalakan antara. Rentang kapasitasnya sama untuk masing-masing Aurora Serverless v2 penulis atau pembaca dalam cluster DB. Masing-masing Aurora Serverless v2 penulis atau pembaca memiliki kapasitasnya sendiri, jatuh di suatu tempat dalam kisaran itu.

Tabel berikut menunjukkan Aurora Serverless v2 rentang kapasitas yang didukung untuk Aurora MySQL dan Aurora PostgreSQL.

| Rentang kapasitas (ACUs) | Versi yang didukung Aurora MySQL | Aurora PostgreSQL versi yang didukung |
|---|---|---|
| 0,5—128 | 3.02.0 dan lebih tinggi | 13.6 dan lebih tinggi, 14.3 dan lebih tinggi, 15.2 dan lebih tinggi, 16.1 dan lebih tinggi |
| 0,5—256 | 3.06.0 dan lebih tinggi | 13.13 dan lebih tinggi, 14.10 dan lebih tinggi, 15.5 dan lebih tinggi, 16.1 dan lebih tinggi |
| 0—256 | 3.08.0 dan lebih tinggi | 13.15 dan lebih tinggi, 14.12 dan lebih tinggi, 15.7 dan lebih tinggi, 16.3 dan lebih tinggi |

Yang terkecil Aurora Serverless v2 kapasitas yang dapat Anda tentukan adalah 0 ACUs, untuk Aurora Serverless v2 versi yang mendukung fitur jeda otomatis. Anda dapat menentukan angka yang lebih tinggi jika kapasitasnya kurang dari atau sama dengan nilai kapasitas maksimum. Mengatur kapasitas minimum ke angka kecil akan memungkinkan klaster DB yang berbeban ringan

mengonsumsi sumber daya komputasi minimal. Pada saat yang sama, klaster tersebut tetap siap untuk menerima koneksi dengan segera dan menaikkan skalanya ketika menjadi sibuk.

Kami merekomendasikan pengaturan kapasitas minimum ke sebuah nilai yang memungkinkan setiap penulis atau pembaca DB mempertahankan rangkaian aplikasi yang beroperasi di pool buffer. Dengan begitu, konten pool buffer tidak dibuang selama periode idle. Untuk semua pertimbangan saat memilih nilai kapasitas minimum, lihat Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster. Untuk semua pertimbangan saat memilih nilai kapasitas maksimum, lihat Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster.

Bergantung pada bagaimana Anda mengkonfigurasi pembaca dalam penyebaran multi-AZ, kapasitas mereka dapat dikaitkan dengan kapasitas penulis atau secara mandiri. Untuk detail tentang cara melakukannya, lihat Aurora Serverless v2 penskalaan.

Pemantauan Aurora Serverless v2 melibatkan pengukuran nilai kapasitas untuk penulis dan pembaca di cluster DB Anda dari waktu ke waktu. Jika basis data Anda tidak menurunkan skalanya ke kapasitas minimum, Anda dapat mengambil tindakan seperti menyesuaikan kapasitas minimum dan mengoptimalkan aplikasi basis data Anda. Jika basis data Anda secara konsisten mencapai kapasitas maksimumnya, Anda dapat mengambil tindakan seperti meningkatkan kapasitas maksimum. Anda juga dapat mengoptimalkan aplikasi basis data Anda dan menyebarkan beban kueri ke lebih banyak pembaca.

Biaya untuk Aurora Serverless v2 kapasitas diukur dalam hal jam Acu-jam. Untuk informasi tentang bagaimana Aurora Serverless v2 biaya dihitung, lihat halaman harga Aurora.

Misalkan jumlah total penulis dan pembaca di cluster Anda adalah*n*. Dalam hal ini, cluster mengkonsumsi kira-kira *n* x *minimum ACUs* ketika Anda tidak menjalankan operasi database apa pun. Aurora sendiri mungkin menjalankan operasi pemantauan atau pemeliharaan yang menyebabkan sejumlah kecil beban. Klaster tersebut mengonsumsi tidak lebih dari *n* x *maximum ACUs* ketika basis data berjalan pada kapasitas penuh.

Untuk detail selengkapnya tentang memilih nilai ACU minimum dan maksimum yang sesuai, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora. Nilai ACU minimum dan maksimum yang Anda tentukan juga memengaruhi cara kerja beberapa parameter konfigurasi Aurora Aurora Serverless v2. Untuk detail tentang interaksi antara rentang kapasitas dan parameter konfigurasi, lihatMenggunakan grup parameter untuk Aurora Serverless v2.

# Aurora Serverless v2 penskalaan

Untuk masing-masing Aurora Serverless v2 penulis atau pembaca, Aurora terus melacak pemanfaatan sumber daya seperti CPU, memori, dan jaringan. Pengukuran ini secara kolektif disebut beban. Beban mencakup operasi basis data yang dilakukan oleh aplikasi Anda. Hal ini juga mencakup pemrosesan latar belakang untuk server basis data dan tugas administratif Aurora. Ketika kapasitas dibatasi oleh salah satu dari ini, Aurora Serverless v2 skala naik. Aurora Serverless v2 juga meningkatkan skala ketika mendeteksi masalah kinerja yang dapat diselesaikan dengan melakukannya. Anda dapat memantau pemanfaatan sumber daya dan bagaimana pengaruhnya Aurora Serverless v2 penskalaan dengan menggunakan prosedur di CloudWatch Metrik Amazon penting untuk Aurora Serverless v2 danMemantau performa Aurora Serverless v2 dengan Wawasan Performa.

Beban dapat bervariasi di seluruh penulis dan pembaca di klaster DB Anda. Penulis menangani semua laporan bahasa definisi data (DDL), seperti `CREATE TABLE`, `ALTER TABLE`, dan `DROP TABLE`. Penulis juga menangani semua laporan bahasa manipulasi data (DML), seperti `INSERT` dan `UPDATE`. Pembaca dapat memproses pernyataan hanya baca, seperti kueri `SELECT`.

Penskalaan adalah operasi yang meningkat atau menurun Aurora Serverless v2 kapasitas untuk database Anda. Dengan Aurora Serverless v2, setiap penulis dan pembaca memiliki nilai kapasitasnya sendiri saat ini, diukur dalam ACUs. Aurora Serverless v2 skala penulis atau pembaca hingga kapasitas yang lebih tinggi ketika kapasitas saat ini terlalu rendah untuk menangani beban. Operasi ini menurunkan skala penulis atau pembaca ke kapasitas yang lebih rendah ketika kapasitas saat ini lebih tinggi dari yang dibutuhkan.

Tidak seperti Aurora Serverless v1, yang menskalakan dengan menggandakan kapasitas setiap kali cluster DB mencapai ambang batas, Aurora Serverless v2 dapat meningkatkan kapasitas secara bertahap. Ketika permintaan beban kerja Anda mulai mencapai kapasitas database saat ini dari seorang penulis atau pembaca, Aurora Serverless v2 meningkatkan jumlah ACUs untuk penulis atau pembaca itu. Aurora Serverless v2 skala kapasitas dalam peningkatan yang diperlukan untuk memberikan kinerja terbaik untuk sumber daya yang dikonsumsi. Penskalaan terjadi secara bertahap sekecil 0,5. ACUs Semakin besar kapasitas saat ini, semakin besar inkremen penskalaan dan akibatnya semakin cepat penskalaan dapat terjadi.

Karena Aurora Serverless v2 penskalaan sangat sering, granular, dan tidak mengganggu, tidak menyebabkan peristiwa diskrit seperti itu AWS Management Console Aurora Serverless v1 tidak. Sebagai gantinya, Anda dapat mengukur CloudWatch metrik Amazon seperti `ServerlessDatabaseCapacity` dan `ACUUtilization` dan melacak nilai minimum, maksimum,

dan rata-ratanya dari waktu ke waktu. Untuk mempelajari selengkapnya tentang metrik Aurora, lihat Memantau metrik di klaster Amazon Aurora. Untuk tips tentang pemantauan Aurora Serverless v2, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Anda dapat memilih untuk membuat pembaca diskalakan pada saat yang sama dengan penulis terkait, atau secara independen dari penulis. Anda dapat melakukannya dengan menentukan tingkat promosi untuk pembaca tersebut.

- Pembaca di tingkat promosi 0 dan 1 akan diskalakan pada saat yang sama dengan penulis. Perilaku penskalaan tersebut membuat pembaca di tingkat prioritas 0 dan 1 cocok untuk mendukung ketersediaan. Hal ini karena pembaca tersebut selalu diatur ukurannya sesuai dengan kapasitas yang tepat untuk mengambil alih beban kerja dari penulis jika terjadi failover.
- Pembaca di tingkat promosi 2–15 akan diskalakan secara independen dari penulis. Setiap pembaca tetap berada dalam nilai ACU minimum dan maksimum yang Anda tentukan untuk klaster Anda. Ketika pembaca diskalakan secara independen dari DB penulis terkait, pembaca tersebut bisa menjadi idle dan menurunkan skalanya sementara penulis terus memproses volume transaksi yang tinggi. Pembaca tersebut masih tersedia sebagai target failover jika tidak ada pembaca lain yang tersedia di tingkat promosi yang lebih rendah. Namun, jika dipromosikan menjadi penulis, pembaca tersebut mungkin perlu dinaikkan skalanya untuk menangani beban kerja penuh dari penulis.

Untuk detail tentang tingkat promosi, lihat Memilih tingkat promosi untuk Aurora Serverless v2 pembaca.

Gagasan poin penskalaan dan periode batas waktu terkait dari Aurora Serverless v1 jangan terapkan di Aurora Serverless v2. Aurora Serverless v2 penskalaan dapat terjadi saat koneksi database terbuka, sementara transaksi SQL sedang dalam proses, sementara tabel dikunci, dan sementara tabel sementara sedang digunakan. Aurora Serverless v2 tidak menunggu titik tenang untuk memulai penskalaan. Penskalaan tidak mengganggu operasi basis data apa pun yang sedang berlangsung.

Jika beban kerja Anda membutuhkan lebih banyak kapasitas baca daripada yang tersedia dengan satu penulis dan satu pembaca, Anda dapat menambahkan beberapa Aurora Serverless v2 pembaca ke cluster. Masing-masing Aurora Serverless v2 pembaca dapat menskalakan dalam kisaran nilai kapasitas minimum dan maksimum yang Anda tentukan untuk cluster DB Anda. Anda dapat menggunakan titik akhir pembaca klaster untuk mengarahkan sesi hanya baca ke pembaca dan mengurangi beban pada penulis.

Apakah Aurora Serverless v2 melakukan penskalaan, dan seberapa cepat penskalaan terjadi setelah dimulai, juga tergantung pada pengaturan ACU minimum dan maksimum untuk cluster. Selain itu, hal tersebut bergantung pada apakah pembaca dikonfigurasi untuk diskalakan bersama dengan penulis atau secara independen. Untuk detail tentang faktor-faktor yang mempengaruhi Aurora Serverless v2 penskalaan, lihatPerforma dan penskalaan untuk Aurora Serverless v2.

### Penskalaan ke Nol

Dalam versi Aurora MySQL dan Aurora PostgreSQL baru-baru ini, Aurora Serverless v2 penulis dan pembaca dapat menurunkan skala hingga nol ACUs. Kami menyebut kemampuan ini sebagai jeda dan lanjutkan otomatis, atau jeda otomatis. Anda dapat memilih apakah akan mengizinkan perilaku ini dengan menentukan nilai nol atau bukan nol untuk kapasitas minimum. Anda juga dapat memilih berapa lama untuk menunggu sebelum Aurora Serverless v2 contoh jeda. Untuk informasi tentang versi mana yang memiliki kemampuan ini, lihatAurora Serverless v2 kapasitas. Untuk informasi tentang cara menggunakannya secara efektif, lihatPenskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2.

Dalam versi Aurora MySQL dan Aurora PostgreSQL yang lebih lama, idle Aurora Serverless v2 penulis dan pembaca dapat menurunkan ke nilai ACU minimum yang Anda tentukan untuk cluster, tetapi tidak sampai ke nol ACUs. Dalam hal ini, nol ACUs tidak tersedia sebagai pilihan saat Anda mengatur rentang kapasitas. Perilaku itu berbeda dari Aurora Serverless v1, yang dapat berhenti setelah periode kemalasan, tetapi kemudian membutuhkan waktu untuk melanjutkan ketika Anda membuka koneksi baru.

Saat cluster DB Anda dengan Aurora Serverless v2 kapasitas tidak diperlukan untuk beberapa waktu, Anda juga dapat menghentikan dan memulai seluruh cluster, sama seperti dengan cluster DB yang disediakan. Teknik ini paling tepat untuk pengembangan dan pengujian sistem, di mana mereka mungkin tidak diperlukan selama berjam-jam pada suatu waktu, dan kecepatan melanjutkan cluster tidak penting. Fitur stop/start cluster tersedia untuk semua Aurora Serverless v2 versi. Untuk informasi selengkapnya tentang fitur tersebut, lihatMenghentikan dan memulai klaster DB Amazon Aurora.

## Aurora Serverless v2 dan ketersediaan tinggi

Cara untuk menetapkan ketersediaan tinggi untuk klaster DB Aurora adalah dengan menjadikannya sebagai klaster DB Multi-AZ. Klaster DB Aurora Multi-AZ memiliki kapasitas komputasi yang tersedia setiap saat di lebih dari satu Zona Ketersediaan (AZ). Konfigurasi tersebut membuat basis data Anda tetap aktif dan berjalan bahkan jika terjadi pemadaman yang signifikan. Aurora melakukan failover otomatis jika terjadi masalah yang memengaruhi penulis atau bahkan seluruh AZ. Dengan

Aurora Serverless v2, Anda dapat memilih kapasitas komputasi siaga untuk meningkatkan dan menurunkan bersama dengan kapasitas penulis. Dengan begitu, kapasitas komputasi di AZ kedua siap mengambil alih beban kerja saat ini kapan saja. Pada saat yang sama, kapasitas komputasi di semua AZs dapat menurunkan skala ketika database menganggur. Untuk detail tentang cara kerja Aurora Wilayah AWS dan Availability Zones, lihat. Ketersediaan yang tinggi untuk instans DB Aurora

Bagian Aurora Serverless v2 Kemampuan multi-AZ menggunakan pembaca selain penulis. Support untuk pembaca adalah hal baru untuk Aurora Serverless v2 dibandingkan dengan Aurora Serverless v1. Anda dapat menambahkan hingga 15 Aurora Serverless v2 pembaca tersebar di 3 AZs ke cluster Aurora DB.

Untuk aplikasi bisnis penting yang harus tetap tersedia bahkan jika terjadi masalah yang memengaruhi seluruh klaster Anda atau seluruh AWS Wilayah, Anda dapat menyiapkan basis data global Aurora. Anda dapat menggunakan Aurora Serverless v2 kapasitas di cluster sekunder sehingga mereka siap untuk mengambil alih selama pemulihan bencana. Klaster tersebut juga dapat menurunkan skalanya ketika basis data tidak sibuk. Untuk detail tentang basis data global Aurora, lihat Menggunakan Database Global Amazon Aurora.

Aurora Serverless v2 berfungsi seperti disediakan untuk failover dan fitur ketersediaan tinggi lainnya. Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.

Misalkan Anda ingin memastikan ketersediaan maksimum untuk Aurora Serverless v2 klaster. Anda dapat membuat pembaca selain penulis. Jika Anda menetapkan pembaca ke tingkat promosi 0 atau 1, penskalaan apa pun yang terjadi pada penulis juga terjadi pada pembaca. Dengan begitu, pembaca dengan kapasitas yang sama selalu siap mengambil alih untuk penulis jika terjadi failover.

Misalkan Anda ingin menjalankan laporan triwulanan untuk bisnis Anda seiring klaster Anda terus memproses transaksi. Jika Anda menambahkan Aurora Serverless v2 pembaca ke cluster dan menetapkannya ke tingkat promosi dari 2 hingga 15, Anda dapat terhubung langsung ke pembaca itu untuk menjalankan laporan. Bergantung pada berapa banyak memori dan CPU yang diperlukan kueri pelaporan, pembaca dapat menaikkan skala untuk mengakomodasi beban kerja. Pembaca ini kemudian dapat kembali diturunkan skalanya ketika laporannya sudah selesai.

## Aurora Serverless v2 dan penyimpanan

Penyimpanan untuk setiap cluster Aurora DB terdiri dari enam salinan dari semua data Anda, tersebar di tiga. AZs Replikasi data default ini berlaku terlepas dari apakah klaster DB Anda mencakup pembaca selain penulis. Dengan begitu, data Anda aman, bahkan dari masalah yang memengaruhi kapasitas komputasi klaster.

Aurora Serverless v2 penyimpanan memiliki karakteristik keandalan dan daya tahan yang sama seperti yang dijelaskan dalamPenyimpanan Amazon Aurora. Itu karena penyimpanan untuk cluster Aurora DB bekerja sama apakah kapasitas komputasi menggunakan Aurora Serverless v2 atau disediakan.

## Parameter konfigurasi untuk klaster Aurora

Anda dapat menyesuaikan semua parameter konfigurasi cluster dan database yang sama untuk cluster dengan Aurora Serverless v2 kapasitas seperti untuk cluster DB yang disediakan. Namun, beberapa parameter terkait kapasitas ditangani secara berbeda untuk Aurora Serverless v2. Dalam klaster konfigurasi campuran, nilai parameter yang Anda tentukan untuk parameter terkait kapasitas tersebut masih berlaku untuk penulis dan pembaca yang disediakan.

Hampir semua parameter bekerja dengan cara yang sama Aurora Serverless v2 penulis dan pembaca seperti untuk yang disediakan. Hal ini tidak termasuk beberapa parameter yang disesuaikan oleh Aurora secara otomatis selama penskalaan, dan beberapa parameter yang disimpan oleh Aurora pada nilai tetap yang bergantung pada pengaturan kapasitas maksimum.

Misalnya, jumlah memori yang dicadangkan untuk cache buffer meningkat saat penulis atau pembaca dinaikkan skalanya, dan berkurang saat penulis atau pembaca diturunkan skalanya. Dengan begitu, memori dapat dilepaskan ketika basis data Anda tidak sibuk. Sebaliknya, Aurora secara otomatis menetapkan jumlah maksimum koneksi ke nilai yang sesuai berdasarkan pengaturan kapasitas maksimum. Dengan begitu, koneksi aktif tidak terputus jika beban turun dan Aurora Serverless v2 skala ke bawah. Untuk informasi tentang bagaimana Aurora Serverless v2 menangani parameter tertentu, lihatMenggunakan grup parameter untuk Aurora Serverless v2.

# Persyaratan dan batasan untuk Aurora Serverless v2

Saat Anda membuat cluster tempat Anda ingin menggunakan Aurora Serverless v2 Instans DB, perhatikan persyaratan dan batasan berikut.

Topik

- Ketersediaan wilayah dan versi
- Cluster yang menggunakan Aurora Serverless v2 harus memiliki rentang kapasitas yang ditentukan
- Beberapa fitur yang disediakan tidak didukung di Aurora Serverless v2
- Beberapa Aurora Serverless v2 Aspek berbeda dari Aurora Serverless v1

## Ketersediaan wilayah dan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi khusus dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi lebih lanjut tentang versi dan ketersediaan Wilayah dengan Aurora dan Aurora Serverless v2, lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2.

Contoh berikut menunjukkan AWS CLI perintah untuk mengkonfirmasi nilai mesin DB yang tepat yang dapat Anda gunakan Aurora Serverless v2 untuk yang spesifik Wilayah AWS. `--db-instance-class`Parameter untuk Aurora Serverless v2 selalu`db.serverless`. Parameter `--engine` dapat berupa `aurora-mysql` atau `aurora-postgresql`. Gantikan nilai `--region` dan `--engine` yang sesuai untuk mengonfirmasi nilai `--engine-version` yang dapat Anda gunakan. Jika perintah tidak menghasilkan output apa pun, Aurora Serverless v2 tidak tersedia untuk kombinasi Wilayah AWS dan mesin DB itu.

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-
class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text
```

## Cluster yang menggunakan Aurora Serverless v2 harus memiliki rentang kapasitas yang ditentukan

Klaster Aurora harus memiliki atribut `ServerlessV2ScalingConfiguration` sebelum Anda dapat menambahkan instans DB apa pun yang menggunakan kelas instans DB `db.serverless`. Atribut ini menentukan rentang kapasitas. Aurora Serverless v2 Kapasitas berkisar dari minimum 0 unit kapasitas Aurora (ACU) hingga maksimum 256 ACUs, dengan penambahan 0,5 ACU. Nilai minimum yang diizinkan tergantung pada versi Aurora. Setiap ACU menyediakan kapasitas yang setara dengan sekitar 2 gibibyte (GiB) RAM serta CPU dan jaringan terkait. Untuk detail tentang caranya Aurora Serverless v2 menggunakan pengaturan rentang kapasitas, lihatBagaimana Aurora Serverless v2 cara kerja.

Untuk rentang kapasitas yang diizinkan untuk berbagai versi mesin DB, lihatAurora Serverless v2 kapasitas.

Anda dapat menentukan nilai ACU minimum dan maksimum di AWS Management Console saat Anda membuat cluster dan terkait Aurora Serverless v2 contoh DB. Anda juga dapat menentukan opsi `--serverless-v2-scaling-configuration` di AWS CLI. Atau, Anda dapat menentukan parameter `ServerlessV2ScalingConfiguration` dengan API Amazon RDS. Anda dapat menentukan atribut ini saat Anda membuat klaster atau mengubah klaster yang sudah ada. Untuk prosedur dalam mengatur rentang kapasitas, lihat Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster. Untuk pembahasan mendetail tentang cara memilih nilai kapasitas minimum dan maksimum dan bagaimana pengaturan tersebut memengaruhi beberapa parameter basis data, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora.

## Beberapa fitur yang disediakan tidak didukung di Aurora Serverless v2

Fitur berikut dari instans DB yang disediakan Aurora saat ini tidak tersedia Amazon Aurora Serverless v2:

- Stream aktivitas basis data (DAS).
- Manajemen cache klaster untuk Aurora PostgreSQL. Parameter `apg_ccm_enabled` konfigurasi tidak berlaku untuk Aurora Serverless v2 Instans DB.

Beberapa fitur Aurora bekerja dengan Aurora Serverless v2, tetapi dapat menyebabkan masalah jika rentang kapasitas Anda lebih rendah dari yang dibutuhkan untuk persyaratan memori untuk fitur-fitur tersebut dengan beban kerja spesifik Anda. Dalam hal ini, database Anda mungkin tidak berfungsi sebaik biasanya, atau mungkin mengalami out-of-memory kesalahan. Untuk rekomendasi tentang pengaturan rentang kapasitas yang sesuai, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora. Untuk informasi pemecahan masalah jika database Anda menemukan out-of-memory kesalahan karena rentang kapasitas yang salah dikonfigurasi, lihat. Menghindari out-of-memory kesalahan

Aurora Auto Scaling tidak didukung. Jenis penskalaan ini menambahkan pembaca baru untuk menangani beban kerja intensif baca tambahan, berdasarkan penggunaan CPU. Namun, penskalaan berdasarkan penggunaan CPU tidak berarti Aurora Serverless v2. Sebagai alternatif, Anda dapat membuat Aurora Serverless v2 pembaca DB melakukan instans terlebih dahulu dan membiarkannya diperkecil ke kapasitas rendah. Hal tersebut adalah cara yang lebih cepat dan tidak terlalu mengganggu untuk menskalakan kapasitas baca klaster daripada menambahkan instans DB baru secara dinamis.

## Beberapa Aurora Serverless v2 Aspek berbeda dari Aurora Serverless v1

Jika Anda seorang Aurora Serverless v1 pengguna dan ini adalah pertama kalinya Anda menggunakan Aurora Serverless v2, konsultasikan perbedaan antara Aurora Serverless v2 and Aurora Serverless v1 persyaratan untuk memahami bagaimana persyaratan berbeda antara Aurora Serverless v1 and Aurora Serverless v2.

# Membuat cluster DB yang menggunakan Aurora Serverless v2

Untuk membuat cluster Aurora di mana Anda dapat menambahkan Aurora Serverless v2 Instans DB, Anda mengikuti prosedur yang sama seperti diMembuat klaster DB Amazon Aurora. Dengan Aurora Serverless v2, cluster Anda dapat dipertukarkan dengan cluster yang disediakan. Anda dapat memiliki cluster di mana beberapa instans DB digunakan Aurora Serverless v2 dan beberapa instance DB disediakan.

Topik

- Pengaturan untuk Aurora Serverless v2 Klaster DB
- Membuat Aurora Serverless v2 klaster DB.
- Membuat Aurora Serverless v2 penulis contoh DB

## Pengaturan untuk Aurora Serverless v2 Klaster DB

Pastikan bahwa pengaturan awal cluster memenuhi persyaratan yang tercantum dalamPersyaratan dan batasan untuk Aurora Serverless v2. Tentukan pengaturan berikut untuk memastikan bahwa Anda dapat menambahkan Aurora Serverless v2 Instans DB ke cluster:

Wilayah AWS

Buat cluster di Wilayah AWS tempat Aurora Serverless v2 Instans DB tersedia. Untuk detail tentang Wilayah yang tersedia, lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2.

Versi mesin DB

Pilih versi mesin yang kompatibel Aurora Serverless v2. Untuk informasi tentang Aurora Serverless v2 persyaratan versi, lihatPersyaratan dan batasan untuk Aurora Serverless v2.

Kelas instans DB

Jika Anda membuat cluster menggunakan AWS Management Console, Anda memilih kelas instans DB untuk instance DB penulis pada saat yang sama. Pilih kelas instans DB Nirserver. Ketika Anda memilih kelas instans DB tersebut, Anda juga menentukan rentang kapasitas untuk instans DB penulis. Rentang kapasitas yang sama berlaku untuk semua yang lain Aurora Serverless v2 Instans DB yang Anda tambahkan ke cluster itu.

Jika Anda tidak melihat pilihan Tanpa Server untuk kelas instans DB, pastikan Anda memilih versi mesin DB yang didukung. Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2

Bila Anda menggunakan AWS CLI atau Amazon RDSAPI, parameter yang Anda tentukan untuk kelas instans DB adalah`db.serverless`.

Rentang kapasitas

Isi nilai unit kapasitas Aurora minimum dan maksimum (ACU) yang berlaku untuk semua instans DB di cluster. Opsi ini tersedia di halaman konsol Buat klaster dan Tambahkan pembaca saat Anda memilih Nirserver untuk kelas instans DB.

Untuk rentang kapasitas yang diizinkan untuk berbagai versi mesin DB, lihatAurora Serverless v2 kapasitas.

Jika Anda tidak melihat ACU bidang minimum dan maksimum, pastikan Anda memilih kelas instans DB Tanpa Server untuk instance DB penulis.

Jika Anda awalnya membuat cluster dengan instans DB yang disediakan, Anda tidak menentukan minimum dan maksimum. ACUs Dalam hal ini Anda dapat memodifikasi klaster sesudahnya untuk menambahkan pengaturan tersebut. Anda juga dapat menambahkan Aurora Serverless v2 pembaca instance DB ke cluster. Anda menentukan rentang kapasitas sebagai bagian dari proses tersebut.

Sampai Anda menentukan rentang kapasitas untuk klaster Anda, Anda tidak dapat menambahkan apa pun Aurora Serverless v2 Instans DB ke cluster menggunakan AWS CLI or RDSAPI.
Jika Anda mencoba menambahkan Aurora Serverless v2 Misalnya DB, Anda mendapatkan kesalahan. Dalam AWS CLI atau RDS API prosedur, rentang kapasitas diwakili oleh `ServerlessV2ScalingConfiguration` atribut.

Untuk cluster yang berisi lebih dari satu instans DB pembaca, prioritas failover masing-masing Aurora Serverless v2 instans DB pembaca memainkan peran penting dalam bagaimana instans DB itu naik

dan turun. Anda tidak dapat menentukan prioritas saat membuat klaster pada awalnya. Ingat properti ini saat Anda menambahkan instans DB pembaca kedua atau berikutnya ke klaster Anda. Untuk informasi selengkapnya, lihat Memilih tingkat promosi untuk Aurora Serverless v2 pembaca.

## Membuat Aurora Serverless v2 klaster DB.

Anda dapat menggunakan AWS Management Console, AWS CLI, atau RDS API untuk membuat Aurora Serverless v2 klaster DB.

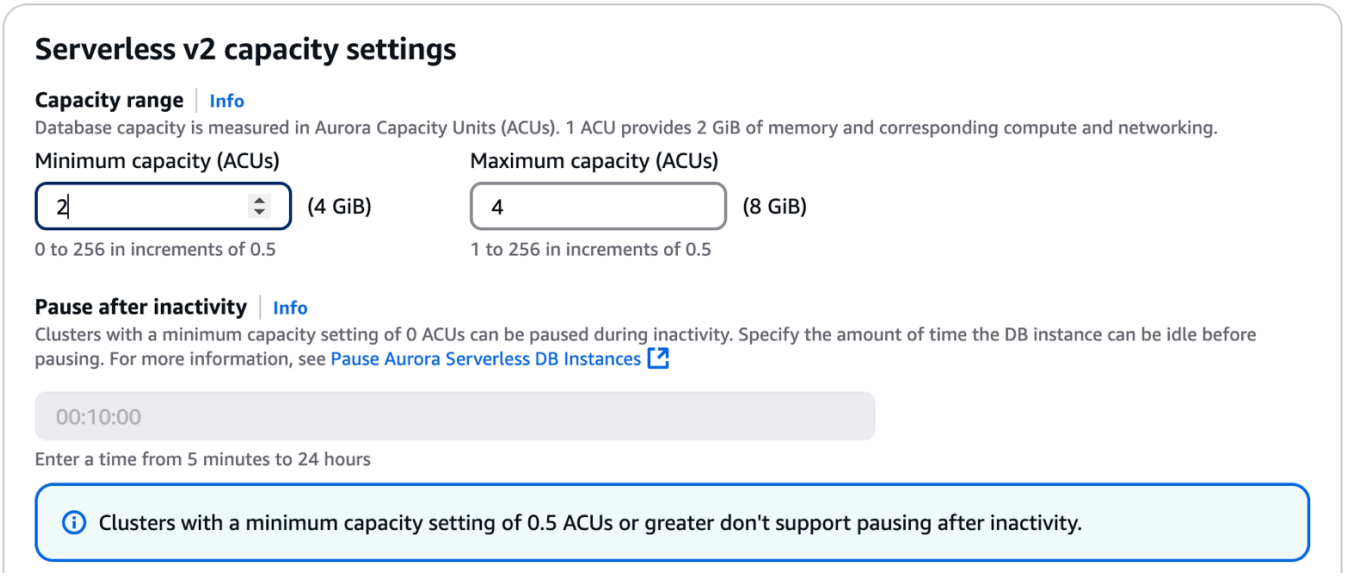
Konsol

Untuk membuat cluster dengan Aurora Serverless v2 penulis

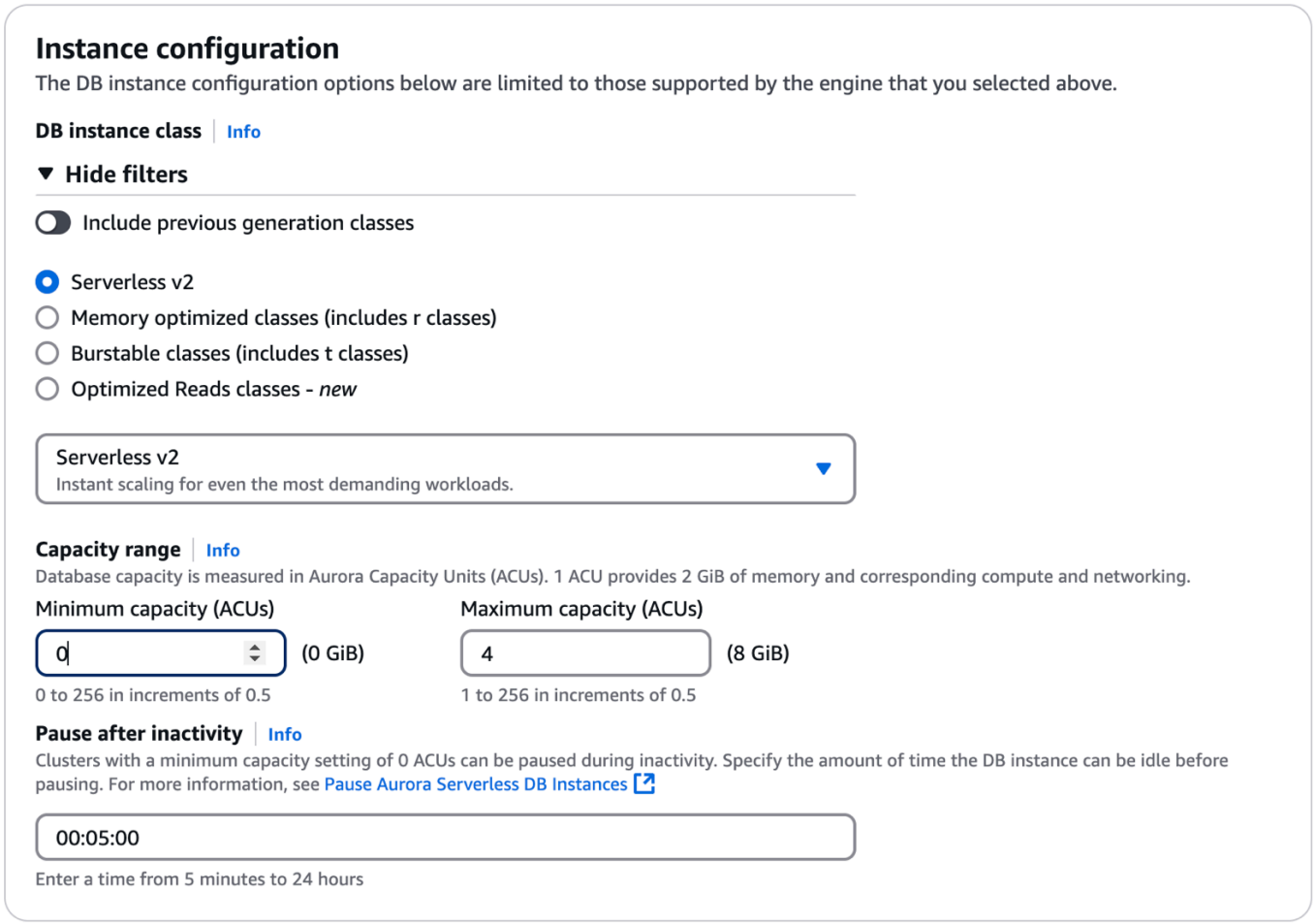
1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih Buat database. Di halaman yang muncul, pilih opsi berikut:
   - Untuk tipe Engine, pilih Aurora (My SQL Compatible) atau Aurora (Postgre Compatible). SQL
   - Untuk Versi, pilih salah satu versi yang didukung untukDaerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2.
4. Untuk kelas instans DB, pilih Serverless v2.
5. Untuk rentang Kapasitas, Anda dapat menerima rentang default. Atau, Anda dapat memilih nilai lain untuk unit kapasitas minimum dan maksimum. Anda dapat memilih dari 0 ACUs minimum hingga 256 ACUs maksimum, dengan penambahan 0,5ACU.

   Untuk informasi lebih lanjut tentang Aurora Serverless v2 unit kapasitas, lihat Aurora Serverless v2 kapasitas danPerforma dan penskalaan untuk Aurora Serverless v2.

   Tergantung pada mesin dan versi yang Anda pilih, batas atas mungkin 128ACUs, batas bawah mungkin 0,5ACUs, atau keduanya. Untuk detail tentang batas untuk setiap kombinasi mesin dan versi Aurora, lihat. Aurora Serverless v2 kapasitas




Memilih kapasitas minimum 0 ACUs memungkinkan Aurora Serverless v2 jeda otomatis dan kemampuan melanjutkan. Dalam hal ini, Anda dapat membuat pilihan tambahan berapa lama Aurora Serverless v2 Instans DB menunggu tanpa koneksi database sebelum berhenti secara otomatis. Untuk informasi tentang kemampuan jeda otomatis dan melanjutkan, lihatPenskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2.




6. Pilih pengaturan cluster DB lainnya, seperti yang dijelaskan dalamPengaturan untuk klaster Aurora DB.
7. Pilih Buat database untuk membuat cluster Aurora DB Anda dengan Aurora Serverless v2 Instans DB sebagai instance penulis, juga dikenal sebagai instance DB primer.

CLI

Untuk membuat cluster DB yang kompatibel dengan Aurora Serverless v2 Instans DB menggunakan AWS CLI, Anda mengikuti CLI prosedur diMembuat klaster DB Amazon Aurora. Sertakan parameter berikut dalam perintah `create-db-cluster` Anda:

- --wilayah *AWS_Region_where_Aurora Serverless v2_instances_are_available*
- --versi mesin *serverless_v2_compatible_engine_version*
- MinCapacity--serverless-v2-scaling-konfigurasi =, = *minimum_capacity* MaxCapacity *maximum_capacity*

Contoh berikut menunjukkan penciptaan sebuah Aurora Serverless v2 klaster DB.

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> Saat Anda membuat Aurora Serverless v2 Cluster DB menggunakan AWS CLI, mode mesin muncul di output sebagai `provisioned` bukan`serverless`. Mode `serverless` mesin mengacu pada Aurora Serverless v1.

Contoh ini menentukan `--manage-master-user-password` opsi untuk menghasilkan kata sandi administratif dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

Untuk informasi tentang Aurora Serverless v2 persyaratan versi, lihatPersyaratan dan batasan untuk Aurora Serverless v2. Untuk informasi tentang angka yang diizinkan untuk rentang kapasitas dan apa yang direpresentasikan oleh angka-angka tersebut, lihat Aurora Serverless v2 kapasitas dan Performa dan penskalaan untuk Aurora Serverless v2.

Untuk memverifikasi apakah klaster yang ada memiliki pengaturan kapasitas yang ditentukan, periksa output dari perintah `describe-db-clusters` untuk atribut `ServerlessV2ScalingConfiguration`. Atribut tersebut terlihat seperti yang berikut ini.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> Jika Anda tidak menentukan minimum dan maksimum ACUs saat membuat cluster, Anda dapat menggunakan `modify-db-cluster` perintah sesudahnya untuk menambahkan pengaturan itu. Sampai Anda melakukannya, Anda tidak dapat menambahkan Aurora Serverless v2 Instans DB ke cluster. Jika Anda mencoba menambahkan instans DB `db.serverless`, Anda mendapatkan pesan kesalahan.

API

Untuk membuat cluster DB yang kompatibel dengan Aurora Serverless v2 Instans DB menggunakan RDSAPI, Anda mengikuti API prosedur diMembuat klaster DB Amazon Aurora. Pilih pengaturan berikut: Pastikan bahwa operasi `CreateDBCluster` Anda mencakup parameter berikut:

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

Untuk informasi tentang Aurora Serverless v2 persyaratan versi, lihatPersyaratan dan batasan untuk Aurora Serverless v2. Untuk informasi tentang angka yang diizinkan untuk rentang kapasitas dan apa yang direpresentasikan oleh angka-angka tersebut, lihat Aurora Serverless v2 kapasitas dan Performa dan penskalaan untuk Aurora Serverless v2.

Untuk memeriksa apakah klaster yang ada memiliki pengaturan kapasitas yang ditentukan, periksa output dari operasi `DescribeDBClusters` untuk atribut `ServerlessV2ScalingConfiguration`. Atribut tersebut terlihat seperti yang berikut ini.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> Jika Anda tidak menentukan minimum dan maksimum ACUs saat membuat cluster, Anda dapat menggunakan `ModifyDBCluster` operasi sesudahnya untuk menambahkan pengaturan itu. Sampai Anda melakukannya, Anda tidak dapat menambahkan Aurora

Serverless v2 Instans DB ke cluster. Jika Anda mencoba menambahkan instans DB `db.serverless`, Anda mendapatkan pesan kesalahan.

## Membuat Aurora Serverless v2 penulis contoh DB

Meskipun Anda menentukan Aurora Serverless v2 rentang kapasitas saat Anda membuat cluster Aurora, Anda dapat memilih apakah akan menggunakan Aurora Serverless v2 atau disediakan untuk setiap instans DB di cluster. Untuk mulai menggunakan Aurora Serverless v2 segera di cluster DB Anda, tambahkan instance DB penulis yang menggunakan kelas `db.serverless` instance. Di konsol, Anda biasanya membuat pilihan ini sebagai bagian dari alur kerja untuk membuat cluster DB. Oleh karena itu, prosedur ini berlaku sebagian besar ketika Anda melakukan pengaturan melaluiCLI.

Konsol

Saat Anda membuat cluster DB menggunakan AWS Management Console, Anda menentukan properti instance DB penulis pada saat yang bersamaan. Untuk membuat instance DB penulis menggunakan Aurora Serverless v2, pilih kelas instans DB Tanpa Server.

Kemudian Anda mengatur rentang kapasitas untuk cluster dengan menentukan nilai unit kapasitas Aurora minimum dan maksimum ACU (). Nilai minimum dan maksimum ini berlaku untuk masing-masing Aurora Serverless v2 Instans DB di cluster. Untuk prosedur itu dan pentingnya pengaturan kapasitas minimum dan maksimum, lihatMembuat Aurora Serverless v2 klaster DB..

Jika Anda tidak membuat Aurora Serverless v2 Instans DB saat pertama kali membuat cluster, Anda dapat menambahkan satu atau lebih Aurora Serverless v2 Instans DB nanti. Untuk melakukannya, ikuti prosedur dalam Menambahkan Aurora Serverless v2 pembaca dan Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2. Anda menentukan rentang kapasitas pada saat Anda menambahkan yang pertama Aurora Serverless v2 Instans DB ke cluster. Anda dapat mengubah rentang kapasitas nanti dengan mengikuti prosedur dalam Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

CLI

Saat Anda membuat Aurora Serverless v2 Cluster DB menggunakan AWS CLI, Anda secara eksplisit menambahkan instance DB penulis menggunakan perintah. create-db-instance Sertakan parameter berikut:

- `--db-instance-class db.serverless`

Contoh berikut menunjukkan penciptaan sebuah Aurora Serverless v2 penulis contoh DB.

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

# Mengelola Aurora Serverless v2 Klaster DB

Dengan Aurora Serverless v2, cluster Anda dapat dipertukarkan dengan cluster yang disediakan. Bagian Aurora Serverless v2 properti berlaku untuk satu atau lebih instance DB dalam cluster DB. Dengan demikian, prosedur untuk membuat klaster, memodifikasi klaster, membuat dan memulihkan snapshot, dan sebagainya, pada dasarnya sama dengan jenis klaster Aurora lainnya. Untuk prosedur umum untuk mengelola klaster Aurora dan instans DB, lihat Mengelola klaster DB Amazon Aurora.

Dalam topik berikut, Anda dapat mempelajari tentang pertimbangan manajemen untuk cluster yang berisi Aurora Serverless v2 Instans DB.

Topik

- Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster
- Memeriksa rentang kapasitas untuk Aurora Serverless v2
- Menambahkan Aurora Serverless v2 pembaca
- Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2
- Mengonversi Aurora Serverless v2 penulis atau pembaca untuk disediakan
- Menjeda Aurora Serverless v2 Instans DB
- Memilih tingkat promosi untuk Aurora Serverless v2 pembaca
- Menggunakan TLS/SSL dengan Aurora Serverless v2
- Melihat Aurora Serverless v2 penulis dan pembaca
- Penebangan untuk Aurora Serverless v2

## Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster

Untuk memodifikasi parameter konfigurasi atau pengaturan lain untuk cluster yang berisi Aurora Serverless v2 Instans DB, atau instans DB itu sendiri, mengikuti prosedur umum yang sama seperti untuk cluster yang disediakan. Untuk detailnya, lihat Memodifikasi klaster DB Amazon Aurora.

Pengaturan paling penting yang unik untuk Aurora Serverless v2 adalah rentang kapasitas. Setelah Anda menetapkan nilai unit kapasitas Aurora minimum dan maksimum (ACU) untuk cluster Aurora, Anda tidak perlu secara aktif menyesuaikan kapasitas Aurora Serverless v2 Instans DB di cluster. Aurora akan melakukannya untuk Anda. Pengaturan ini dikelola pada tingkat klaster. Nilai ACU minimum dan maksimum yang sama berlaku untuk masing-masing Aurora Serverless v2 Instans DB di cluster.

Anda dapat mengatur nilai spesifik berikut:

- Minimum ACUs — Aurora Serverless v2 Instans DB dapat mengurangi kapasitas hingga jumlah ini ACUs.
- Maksimum ACUs — Aurora Serverless v2 Instans DB dapat meningkatkan kapasitas hingga jumlah ini ACUs.

Versi mesin DB yang lebih baru memungkinkan kapasitas maksimum 256 ACUs, kapasitas minimum 0 ACUs, atau keduanya. Untuk rentang kapasitas untuk berbagai versi mesin DB, lihatAurora Serverless v2 kapasitas.

Untuk kemampuan jeda otomatis dan lanjutkan yang diaktifkan dengan menyetel kapasitas minimum ke 0 ACUs, lihat. Penskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2

Untuk informasi lebih lanjut tentang memastikan bahwa Aurora Serverless v2 Cluster DB dapat menskalakan hingga 256 ACUs, lihatUpgrade Anda Aurora Serverless v2 Cluster DB untuk memungkinkan penskalaan hingga 256 ACUs.

> **Note**
>
> Ketika Anda memodifikasi rentang kapasitas untuk Aurora Serverless v2 Cluster DB, perubahan terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama jendela pemeliharaan terjadwal berikutnya.

Masuk Aurora Serverless v2, Anda tidak dapat langsung mengatur kapasitas saat ini tanpa memodifikasi rentang kapasitas, karena kapasitas menyesuaikan secara dinamis berdasarkan beban kerja dalam rentang yang ditentukan. Namun, Anda dapat memengaruhi kapasitas secara tidak langsung dengan cara-cara berikut:

- Sesuaikan kapasitas minimum — Turunkan sementara kapasitas minimum untuk mengurangi kapasitas dasar saat beban kerja ringan.

- Jeda penskalaan sementara — Untuk memperbaiki kapasitas pada nilai tertentu, atur kapasitas minimum dan maksimum ke tingkat yang sama.
- Skala proaktif untuk penggunaan yang meledak — Jika Anda mengantisipasi beban kerja yang meledak, tingkatkan kapasitas minimum sebelumnya untuk mempertahankan baseline yang lebih tinggi selama periode aktivitas tinggi.

Untuk mengetahui detail tentang efek rentang kapasitas dan cara memantau dan menyesuaikannya, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2 dan Performa dan penskalaan untuk Aurora Serverless v2. Tujuan Anda adalah memastikan bahwa kapasitas maksimum klaster cukup tinggi untuk menangani lonjakan beban kerja, dan kapasitas minimumnya cukup rendah untuk meminimalkan biaya saat klaster tidak sibuk.

Misalnya, berdasarkan pemantauan, Anda menentukan bahwa rentang ACU untuk klaster harus lebih tinggi, lebih rendah, lebih luas, atau lebih sempit. Anda dapat mengatur kapasitas cluster Aurora ke rentang tertentu ACUs dengan, API AWS Management Console AWS CLI, atau Amazon RDS. Rentang kapasitas ini berlaku untuk setiap Aurora Serverless v2 Instans DB di cluster.

Misalnya, cluster Anda memiliki rentang kapasitas 1-16 ACUs dan berisi dua Aurora Serverless v2 Instans DB. Kemudian cluster secara keseluruhan mengkonsumsi antara 2 ACUs (saat idle) dan 32 ACUs (bila digunakan sepenuhnya). Jika Anda mengubah rentang kapasitas dari 8 menjadi 40,5 ACUs, sekarang seluruh cluster mengkonsumsi 16 ACUs saat idle, dan hingga 81 ACUs saat digunakan sepenuhnya.

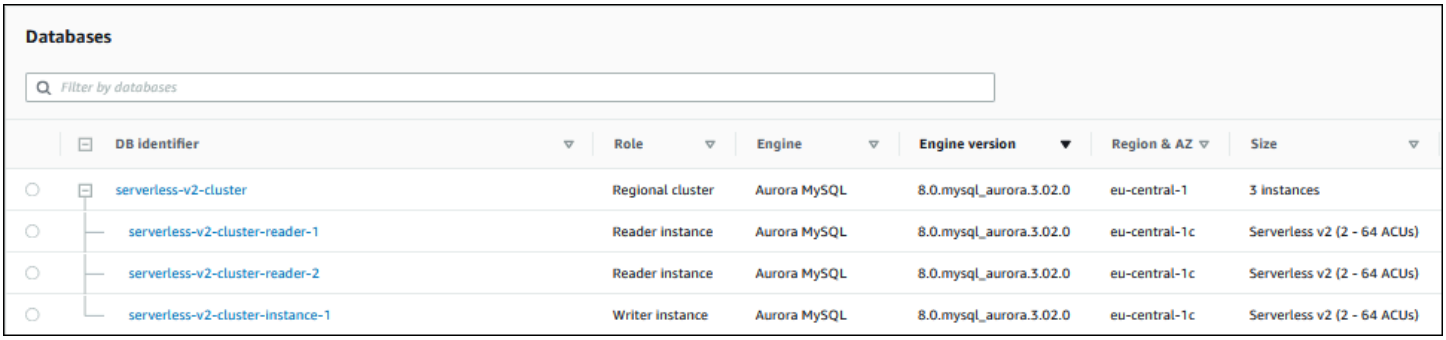
Aurora secara otomatis menetapkan parameter tertentu untuk Aurora Serverless v2 Instans DB ke nilai yang bergantung pada nilai ACU maksimum dalam rentang kapasitas. Untuk mengetahui daftar parameter tersebut, lihat Koneksi maksimum untuk Aurora Serverless v2. Untuk parameter statis yang bergantung pada jenis perhitungan ini, nilainya dievaluasi lagi saat Anda mem-boot ulang instans DB. Dengan demikian, Anda dapat memperbarui nilai parameter tersebut dengan mem-boot ulang instans DB setelah mengubah rentang kapasitas. Untuk memeriksa apakah Anda perlu mem-boot ulang instans DB Anda untuk menerapkan perubahan parameter tersebut, lihat atribut `ParameterApplyStatus` instans DB. Nilai `pending-reboot` menunjukkan bahwa boot ulang akan menerapkan perubahan pada beberapa nilai parameter.

Konsol

Anda dapat mengatur rentang kapasitas cluster yang berisi Aurora Serverless v2 Instans DB dengan. AWS Management Console

Saat Anda menggunakan konsol, Anda mengatur rentang kapasitas untuk cluster pada saat Anda menambahkan yang pertama Aurora Serverless v2 Instans DB ke cluster itu. Anda dapat melakukannya saat Anda memilih kelas instans DB Nirserver v2 untuk instans DB penulis saat Anda membuat klaster. Atau Anda mungkin melakukannya ketika Anda memilih kelas instans DB Tanpa Server saat Anda menambahkan Aurora Serverless v2 pembaca instance DB ke cluster. Anda juga dapat melakukannya saat mengonversi instans DB yang sudah ada di klaster ke kelas instans DB Nirserver. Untuk versi lengkap prosedur tersebut, lihat Membuat Aurora Serverless v2 penulis contoh DB, Menambahkan Aurora Serverless v2 pembaca, dan Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2.

Apapun rentang kapasitas yang Anda tetapkan di tingkat cluster berlaku untuk semua Aurora Serverless v2 Instans DB di cluster Anda. Gambar berikut menunjukkan sebuah cluster dengan beberapa Aurora Serverless v2 contoh DB pembaca. Masing-masing memiliki rentang kapasitas yang identik 2-64 ACUs.

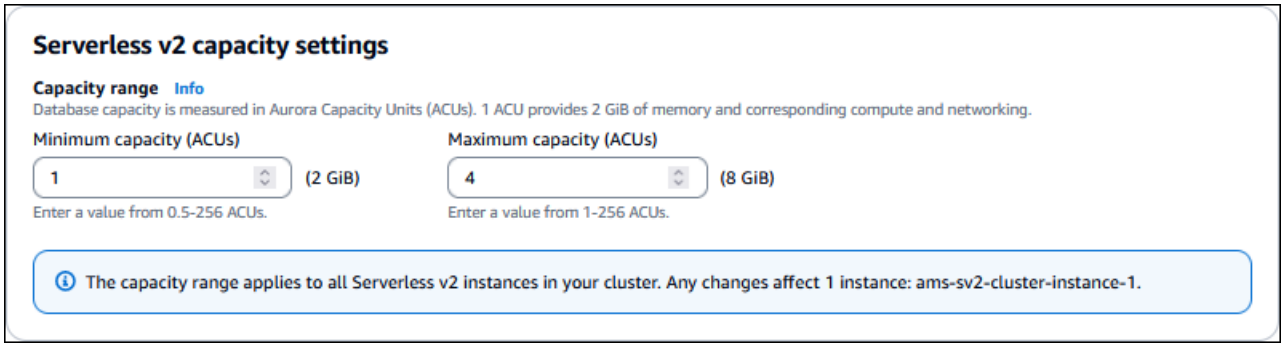



Untuk memodifikasi rentang kapasitas suatu Aurora Serverless v2 cluster

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster yang berisi Aurora Serverless v2 Instans DB dari daftar. Cluster harus sudah berisi setidaknya satu Aurora Serverless v2 contoh DB. Jika tidak, Aurora tidak menampilkan bagian Rentang kapasitas.
4. Untuk Tindakan, pilih Modifikasi.
5. Di bagian Rentang kapasitas, pilih hal berikut:

    a. Masukkan nilai untuk Minimum ACUs. Konsol menampilkan rentang nilai yang diizinkan. Anda dapat memilih kapasitas minimum dari 0 hingga 256 ACUs. Anda dapat memilih kapasitas maksimum dari 1 hingga 256 ACUs. Anda dapat menyesuaikan nilai kapasitas dengan penambahan 0,5 ACU.

b. Masukkan nilai untuk Maksimum ACUs. Nilai ini harus lebih besar dari atau sama dengan Minimum ACUs. Konsol menampilkan rentang nilai yang diizinkan. Gambar berikut menunjukkan pilihan tersebut.




6. Pilih Lanjutkan. Halaman Ringkasan modifikasi akan muncul.
7. Pilih Terapkan segera.

   Perubahan kapasitas terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama periode pemeliharaan terjadwal berikutnya.
8. Pilih Ubah klaster untuk menerima ringkasan modifikasi tersebut. Anda juga dapat memilih Kembali untuk memodifikasi perubahan atau Batalkan untuk menghapus perubahan Anda.

AWS CLI

Untuk mengatur kapasitas cluster tempat Anda ingin menggunakan Aurora Serverless v2 Instans DB menggunakan AWS CLI, jalankan modify-db-cluster AWS CLI perintah. Tentukan opsi `--serverless-v2-scaling-configuration`. Cluster mungkin sudah berisi satu atau lebih Aurora Serverless v2 Instans DB, atau Anda dapat menambahkan instans DB nanti. Nilai valid untuk bidang `MinCapacity` dan `MaxCapacity` mencakup hal berikut:

- `0`, `0.5`1,`1.5`,2,,, dan sebagainya, dalam langkah 0,5, hingga maksimum 256. Nilai ACU minimum 0 hanya tersedia untuk versi Aurora yang mendukung fitur jeda otomatis.

Dalam contoh ini, Anda mengatur rentang ACU dari klaster DB Aurora bernama `sample-cluster` hingga minimum `48.5` dan maksimum 64.

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
  --serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

Perubahan kapasitas terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama periode pemeliharaan terjadwal berikutnya.

Setelah melakukannya, Anda dapat menambahkan Aurora Serverless v2 Instans DB ke cluster dan setiap instans DB baru dapat menskalakan antara 48,5 dan 64. ACUs Rentang kapasitas baru juga berlaku untuk semua Aurora Serverless v2 Instans DB yang sudah ada di cluster. Instans DB menaikkan dan menurunkan skala jika perlu agar berada dalam rentang kapasitas baru.

Untuk contoh tambahan tentang pengaturan rentang kapasitas menggunakan CLI, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora.

Untuk mengubah konfigurasi penskalaan Aurora Serverless Cluster DB menggunakan AWS CLI, jalankan modify-db-cluster AWS CLI perintah. Tentukan opsi `--serverless-v2-scaling-configuration` untuk mengonfigurasi kapasitas minimum dan kapasitas maksimum. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL:`0`,,`0.5`,`1`, `1.52`, dan sebagainya, dengan penambahan 0,5 hingga maksimum. ACUs `256`
- Aurora PostgreSQL:`0`,,,,`0.5`, `11.5`, dan sebagainya2, dengan penambahan 0,5 hingga maksimum. ACUs `256`
- Nilai ACU minimum 0 hanya tersedia untuk versi Aurora yang mendukung fitur jeda otomatis.

Dalam contoh berikut, Anda memodifikasi konfigurasi penskalaan Aurora Serverless v2 Instans DB bernama `sample-instance` itu bagian dari cluster bernama`sample-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

Untuk Windows:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

API RDS

Anda dapat mengatur kapasitas instans Aurora DB dengan operasi Modify DBCluster API. Tentukan parameter `ServerlessV2ScalingConfiguration`. Nilai valid untuk bidang `MinCapacity` dan `MaxCapacity` mencakup hal berikut:

- `0`, `0.5`,`1.5`,2,,, dan sebagainya, dalam langkah 0,5, hingga maksimum 256. Nilai ACU minimum 0 hanya tersedia untuk versi Aurora yang mendukung fitur jeda otomatis.

Anda dapat memodifikasi konfigurasi penskalaan klaster yang berisi Aurora Serverless v2 Instans DB dengan operasi Modify DBCluster API. Tentukan parameter `ServerlessV2ScalingConfiguration` untuk mengonfigurasi kapasitas minimum dan kapasitas maksimum. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL:`0`,,`0.5`,`1`, `1.52`, dan sebagainya, dengan penambahan 0,5 hingga maksimum. ACUs 256
- Aurora PostgreSQL:`0`,,,,`0.5`, `11.5`, dan sebagainya2, dengan penambahan 0,5 hingga maksimum. ACUs 256
- Nilai ACU minimum 0 hanya tersedia untuk versi Aurora yang mendukung fitur jeda otomatis.

Perubahan kapasitas terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama periode pemeliharaan terjadwal berikutnya.

## Upgrade Anda Aurora Serverless v2 Cluster DB untuk memungkinkan penskalaan hingga 256 ACUs

Dalam beberapa kasus, ketika Anda mencoba untuk menskalakan Aurora Serverless v2 DB cluster untuk kapasitas lebih besar dari 128 ACUs, Anda menerima pesan kesalahan. Pesan kesalahan memberi tahu Anda instance mana yang tidak kompatibel dengan rentang penskalaan baru.

Ketidakmampuan untuk skala lebih besar dari 128 ACUs dapat terjadi karena salah satu dari dua alasan:

- Versi mesin DB yang lebih lama - Tingkatkan versi mesin DB ke versi yang mendukung 256 ACUs. Untuk informasi selengkapnya, lihat Aurora Serverless v2 kapasitas.
- Platform lama - Tingkatkan platform yang mendasari untuk Anda Aurora Serverless v2 klaster DB. Anda dapat melakukannya dengan salah satu cara berikut:

- Hentikan dan restart cluster DB. Ini akan mengakhiri platform dasar yang ada dan menyediakan yang baru yang mampu 256 ACUs.

  Namun, menghentikan dan memulai cluster DB menimbulkan beberapa downtime, biasanya beberapa menit. Untuk informasi selengkapnya, lihat Menghentikan dan memulai klaster DB Amazon Aurora.

- Ganti instance yang lebih lama dan gagal ke salah satu instance baru.

  1. Tambahkan instance DB pembaca. Instans pembaca baru akan berjalan pada perangkat keras yang diperbarui yang mampu 256 ACUs. Untuk informasi selengkapnya, lihat Menambahkan Aurora Serverless v2 pembaca.
  2. Gagal ke salah satu contoh pembaca baru. Untuk informasi selengkapnya, lihat Gagal atas kluster DB Amazon Aurora.
  3. Setelah gagal, Anda dapat menghapus instance lama, termasuk instance penulis sebelumnya.

- Gunakan penerapan biru/hijau. Untuk informasi selengkapnya, lihat Ikhtisar Amazon Aurora Penerapan Biru/Hijau.

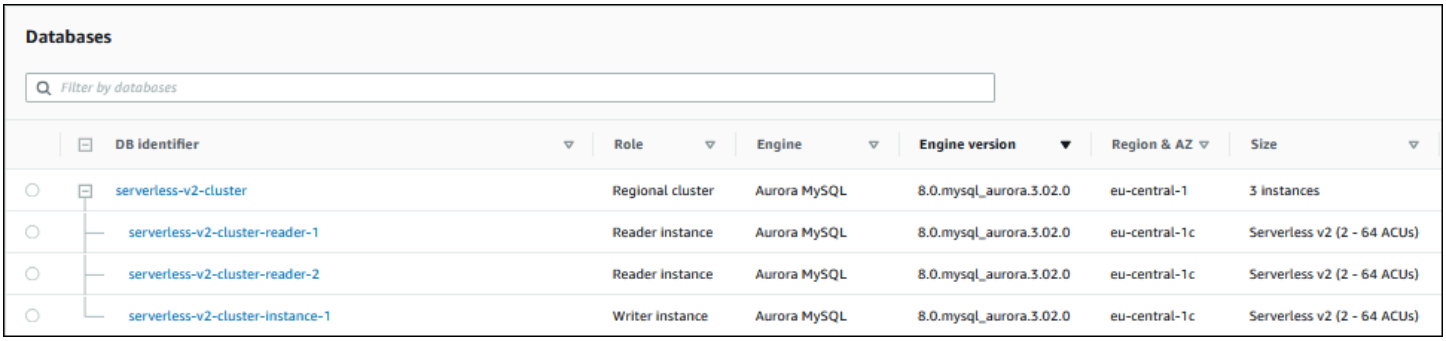
  1. Buat penyebaran biru/hijau. Untuk informasi selengkapnya, lihat Membuat penyebaran biru/hijau di Amazon RDS Aurora.
  2. Konfirmasikan bahwa Anda dapat mengatur kapasitas maksimum untuk lingkungan pementasan (hijau) ke 256 ACUs.
  3. Beralih ke lingkungan hijau. Untuk informasi selengkapnya, lihat Mengalihkan penerapan biru/hijau di Amazon RDS Aurora.
  4. Hapus cluster DB asli.

  > **Note**
  >
  > Jika Anda mempertahankan kredensi database AWS Secrets Manager, Anda tidak dapat menggunakan penerapan biru/hijau.
  > Aurora Global Database tidak mendukung penerapan biru/hijau.

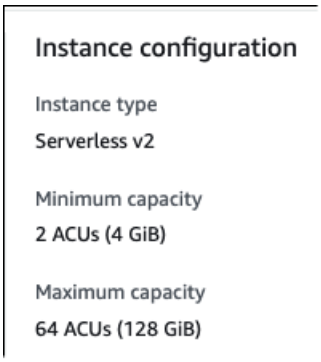
## Memeriksa rentang kapasitas untuk Aurora Serverless v2

Prosedur untuk memeriksa rentang kapasitas untuk Anda Aurora Serverless v2 cluster mengharuskan Anda terlebih dahulu mengatur rentang kapasitas. Jika Anda belum melakukannya, ikuti prosedur dalam Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

Apapun rentang kapasitas yang Anda tetapkan di tingkat cluster berlaku untuk semua Aurora Serverless v2 Instans DB di cluster Anda. Gambar berikut menunjukkan sebuah cluster dengan beberapa Aurora Serverless v2 Instans DB. Masing-masing memiliki rentang kapasitas yang identik.




Anda juga dapat melihat halaman detail untuk setiap Aurora Serverless v2 Instans DB di cluster. Rentang kapasitas instans DB muncul di tab Konfigurasi.




Anda juga dapat melihat rentang kapasitas saat ini untuk klaster pada halaman Ubah untuk klaster. Pada tahap ini, Anda dapat mengubah rentang kapasitas. Untuk semua cara yang dapat Anda gunakan untuk mengatur atau mengubah rentang kapasitas, lihat Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

## Memeriksa rentang kapasitas saat ini untuk klaster Aurora

Anda dapat memeriksa rentang kapasitas yang dikonfigurasi Aurora Serverless v2 Instans DB di cluster dengan memeriksa `ServerlessV2ScalingConfiguration` atribut untuk cluster. AWS CLI Contoh berikut menunjukkan cluster dengan kapasitas minimum 0,5 unit kapasitas Aurora (ACUs) dan kapasitas maksimum 16. ACUs

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
```

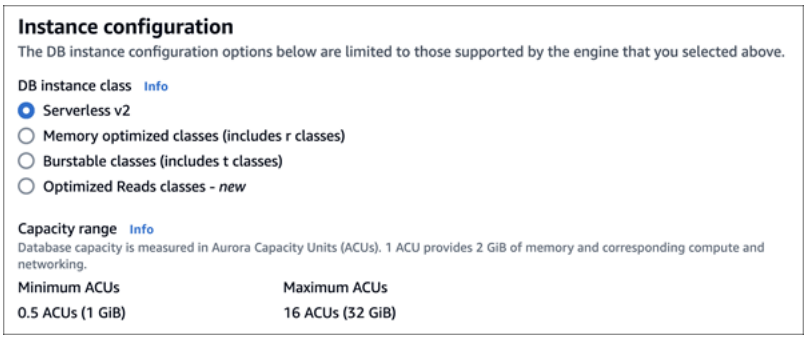
```
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

## Menambahkan Aurora Serverless v2 pembaca

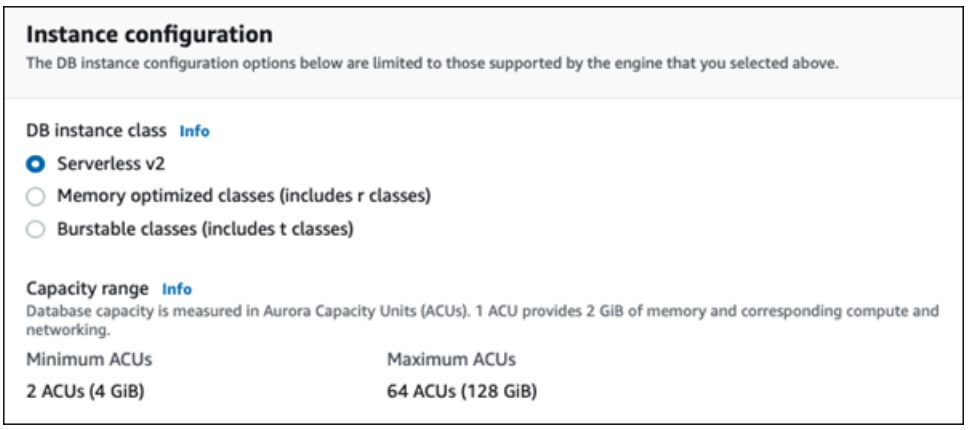
Untuk menambahkan Aurora Serverless v2 pembaca instans DB ke cluster Anda, Anda mengikuti prosedur umum yang sama seperti di[Menambahkan Replika Aurora ke klaster DB](). Pilih kelas instans Nirserver v2 untuk instans DB baru.

Jika instans DB pembaca adalah yang pertama Aurora Serverless v2 Instans DB di cluster, Anda juga memilih rentang kapasitas. Pengaturan ini berlaku untuk instans DB pembaca ini dan lainnya Aurora Serverless v2 Instans DB yang Anda tambahkan ke cluster. Masing-masing Aurora Serverless v2 Instans DB dapat menskalakan antara nilai ACU minimum dan maksimum.

Jika ada yang lain Aurora Serverless v2 contoh sudah ada di cluster, dialog Add reader menunjukkan rentang kapasitas saat ini untuk cluster. Dalam hal ini, Anda tidak dapat mengubah kapasitas. Gambar berikut menunjukkan laporan kapasitas cluster saat ini.




Jika Anda sudah menambahkan Aurora Serverless v2 Instans DB ke cluster, menambahkan yang lain Aurora Serverless v2 instans DB pembaca menunjukkan rentang kapasitas saat ini. Gambar berikut menunjukkan kontrol hanya baca tersebut.




Jika Anda ingin mengubah rentang kapasitas klaster, ikuti prosedur dalam Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

Untuk cluster yang berisi lebih dari satu instans DB pembaca, prioritas failover masing-masing Aurora Serverless v2 instans DB pembaca memainkan peran penting dalam bagaimana instans DB itu naik dan turun. Anda tidak dapat menentukan prioritas saat membuat klaster pada awalnya. Ingat properti ini saat Anda menambahkan pembaca kedua atau instans DB yang lebih baru ke klaster Anda. Untuk informasi selengkapnya, lihat Memilih tingkat promosi untuk Aurora Serverless v2 pembaca.

Untuk cara lain agar Anda dapat melihat rentang kapasitas saat ini untuk sebuah klaster, lihat Memeriksa rentang kapasitas untuk Aurora Serverless v2.

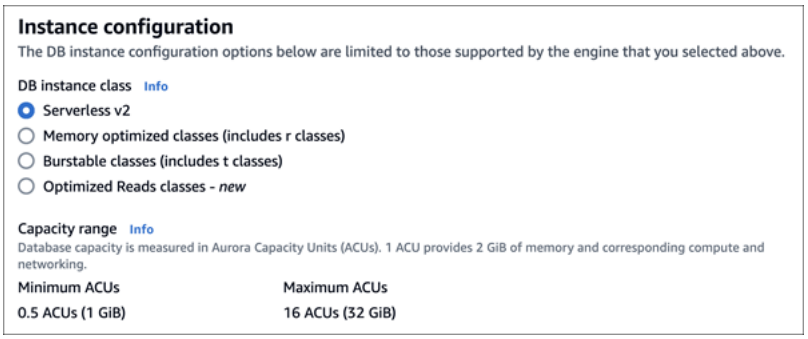
## Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2

Anda dapat mengonversi instans DB yang disediakan untuk digunakan Aurora Serverless v2. Untuk melakukannya, Anda mengikuti prosedur diMemodifikasi instans DB dalam klaster DB. Klaster harus memenuhi persyaratan dalam Persyaratan dan batasan untuk Aurora Serverless v2. Misalnya, Aurora Serverless v2 Instans DB mengharuskan cluster menjalankan versi mesin minimum tertentu.

Misalkan Anda mengonversi cluster yang disediakan yang sedang berjalan untuk memanfaatkan Aurora Serverless v2. Dalam hal ini, Anda dapat meminimalkan waktu henti dengan mengonversi instans DB ke Aurora Serverless v2 sebagai langkah pertama dalam proses peralihan. Untuk prosedur lengkapnya, lihat Beralih dari klaster yang disediakan ke Aurora Serverless v2.

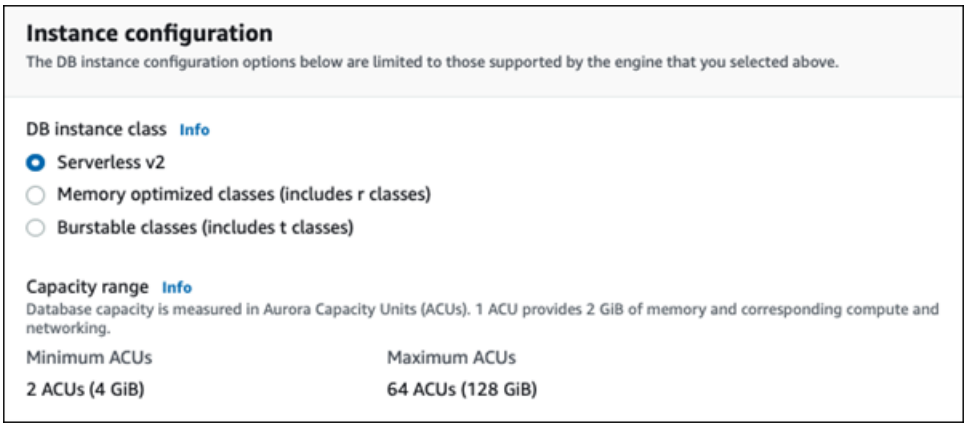
Jika instans DB yang Anda konversi adalah yang pertama Aurora Serverless v2 Instans DB di cluster, Anda memilih rentang kapasitas untuk cluster sebagai bagian dari operasi Modify. Rentang kapasitas ini berlaku untuk masing-masing Aurora Serverless v2 Instans DB yang Anda tambahkan ke cluster.

Gambar berikut menunjukkan halaman tempat Anda menentukan unit kapasitas Aurora minimum dan maksimum ()ACUs.




Untuk detail tentang pentingnya rentang kapasitas, lihat Aurora Serverless v2 kapasitas.

Jika cluster sudah berisi satu atau lebih Aurora Serverless v2 Instans DB, Anda melihat rentang kapasitas yang ada selama operasi Modify. Gambar berikut menunjukkan contoh panel informasi tersebut.




Jika Anda ingin mengubah rentang kapasitas untuk cluster setelah Anda menambahkan lebih banyak Aurora Serverless v2 Instans DB, ikuti prosedur diMengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

## Mengonversi Aurora Serverless v2 penulis atau pembaca untuk disediakan

Anda dapat mengonversi Aurora Serverless v2 Instans DB ke instance DB yang disediakan. Untuk melakukannya, ikuti prosedur dalam Memodifikasi instans DB dalam klaster DB. Pilih kelas instans DB selain Nirserver.

Anda dapat mengonversi Aurora Serverless v2 Instans DB untuk disediakan jika membutuhkan kapasitas yang lebih besar daripada yang tersedia dengan unit kapasitas Aurora maksimum () dari ACUs Aurora Serverless v2 contoh DB. Misalnya, kelas instans db.r5 dan db.r6g DB terbesar

memiliki kapasitas memori yang lebih besar daripada Aurora Serverless v2 Instans DB dapat menskalakan ke.

> **Tip**
>
> Beberapa kelas instans DB yang lebih lama seperti db.r3 dan db.t2 tidak tersedia untuk versi Aurora yang Anda gunakan Aurora Serverless v2. Untuk melihat kelas instans DB mana yang dapat Anda gunakan saat mengonversi Aurora Serverless v2 Instans DB ke yang disediakan, lihat. Mesin DB yang didukung untuk kelas instans DB

Jika Anda mengonversi instance DB penulis cluster Anda dari Aurora Serverless v2 untuk disediakan, Anda dapat mengikuti prosedur Beralih dari klaster yang disediakan ke Aurora Serverless v2 tetapi secara terbalik. Ganti salah satu instance DB pembaca di cluster dari Aurora Serverless v2 untuk disediakan. Kemudian lakukan failover untuk membuat instans DB terprovisi tersebut menjadi penulis.

Rentang kapasitas apa pun yang sebelumnya Anda tentukan untuk klaster tetap ada, meskipun semuanya Aurora Serverless v2 Instans DB dihapus dari cluster. Jika Anda ingin mengubah rentang kapasitas, Anda dapat memodifikasi klaster, seperti yang dijelaskan dalam Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

## Menjeda Aurora Serverless v2 Instans DB

Anda dapat mengonfigurasi klaster Aurora untuk menjeda dan melanjutkannya secara otomatis Aurora Serverless v2 Instans DB. Untuk informasi selengkapnya, lihat Penskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2.

## Memilih tingkat promosi untuk Aurora Serverless v2 pembaca

Untuk cluster yang berisi beberapa Aurora Serverless v2 Instans DB atau campuran yang disediakan dan Aurora Serverless v2 Instans DB, perhatikan pengaturan tingkat promosi untuk masing-masing Aurora Serverless v2 contoh DB. Pengaturan ini mengontrol lebih banyak perilaku untuk Aurora Serverless v2 Instans DB daripada untuk instans DB yang disediakan.

Dalam AWS Management Console, Anda menentukan pengaturan ini menggunakan pilihan prioritas Failover di bawah Konfigurasi tambahan untuk Buat database, Modify instance, dan Add reader pages. Anda melihat properti ini untuk instans DB yang ada di kolom Tingkat prioritas opsional pada

halaman Basis data. Anda juga dapat melihat properti ini di halaman detail untuk klaster DB atau instans DB.

Untuk instans DB terprovisi, pilihan tingkat 0–15 hanya menentukan urutan yang digunakan Aurora dalam memilih instans DB pembaca mana yang akan dipromosikan menjadi penulis selama operasi failover. Untuk Aurora Serverless v2 Instance DB pembaca, nomor tier juga menentukan apakah instans DB menskalakan agar sesuai dengan kapasitas instans DB penulis atau skala secara independen berdasarkan beban kerjanya sendiri. Aurora Serverless v2 Instans DB pembaca di tingkat 0 atau 1 disimpan pada kapasitas minimum setidaknya setinggi instans DB penulis. Dengan begitu, instans DB pembaca tersebut akan siap mengambil alih dari instans DB penulis jika terjadi failover. Jika instans DB penulis adalah instans DB yang disediakan, Aurora memperkirakan yang setara Aurora Serverless v2 kapasitas. Ini menggunakan perkiraan itu sebagai kapasitas minimum untuk Aurora Serverless v2 pembaca contoh DB.

Aurora Serverless v2 instans DB pembaca di tingkatan 2-15 tidak memiliki kendala yang sama pada kapasitas minimumnya. Saat idle, instans DB ini dapat diturunkan skalanya ke nilai unit kapasitas Aurora (ACU) minimum yang ditentukan dalam rentang kapasitas klaster.

AWS CLI Contoh Linux berikut menunjukkan cara memeriksa tingkatan promosi dari semua instans DB di cluster Anda. Bidang akhir menyertakan nilai `True` untuk instans DB penulis dan `False` untuk semua instans DB pembaca.

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

AWS CLI Contoh Linux berikut menunjukkan cara mengubah tingkat promosi instans DB tertentu di cluster Anda. Perintah ini pertama-tama memodifikasi instans DB dengan tingkat promosi baru. Kemudian perintah ini akan menunggu instans DB tersedia lagi dan mengonfirmasi tingkat promosi baru untuk instans DB.

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
```

```
  --query '*[].[PromotionTier]' --output text
0
```

Untuk panduan selengkapnya tentang menentukan tingkat promosi untuk kasus penggunaan yang berbeda-beda, lihat Aurora Serverless v2 penskalaan.

## Menggunakan TLS/SSL dengan Aurora Serverless v2

Aurora Serverless v2 dapat menggunakan protokol Transport LayerSecurity/Secure Sockets Layer (TLS/SSL) untuk mengenkripsi komunikasi antara klien dan Aurora Serverless v2 Instans DB. Ini mendukung TLS/SSL versions 1.0, 1.1, and 1.2. For general information about using TLS/SSL dengan Aurora, lihat. Koneksi TLS ke cluster DB MySQL Aurora

Untuk mempelajari selengkapnya tentang menghubungkan ke basis data Aurora MySQL dengan klien MySQL, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Aurora Serverless v2 mendukung semua mode TLS/SSL yang tersedia untuk klien MySQL (`mysql`) dan klien PostgreSQL (), termasuk yang tercantum dalam tabel berikut. `psql`

| Deskripsi mode TLS/SSL | mysql | psql |
|---|---|---|
| Hubungkan tanpa menggunakan TLS/SSL. | DISABLED | disable |
| Mencoba koneksi menggunakan TLS/SSL terlebih dahulu, tetapi kembali ke non-SSL jika diperlukan. | PREFERRED | prefer (default) |
| Memberlakukan penggunaan TLS/SSL. | REQUIRED | require |
| Memberlakukan TLS/SSL dan memverifikasi otorisasi sertifikat (CA). | VERIFY_CA | verify-ca |
| Memberlakukan TLS/SSL, memverifikasi CA, dan memverifikasi nama host CA. | VERIFY_IDENTITY | verify-full |

Aurora Serverless v2 menggunakan sertifikat wildcard. Jika Anda menentukan opsi "memverifikasi CA" atau "memverifikasi CA dan nama host CA" saat menggunakan TLS/SSL, pertama-tama unduh trust store Amazon root CA 1 dari Amazon Trust Services. Setelah melakukannya, Anda dapat mengidentifikasi file berformat PEM ini dalam perintah klien Anda. Untuk melakukannya menggunakan PostgreSQL, lakukan hal berikut.

Untuk Linux, macOS, atau Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Untuk mempelajari selengkapnya tentang mengoperasikan basis data Aurora PostgreSQL menggunakan klien Postgres, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data PostgreSQL.

Untuk informasi selengkapnya tentang menghubungkan ke klaster DB Aurora secara umum, lihat Menghubungkan ke klaster DB Amazon Aurora.

### Suite cipher yang didukung untuk koneksi ke Aurora Serverless v2 Klaster DB

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data Anda. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS/SSL klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Tindakan ini mencegah penggunaan cipher yang tidak aman atau yang sudah dihentikan.

Aurora Serverless v2 Cluster DB yang didasarkan pada Aurora MySQL mendukung suite cipher yang sama dengan cluster DB yang disediakan Aurora MySQL. Untuk informasi tentang cipher suite ini, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL.

Aurora Serverless v2 Cluster DB yang didasarkan pada Aurora PostgreSQL mendukung suite cipher yang sama dengan cluster DB yang disediakan Aurora PostgreSQL. Untuk informasi tentang cipher suite ini, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL.

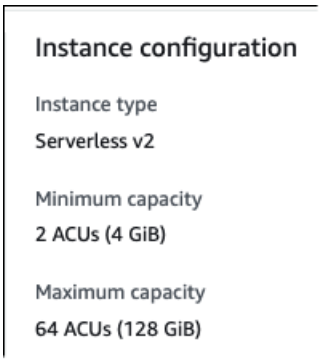
## Melihat Aurora Serverless v2 penulis dan pembaca

Anda dapat melihat detail Aurora Serverless v2 Instans DB dengan cara yang sama seperti yang Anda lakukan untuk instance DB yang disediakan. Untuk melakukannya, ikuti prosedur umum dari Melihat klaster DB Amazon Aurora. Sebuah klaster mungkin berisi semua Aurora Serverless v2 Instans DB, semua instans DB yang disediakan, atau beberapa dari masing-masing.

Setelah Anda membuat satu atau lebih Aurora Serverless v2 Instans DB, Anda dapat melihat instance DB mana yang bertipe Tanpa Server dan tipe Instance. Anda juga dapat melihat unit kapasitas Aurora minimum dan maksimum (ACUs) yang Aurora Serverless v2 Instans DB dapat digunakan. Setiap ACU merupakan kombinasi dari kapasitas pemrosesan (CPU) dan memori (RAM). Rentang kapasitas ini berlaku untuk masing-masing Aurora Serverless v2 Instans DB di cluster. Untuk prosedur untuk memeriksa rentang kapasitas cluster atau apapun Aurora Serverless v2 Instans DB di cluster, lihatMemeriksa rentang kapasitas untuk Aurora Serverless v2.

Di AWS Management Console, Aurora Serverless v2 Instans DB ditandai di bawah kolom Ukuran di halaman Database. Instans DB terprovisi menunjukkan nama kelas instans DB seperti r6g.xlarge. Bagian Aurora Serverless Instans DB menunjukkan Tanpa Server untuk kelas instans DB, bersama dengan kapasitas minimum dan maksimum instans DB. Misalnya, Anda mungkin melihat Tanpa Server v2 (4—64 ACUs) atau Tanpa Server v2 (1-40). ACUs

Anda dapat menemukan informasi yang sama pada tab Konfigurasi untuk masing-masing Aurora Serverless v2 Instans DB di konsol. Misalnya, Anda mungkin melihat bagian Tipe instans seperti berikut ini. Di sini, nilai tipe Instance adalah Serverless v2, nilai kapasitas Minimum adalah 2 ACUs (4 GiB), dan nilai kapasitas maksimum adalah  ACUs 64 (128 GiB).




Anda dapat memantau kapasitas masing-masing Aurora Serverless v2 Instans DB dari waktu ke waktu. Dengan begitu, Anda dapat memeriksa minimum, maksimum, dan rata-rata yang ACUs dikonsumsi oleh setiap instans DB. Anda juga dapat memeriksa seberapa dekat instans DB dengan kapasitas minimum atau maksimumnya. Untuk melihat detail tersebut di AWS Management Console, periksa grafik CloudWatch metrik Amazon pada tab Monitoring untuk instans DB. Untuk informasi tentang metrik yang perlu dilihat dan cara menafsirkannya, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

## Penebangan untuk Aurora Serverless v2

Untuk mengaktifkan pencatatan log basis data, Anda menentukan log yang akan diaktifkan menggunakan parameter konfigurasi di grup parameter kustom Anda.

Untuk Aurora MySQL, Anda dapat mengaktifkan log berikut.

| Aurora MySQL | Deskripsi |
| --- | --- |
| `general_log` | Membuat log umum. Atur ke 1 untuk mengaktif kan. Default-nya adalah nonaktif (0). |
| `log_queries_not_using_indexes` | Mencatat setiap kueri ke log kueri lambat yang tidak menggunakan indeks. Default-nya adalah nonaktif (0). Atur ke 1 untuk mengaktifkan log ini. |
| `long_query_time` | Mencegah kueri yang berjalan cepat agar tidak dicatat dalam log kueri lambat. Dapat diatur ke angka float antara 0 dan 31536000. Default-nya adalah 0 (tidak aktif). |
| `server_audit_events` | Daftar peristiwa untuk dicatat dalam log. Nilai yang didukung adalah `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` , dan `TABLE`. |
| `server_audit_logging` | Atur ke 1 untuk mengaktifkan pencatatan log audit server. Jika Anda mengaktifkan ini, Anda dapat menentukan peristiwa audit yang akan dikirim CloudWatch dengan mencantumkannya di `server_audit_events` parameter. |
| `slow_query_log` | Membuat log kueri lambat. Atur ke 1 untuk mengaktifkan log kueri lambat. Default-nya adalah nonaktif (0). |

Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.

Untuk Aurora PostgreSQL, Anda dapat mengaktifkan log berikut di Aurora Serverless v2 Instans DB.

| Aurora PostgreSQL | Deskripsi |
| --- | --- |
| `log_connections` | Mencatat log setiap koneksi yang berhasil. |
| `log_disconnections` | Mencatat log terkait akhir sebuah sesi, termasuk durasinya. |
| `log_lock_waits` | Default-nya adalah 0 (nonaktif). Atur ke 1 untuk mencatat log peristiwa tunggu kunci. |
| `log_min_duration_statement` | Durasi minimum (dalam milidetik) untuk menjalankan pernyataan sebelum dicatat lognya. |
| `log_min_messages` | Mengatur tingkat pesan yang dicatat lognya. Nilai yang didukung adalah `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`. Untuk mencatat data performa ke log `postgres`, tetapkan nilainya ke `debug1`. |
| `log_temp_files` | Mencatat log penggunaan file sementara yang melampaui kilobyte (kB) yang ditentukan. |
| `log_statement` | Mengontrol pernyataan SQL tertentu yang dicatat lognya. Nilai yang didukung adalah `none`, `ddl`, `mod`, dan `all`. Default-nya adalah `none`. |

Topik

- Logging dengan Amazon CloudWatch
- Melihat Aurora Serverless v2 log di Amazon CloudWatch
- Kapasitas pemantauan dengan Amazon CloudWatch
- Pemantauan Aurora Serverless v2 jeda dan lanjutkan aktivitas

## Logging dengan Amazon CloudWatch

Setelah Anda menggunakan prosedur Penebangan untuk Aurora Serverless v2 untuk memilih log database mana yang akan diaktifkan, Anda dapat memilih log mana yang akan diunggah ("publikasikan") ke Amazon CloudWatch.

Anda dapat menggunakan Amazon CloudWatch untuk menganalisis data log, membuat alarm, dan melihat metrik. Secara default, log kesalahan untuk Aurora Serverless v2 diaktifkan dan diunggah secara otomatis ke CloudWatch. Anda juga dapat mengunggah log lain dari Aurora Serverless v2 Instans DB ke CloudWatch.

Kemudian Anda memilih log mana yang akan diunggah CloudWatch, dengan menggunakan pengaturan ekspor Log di AWS Management Console atau `--enable-cloudwatch-logs-exports` opsi di AWS CLI.

Anda dapat memilih yang mana dari Anda Aurora Serverless v2 log untuk diunggah ke CloudWatch. Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.

Sama seperti jenis apa pun dari klaster DB Aurora, Anda tidak dapat memodifikasi grup parameter klaster DB default. Sebagai gantinya, buat grup parameter klaster DB Anda sendiri berdasarkan parameter default untuk klaster DB dan jenis mesin Anda. Kami menyarankan Anda membuat grup parameter cluster DB kustom Anda sebelum membuat Aurora Serverless v2 Cluster DB, sehingga tersedia untuk dipilih saat Anda membuat database di konsol.

> **Note**
>
> Untuk Aurora Serverless v2, Anda dapat membuat cluster DB dan grup parameter DB. Ini kontras dengan Aurora Serverless v1, di mana Anda hanya dapat membuat grup parameter cluster DB.

## Melihat Aurora Serverless v2 log di Amazon CloudWatch

Setelah Anda menggunakan prosedur dalam Logging dengan Amazon CloudWatch untuk memilih log basis data mana yang akan diaktifkan, Anda dapat melihat konten log.

Untuk informasi lebih lanjut tentang penggunaan CloudWatch dengan log Aurora MySQL dan Aurora PostgreSQL, lihat dan. Memantau peristiwa log di Amazon CloudWatch Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch

Untuk melihat log untuk Aurora Serverless v2 Klaster DB

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Anda Wilayah AWS.
3. Pilih Grup log.
4. Pilih Anda Aurora Serverless v2 Log cluster DB dari daftar. Pola penamaan log adalah sebagai berikut.

```
/aws/rds/cluster/cluster-name/log_type
```

> **Note**
>
> Untuk Aurora MySQL — Kompatibel Aurora Serverless v2 Cluster DB, log kesalahan menyertakan peristiwa penskalaan kumpulan buffer bahkan ketika tidak ada kesalahan.

## Kapasitas pemantauan dengan Amazon CloudWatch

Dengan Aurora Serverless v2, Anda dapat menggunakan CloudWatch untuk memantau kapasitas dan pemanfaatan semua Aurora Serverless v2 Instans DB di cluster Anda. Anda dapat melihat metrik tingkat instans untuk memeriksa kapasitas masing-masing Aurora Serverless v2 Instans DB saat skala naik dan turun. Anda juga dapat membandingkan metrik terkait kapasitas dengan metrik lain untuk melihat bagaimana perubahan beban kerja memengaruhi konsumsi sumber daya. Misalnya, Anda dapat membandingkan `ServerlessDatabaseCapacity` dengan `DatabaseUsedMemory`, `DatabaseConnections`, dan `DMLThroughput` untuk menilai bagaimana klaster DB Anda merespons selama operasi. Untuk detail tentang metrik terkait kapasitas yang berlaku Aurora Serverless v2, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Untuk memantau Aurora Serverless v2 Kapasitas cluster DB

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Metrik. Semua metrik yang tersedia muncul sebagai kartu di konsol, yang dikelompokkan berdasarkan nama layanan.
3. Pilih RDS.

4. (Opsional) Gunakan kotak Pencarian untuk menemukan metrik yang sangat penting Aurora Serverless v2:`ServerlessDatabaseCapacity`,`ACUUtilization`,`CPUUtilization`, dan`FreeableMemory`.

Kami menyarankan Anda mengatur CloudWatch dasbor untuk memantau Aurora Serverless v2 Kapasitas cluster DB menggunakan metrik terkait kapasitas. Untuk mempelajari caranya, lihat Membangun dasbor dengan CloudWatch.

Untuk mempelajari lebih lanjut tentang menggunakan Amazon CloudWatch dengan Amazon Aurora, lihat. Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch

## Pemantauan Aurora Serverless v2 jeda dan lanjutkan aktivitas

Aurora menulis file log terpisah untuk Aurora Serverless v2 Instans DB dengan jeda otomatis diaktifkan. Aurora menulis ke log untuk setiap interval 10 menit bahwa instance tidak dijeda. Aurora mempertahankan hingga tujuh batang kayu ini, diputar setiap hari. File log saat ini diberi nama`instance.log`, dan log lama diberi nama menggunakan pola`instance.`*`YYYY-MM-DD`*`.`*`N`*`.log`.

Log ini diaktifkan secara default untuk Aurora Serverless v2 Instans DB dengan jeda otomatis diaktifkan. Anda dapat melihat isi log ini di AWS Management Console atau dengan menggunakan AWS CLI atau RDSP API. Saat ini, Anda tidak dapat mengunggah log ini ke CloudWatch.

Peristiwa yang tercantum dalam Peristiwa instans DB memberikan ikhtisar tingkat tinggi tentang aktivitas jeda dan melanjutkan, seperti berikut ini:

- Ketika instance mulai berhenti, dan ketika selesai berhenti.
- Ketika instance mulai dilanjutkan, dan ketika selesai dilanjutkan.
- Kasus di mana instance mencoba untuk berhenti sejenak, tetapi beberapa kondisi mencegahnya berhenti.

`instance.log`Ini memberikan detail yang lebih terperinci tentang alasan mengapa Aurora Serverless v2 misalnya mungkin atau mungkin tidak dapat berhenti sejenak.

Log mungkin menunjukkan bahwa instance dilanjutkan karena alasan yang berbeda:

- aktivitas pengguna: Permintaan koneksi database. Ini bisa dari sesi klien interaktif, panggilan API Data RDS, atau permintaan untuk mengunduh log dari instance.

- Aktivitas latar belakang: Kategori luas yang mencakup semua alasan mengapa Aurora melanjutkan sebuah instance. Misalnya, ketika permintaan koneksi ke instance pembaca menyebabkan instance penulis dilanjutkan, log untuk pembaca menunjukkan aktivitas pengguna, sedangkan log untuk penulis mengklasifikasikan permintaan resume itu sebagai aktivitas latar belakang. Untuk alasan mengapa Aurora mungkin melanjutkan instance selain permintaan koneksi pengguna, lihat. Melanjutkan jeda otomatis Aurora Serverless v2 instans

Ketika Aurora tidak mengetahui kondisi apa pun yang akan mencegah instance berhenti ketika interval jeda otomatis berakhir, ia secara berkala menulis pesan informasi ke log. Untuk cluster dengan hanya satu instans DB, log berisi pesan ini:

```
[INFO] No auto-pause blockers registered since time
```

Untuk cluster dengan beberapa instance DB, pesannya sedikit berbeda. Itu karena seorang penulis mungkin tidak dapat berhenti sejenak karena aktivitas pada salah satu contoh pembaca. Jika aktivitas pada pembaca selesai sebelum interval jeda otomatis berakhir pada penulis, penulis dapat berhenti pada waktu yang diharapkan.

```
[INFO] No auto-pause blockers registered since time.
Database might be required to maintain compute capacity for high availability.
```

Jika operasi jeda dimulai, tetapi permintaan koneksi database baru tiba sebelum instance selesai dijeda, log berisi pesan ini:

```
[INFO] Unable to pause database due to a new database activity
```

Jika Aurora mengetahui kondisi apa pun yang pasti mencegah instance berhenti, log berisi pesan ini yang mencantumkan semua kondisi tersebut:

```
[INFO] Auto-pause blockers registered since time: list_of_conditions
```

Dengan begitu, Aurora tidak mencegah Anda mengaktifkan replikasi, integrasi nol-ETL, Aurora Global Database, dan sebagainya dalam kombinasi dengan fitur jeda otomatis. Log memberi tahu Anda kapan penggunaan fitur tersebut dapat mencegah jeda otomatis diterapkan.

Berikut ini adalah alasan mengapa Aurora Serverless v2 instance mungkin melebihi interval batas waktu jeda otomatis, tetapi dicegah agar tidak menjeda:

- aktivitas database sebelum batas waktu jeda otomatis: Instans DB menerima permintaan koneksi sebelum interval batas waktu berakhir.
- anggota database global: Jika cluster DB adalah bagian dari database global Aurora, Aurora Serverless v2 instance di cluster tidak berhenti sejenak. Sebuah cluster dapat berubah dari cluster mandiri ke cluster database global. Dengan demikian, contoh yang sebelumnya dijeda otomatis mungkin berhenti berhenti, dan melaporkan alasan ini di log. Setelah cluster menjadi anggota database global, itu tidak akan kembali ke cluster mandiri sampai Anda secara eksplisit melepaskannya. Cluster primer masih dianggap sebagai bagian dari database global bahkan jika Anda melepaskan semua cluster sekunder.
- kemampuan replikasi dikonfigurasi: Instans DB penulis memiliki replikasi khusus mesin yang diaktifkan, baik replikasi binlog untuk MySQL atau replikasi logis untuk PostgreSQL. Kondisi ini juga dapat disebabkan oleh penggunaan fitur Aurora lain yang memerlukan pengaktifan replikasi, seperti integrasi nol-ETL atau Database Activity Streams (DAS).
- kelambatan pencadangan berkelanjutan: Jika sistem penyimpanan Aurora belum selesai menerapkan perubahan penyimpanan hingga titik waktu saat ini, instance penulis tidak berhenti sampai menyusul. Kondisi ini hanya mempengaruhi contoh penulis, dan diharapkan relatif singkat.
- layanan atau tindakan pemeliharaan pelanggan: Jika operasi pemeliharaan dimulai, instans DB tidak akan berhenti lagi sampai operasi itu selesai. Kondisi ini mencakup berbagai macam operasi yang mungkin dimulai oleh Anda atau oleh Aurora, seperti upgrade, kloning, mengubah pengaturan konfigurasi, upgrade, men-download file log, dan sebagainya. Peristiwa ini juga terjadi ketika Anda meminta untuk menghapus instance, dan Aurora secara singkat melanjutkan instance sebagai bagian dari mekanisme penghapusan.
- Masalah komunikasi sementara: Jika Aurora tidak dapat menentukan apakah konfigurasi penskalaan saat ini memiliki pengaturan kapasitas minimum nol ACUs, itu tidak menghentikan sementara instance. Ini diharapkan menjadi kejadian yang sangat langka.

# Performa dan penskalaan untuk Aurora Serverless v2

Prosedur dan contoh berikut menunjukkan bagaimana Anda dapat mengatur rentang kapasitas untuk klaster Aurora Serverless v2 dan instans DB terkait. Anda juga dapat menggunakan prosedur berikut untuk memantau seberapa sibuk instans DB Anda. Kemudian, Anda dapat menggunakan temuan Anda untuk menentukan apakah Anda perlu menambah atau mengurangi rentang kapasitas.

Sebelum Anda menggunakan prosedur ini, pastikan Anda memahami cara kerja penskalaan Aurora Serverless v2. Mekanisme penskalaannya berbeda dari Aurora Serverless v1. Untuk detailnya, lihat Aurora Serverless v2 penskalaan.

Daftar Isi


- Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora
  - Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster
  - Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster
  - Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL
  - Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL
- Menggunakan grup parameter untuk Aurora Serverless v2
  - Nilai parameter default
  - Koneksi maksimum untuk Aurora Serverless v2
  - Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya
  - Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2
- Menghindari out-of-memory kesalahan
- CloudWatch Metrik Amazon penting untuk Aurora Serverless v2
  - Bagaimana Aurora Serverless v2 metrik berlaku untuk tagihan Anda AWS
  - Contoh CloudWatch perintah untuk Aurora Serverless v2 metrik
- Memantau performa Aurora Serverless v2 dengan Wawasan Performa
- Memecahkan masalah kapasitas Aurora Serverless v2


## Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora

Dengan instans DB Aurora Serverless v2, Anda mengatur rentang kapasitas yang berlaku untuk semua instans DB di klaster DB Anda pada saat yang sama saat Anda menambahkan instans DB Aurora Serverless v2 pertama ke klaster DB. Untuk prosedur dalam melakukannya, lihat Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.

Anda juga dapat mengubah rentang kapasitas untuk klaster yang ada. Bagian berikut membahas secara lebih mendetail tentang cara memilih nilai minimum dan maksimum yang sesuai dan apa yang terjadi ketika Anda membuat perubahan pada rentang kapasitas. Misalnya, mengubah rentang kapasitas dapat memodifikasi nilai default dari beberapa parameter konfigurasi. Untuk menerapkan semua perubahan parameter, setiap instans DB Aurora Serverless v2 mungkin harus di-boot ulang.

Topik


- Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster

- Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster
- Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL
- Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL

## Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster

Pengguna cenderung selalu memilih 0,5 untuk pengaturan kapasitas Aurora Serverless v2 minimum. Nilai itu memungkinkan instans DB untuk menurunkan kapasitas terkecil saat benar-benar menganggur, sambil tetap aktif. Anda juga dapat mengaktifkan perilaku jeda otomatis dengan menentukan kapasitas minimum 0 ACUs, seperti yang dijelaskan dalam. Penskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2 Namun, tergantung pada bagaimana Anda menggunakan cluster itu dan pengaturan lain yang Anda konfigurasikan, kapasitas minimum yang berbeda mungkin yang paling efektif. Pertimbangkan faktor-faktor berikut saat memilih pengaturan kapasitas minimum:

- Tingkat penskalaan untuk instans DB Aurora Serverless v2 tergantung pada kapasitasnya saat ini. Semakin tinggi kapasitas saat ini, semakin cepat instans DB dapat dinaikkan skalanya. Jika Anda memerlukan instans DB menaikkan skalanya dengan cepat ke kapasitas yang sangat tinggi, pertimbangkan untuk mengatur kapasitas minimum ke sebuah nilai sehingga tingkat penskalaannya memenuhi kebutuhan Anda.
- Jika Anda biasanya memodifikasi kelas instans DB Anda untuk mengantisipasi beban kerja yang sangat tinggi atau rendah, Anda dapat menggunakan pengalaman ini untuk membuat perkiraan kasar tentang rentang kapasitas Aurora Serverless v2 yang setara. Untuk menentukan ukuran memori yang akan digunakan pada saat lalu lintas rendah, baca Spesifikasi perangkat keras kelas instans DB untuk Aurora.

  Misalnya, anggaplah Anda menggunakan kelas instans DB db.r6g.xlarge ketika klaster Anda memiliki beban kerja yang rendah. Kelas instans DB tersebut memiliki memori 32 GiB. Dengan demikian, Anda dapat menentukan pengaturan unit kapasitas Aurora minimum (ACU) sebanyak 16 untuk menyiapkan instans DB Aurora Serverless v2 yang dapat menurunkan skalanya hingga mencapai kapasitas yang kira-kira sama. Itu karena setiap ACU setara dengan sekitar 2 GiB memori. Anda dapat menentukan nilai yang agak lebih rendah agar instans DB menurunkan skalanya lebih jauh jika instans DB db.r6g.xlarge Anda terkadang kurang dimanfaatkan.
- Jika aplikasi Anda beroperasi paling efisien ketika instans DB memiliki sejumlah data dalam cache buffer, pertimbangkan untuk menentukan pengaturan ACU minimum sehingga memorinya cukup besar untuk menampung data yang sering diakses. Jika tidak, beberapa data akan dikeluarkan

dari cache buffer ketika instans DB Aurora Serverless v2 menurunkan skalanya ke ukuran memori yang lebih rendah. Kemudian, ketika instans DB menaikkan skalanya kembali, informasi dari data akan dibacakan kembali ke cache buffer dari waktu ke waktu. Jika jumlah I/O untuk membawa data kembali ke cache buffer cukup besar, mungkin lebih efektif untuk memilih nilai ACU minimum yang lebih tinggi.

- Jika instans DB Aurora Serverless v2 Anda lebih sering berjalan pada kapasitas tertentu, pertimbangkan untuk menentukan pengaturan kapasitas minimum yang lebih rendah dari acuan dasar tersebut, tetapi tidak terlalu rendah. Instans DB Aurora Serverless v2 dapat sangat efektif memperkirakan berapa banyak dan seberapa cepat skala harus dinaikkan ketika kapasitas saat ini tidak secara drastis lebih rendah dari kapasitas yang dibutuhkan.
- Jika beban kerja terprovisi Anda memiliki persyaratan memori yang terlalu tinggi untuk kelas instans DB kecil seperti T3 atau T4g, pilih pengaturan ACU minimum yang menyediakan memori yang sebanding dengan instans DB R5 atau R6g.

  Secara khusus, kami merekomendasikan kapasitas minimum berikut untuk digunakan dengan fitur yang ditentukan (rekomendasi ini dapat berubah):

  - Performance Insights - 2 ACUs
  - Database global Aurora — 8 ACUs (hanya berlaku untuk yang utama) Wilayah AWS
- Di Aurora, replikasi terjadi pada lapisan penyimpanan, sehingga kapasitas pembaca tidak secara langsung mempengaruhi replikasi. Namun, untuk instans DB Aurora Serverless v2 pembaca yang menskalakan secara independen, pastikan bahwa kapasitas minimum cukup untuk menangani beban kerja selama periode intensif penulisan untuk menghindari latensi kueri. Jika instans DB pembaca di tingkatan promosi 2—15 mengalami masalah kinerja, pertimbangkan untuk meningkatkan kapasitas minimum klaster. Untuk detail tentang cara memilih apakah instans DB pembaca akan diskalakan bersama dengan penulis atau terpisah, lihat Memilih tingkat promosi untuk Aurora Serverless v2 pembaca.
- Jika Anda memiliki cluster DB dengan instans DB Aurora Serverless v2 pembaca, pembaca tidak menskalakan bersama dengan instans DB penulis ketika tingkat promosi pembaca bukan 0 atau 1. Dalam hal ini, pengaturan kapasitas minimum yang rendah dapat mengakibatkan lag replikasi yang berlebihan. Hal ini karena pembaca mungkin tidak memiliki kapasitas yang cukup untuk menerapkan perubahan dari penulis ketika basis data sibuk. Kami menyarankan Anda mengatur kapasitas minimum ke nilai yang mewakili jumlah memori dan CPU yang sebanding dengan instans DB penulis.
- Nilai `max_connections` parameter untuk instans Aurora Serverless v2 DB didasarkan pada ukuran memori yang berasal dari maksimum ACUs. Namun, ketika Anda menentukan kapasitas

minimum 0 atau 0,5 ACUs pada instans DB yang kompatibel dengan PostgreSQL, nilai maksimum dibatasi pada 2.000. `max_connections`

Jika Anda bermaksud menggunakan klaster Aurora PostgreSQL untuk beban kerja koneksi tinggi, pertimbangkan untuk menggunakan pengaturan ACU minimum 1 atau lebih tinggi. Untuk detail tentang cara Aurora Serverless v2 menangani parameter konfigurasi `max_connections`, lihat Koneksi maksimum untuk Aurora Serverless v2.

- Waktu yang dibutuhkan instans DB Aurora Serverless v2 untuk menskalakan dari kapasitas minimumnya ke kapasitas maksimumnya akan tergantung pada perbedaan antara nilai ACU minimum dan maksimumnya. Ketika kapasitas instans DB saat ini besar, Aurora Serverless v2 akan dinaikkan skalanya dalam inkremen yang lebih besar daripada ketika instans DB dimulai dari kapasitas kecil. Jadi, jika Anda menentukan kapasitas maksimum yang relatif besar dan instans DB menghabiskan sebagian besar waktunya mendekati kapasitas tersebut, pertimbangkan untuk meningkatkan pengaturan ACU minimum. Dengan begitu, instans DB idle dapat menaikkan skalanya kembali ke kapasitas maksimum dengan lebih cepat.

## Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster

Pengguna cenderung selalu memilih nilai tinggi tertentu untuk pengaturan kapasitas Aurora Serverless v2 maksimum. Kapasitas maksimum yang besar memungkinkan instans DB menaikkan skala secara maksimal saat menjalankan beban kerja intensif. Nilai rendah menghindari kemungkinan biaya tak terduga. Bergantung pada cara Anda menggunakan klaster tersebut dan pengaturan lain yang Anda konfigurasikan, nilai yang paling efektif mungkin lebih tinggi atau lebih rendah dari yang Anda kira. Pertimbangkan faktor-faktor berikut saat memilih pengaturan kapasitas maksimum:

- Kapasitas maksimum harus minimal setinggi kapasitas minimum. Anda dapat mengatur kapasitas minimum dan maksimum agar identik. Namun, dalam hal ini kapasitas tidak pernah naik atau turun. Dengan demikian, menggunakan nilai yang identik untuk kapasitas minimum dan maksimum tidak tepat untuk situasi di luar pengujian.
- Kapasitas maksimum harus lebih tinggi dari 0,5 ACUs. Anda dapat mengatur kapasitas minimum dan maksimum agar sama dalam banyak kasus. Namun, Anda tidak dapat menentukan 0,5 untuk minimum dan maksimum. Gunakan nilai 1 atau lebih tinggi untuk kapasitas maksimum.
- Jika Anda biasanya memodifikasi kelas instans DB Anda untuk mengantisipasi beban kerja yang sangat tinggi atau rendah, Anda dapat menggunakan pengalaman ini untuk memperkirakan rentang kapasitas Aurora Serverless v2 yang setara. Untuk menentukan ukuran memori yang akan

digunakan pada saat lalu lintas tinggi, baca Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Misalnya, anggaplah Anda menggunakan kelas instans DB db.r6g.4xlarge ketika klaster Anda memiliki beban kerja yang tinggi. Kelas instans DB tersebut memiliki memori 128 GiB. Dengan demikian, Anda dapat menentukan pengaturan ACU maksimum sebanyak 64 untuk menyiapkan instans DB Aurora Serverless v2 yang dapat menaikkan skalanya hingga mencapai kapasitas yang kira-kira sama. Itu karena setiap ACU setara dengan sekitar 2 GiB memori. Anda dapat menentukan nilai yang agak lebih tinggi agar instans DB menaikkan skalanya lebih jauh jika instans DB db.r6g.4xlarge Anda terkadang tidak memiliki kapasitas yang cukup untuk menangani beban kerja secara efektif.

- Jika Anda memiliki batas anggaran pada penggunaan basis data Anda, pilih nilai yang tetap dalam batas tersebut bahkan jika semua instans DB Aurora Serverless v2 Anda berjalan pada kapasitas maksimum sepanjang waktu. Ingatlah bahwa ketika Anda memiliki n instans DB Aurora Serverless v2 di klaster Anda, kapasitas Aurora Serverless v2 maksimum teoretis yang dapat dikonsumsi klaster setiap saat adalah n kali pengaturan ACU maksimum untuk klaster. (Jumlah sebenarnya yang dikonsumsi mungkin lebih sedikit, misalnya jika beberapa pembaca diskalakan secara terpisah dari penulis.)
- Jika Anda menggunakan instans DB Aurora Serverless v2 pembaca untuk mengalihkan beberapa beban kerja hanya baca dari instans DB penulis, Anda mungkin dapat memilih pengaturan kapasitas maksimum yang lebih rendah. Anda melakukannya agar setiap instans DB pembaca tidak perlu diskalakan sama tingginya saat klaster hanya berisi satu instans DB.
- Misalkan Anda ingin mencegah penggunaan yang berlebihan karena parameter basis data yang salah dikonfigurasi atau kueri yang tidak efisien dalam aplikasi Anda. Dalam hal ini, Anda dapat menghindari penggunaan berlebihan yang tidak disengaja dengan memilih pengaturan kapasitas maksimum yang lebih rendah dari pengaturan tertinggi absolut yang dapat Anda tetapkan.
- Jika lonjakan karena aktivitas pengguna sebenarnya jarang tetapi dapat terjadi, Anda dapat mempertimbangkan situasi tersebut saat memilih pengaturan kapasitas maksimum. Jika prioritasnya adalah agar aplikasi tetap berjalan dengan performa dan skalabilitas penuh, Anda dapat menentukan pengaturan kapasitas maksimum yang lebih tinggi daripada yang Anda amati dalam penggunaan normal. Jika Anda tidak memiliki masalah dengan aplikasi yang berjalan dengan throughput yang berkurang selama lonjakan aktivitas yang sangat ekstrem, Anda dapat memilih pengaturan kapasitas maksimum yang sedikit lebih rendah. Pastikan Anda memilih pengaturan yang masih memiliki cukup memori dan sumber daya CPU untuk menjaga aplikasi tetap berjalan.

- Jika Anda mengaktifkan pengaturan di klaster Anda yang meningkatkan penggunaan memori untuk setiap instans DB, pertimbangkan memori tersebut saat memutuskan nilai ACU maksimum. Pengaturan tersebut mencakup pengaturan untuk Wawasan Performa, kueri paralel Aurora MySQL, skema performa Aurora MySQL, dan replikasi log biner Aurora MySQL. Pastikan bahwa nilai ACU maksimum memungkinkan instans DB Aurora Serverless v2 menaikkan skalanya secara memadai untuk menangani beban kerja saat fitur tersebut digunakan. Untuk informasi tentang pemecahan masalah yang disebabkan oleh kombinasi pengaturan ACU maksimum rendah dan fitur Aurora yang memerlukan overhead memori, lihat Menghindari out-of-memory kesalahan.

## Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL

AWS CLI Contoh berikut menunjukkan cara memperbarui rentang ACU untuk instans Aurora Serverless v2 DB di cluster Aurora MySQL yang ada. Awalnya, rentang kapasitas untuk cluster adalah 8-32 ACUs.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

Instans DB menganggur dan diperkecil menjadi 8. ACUs Pengaturan terkait kapasitas berikut berlaku untuk instans DB pada saat ini. Untuk merepresentasikan ukuran pool buffer dalam unit yang mudah dibaca, kita membaginya dengan 2 pangkat 30, sehingga menghasilkan pengukuran dalam gibibyte (GiB). Itu karena pengukuran terkait memori untuk Aurora menggunakan unit berdasarkan pangkat 2, bukan pangkat 10.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
```

```
+---------------------------+
|                9294577664 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

Selanjutnya, kita mengubah rentang kapasitas untuk klaster. Setelah perintah `modify-db-cluster` selesai, rentang ACU untuk klaster adalah 12,5–80.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Mengubah rentang kapasitas menyebabkan perubahan pada nilai default dari beberapa parameter konfigurasi. Aurora dapat segera menerapkan beberapa nilai default baru tersebut. Namun, beberapa perubahan parameter hanya berlaku setelah boot ulang. Status `pending-reboot` menunjukkan bahwa boot ulang diperlukan untuk menerapkan beberapa perubahan parameter.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
```

```
}
```

Pada titik ini, cluster menganggur dan instans `serverless-v2-instance-1` DB mengkonsumsi ACUs 12,5. Parameter`innodb_buffer_pool_size` sudah disesuaikan berdasarkan kapasitas instans DB saat ini. Parameter `max_connections` masih mencerminkan nilai dari kapasitas maksimum sebelumnya. Jika nilai tersebut direset, instans DB harus di-boot ulang.

> **Note**
>
> Jika Anda mengatur parameter `max_connections` secara langsung dalam grup parameter DB kustom, boot ulang tidak diperlukan.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+---------------+
| gibibytes     |
+---------------+
| 14.5029296875 |
+---------------+
1 row in set (0.00 sec)
```

Sekarang kita mem-boot ulang instans DB dan menunggu hingga instans tersebut tersedia lagi.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
```

```
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Status `pending-reboot` dihapus. Nilai `in-sync` mengonfirmasi bahwa Aurora telah menerapkan semua perubahan parameter yang tertunda.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

Parameter `innodb_buffer_pool_size` telah meningkat ke ukuran akhir untuk instans DB idle. Parameter `max_connections` telah meningkat untuk mencerminkan nilai yang berasal dari nilai ACU maksimum. Rumus yang digunakan Aurora untuk `max_connections` menyebabkan peningkatan 1.000 ketika ukuran memori berlipat ganda.

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               16139681792 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)
```

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
+-------------------+
1 row in set (0.00 sec)
```

Kami mengatur rentang kapasitas menjadi 0,5-128 ACUs, dan me-reboot instans DB. Sekarang instans DB idle memiliki ukuran cache buffer yang kurang dari 1 GiB, jadi kita mengukurnya dalam mebibyte (MiB). Nilai `max_connections` 5000 berasal dari ukuran memori pada pengaturan kapasitas maksimum.

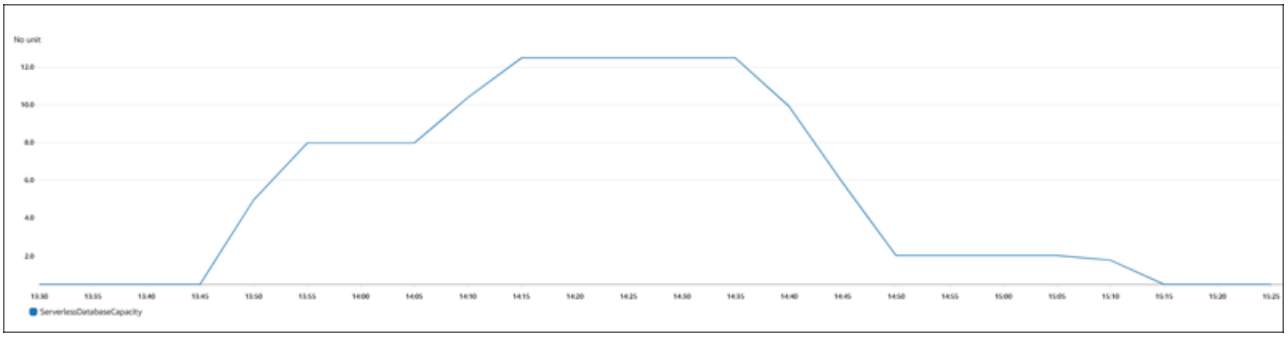
```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+-------------------+
| mebibytes | @@max_connections |
+-----------+-------------------+
|       672 |              5000 |
+-----------+-------------------+
1 row in set (0.00 sec)
```

## Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL

Contoh CLI berikut menunjukkan cara memperbarui rentang ACU untuk instans DB Aurora Serverless v2 di klaster Aurora PostgreSQL yang sudah ada.

1. Rentang kapasitas untuk klaster dimulai pada 0,5–1 ACU.
2. Ubah rentang kapasitas menjadi 8-32 ACUs.
3. Ubah rentang kapasitas menjadi 12,5-80 ACUs.
4. Ubah rentang kapasitas menjadi 0,5—128 ACUs.
5. Kembalikan kapasitas ke rentang awal 0,5–1 ACU.

Gambar berikut menunjukkan perubahan kapasitas di Amazon CloudWatch.




Instans DB menganggur dan diperkecil menjadi 0,5. ACUs Pengaturan terkait kapasitas berikut berlaku untuk instans DB pada saat ini.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Selanjutnya, kita mengubah rentang kapasitas untuk klaster. Setelah perintah `modify-db-cluster` selesai, rentang ACU untuk klaster adalah 8.0–32.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

Mengubah rentang kapasitas menyebabkan perubahan pada nilai default dari beberapa parameter konfigurasi. Aurora dapat segera menerapkan beberapa nilai default baru tersebut. Namun, beberapa perubahan parameter hanya berlaku setelah boot ulang. Status `pending-reboot` menunjukkan bahwa Anda harus melakukan boot ulang untuk menerapkan beberapa perubahan parameter.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

Pada titik ini, cluster menganggur dan instance `serverless-v2-instance-1` DB mengkonsumsi 8.0 ACUs. Parameter`shared_buffers` sudah disesuaikan berdasarkan kapasitas instans DB saat ini. Parameter `max_connections` masih mencerminkan nilai dari kapasitas maksimum sebelumnya. Jika nilai tersebut direset, instans DB harus di-boot ulang.

> **Note**
>
> Jika Anda mengatur parameter `max_connections` secara langsung dalam grup parameter DB kustom, boot ulang tidak diperlukan.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

Kita mem-boot ulang instans DB dan menunggu hingga instans tersebut tersedia lagi.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
```

```
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Sekarang setelah instans DB di-boot ulang, status `pending-reboot` dihapus. Nilai `in-sync` mengonfirmasi bahwa Aurora telah menerapkan semua perubahan parameter yang tertunda.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

Setelah boot ulang, `max_connections` menunjukkan nilai dari kapasitas maksimum baru.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

Selanjutnya, kami mengubah rentang kapasitas untuk cluster menjadi 12, 5-80 ACUs.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
```

```
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Pada titik ini, cluster menganggur dan instans `serverless-v2-instance-1` DB mengkonsumsi ACUs 12,5. Parameter`shared_buffers` sudah disesuaikan berdasarkan kapasitas instans DB saat ini. Nilai `max_connections` masih 5000.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

Kita mem-boot ulang lagi, tetapi nilai parameternya tetap sama. Ini karena `max_connections` memiliki nilai maksimum 5000 untuk klaster DB Aurora Serverless v2 yang menjalankan Aurora PostgreSQL.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

Sekarang kami mengatur kisaran kapasitas dari 0,5 hingga 128 ACUs. Skala cluster DB turun ke 10 ACUs, lalu ke 2. Kita mem-boot ulang instans DB.

```
postgres=> show max_connections;
 max_connections
```

```
-----------------
 2000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

`max_connections`Nilai untuk instans Aurora Serverless v2 DB didasarkan pada ukuran memori yang berasal dari maksimum ACUs. Namun, ketika Anda menentukan kapasitas minimum 0 atau 0,5 ACUs pada instans DB yang kompatibel dengan PostgreSQL, nilai maksimum dibatasi pada 2.000. `max_connections`

Sekarang kita mengembalikan kapasitas ke rentang awal 0,5–1 ACU dan mem-boot ulang instans DB. Parameter `max_connections` telah kembali ke nilai aslinya.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

## Menggunakan grup parameter untuk Aurora Serverless v2

Saat Anda membuat klaster DB Aurora Serverless v2, Anda memilih mesin Aurora DB tertentu dan grup parameter klaster DB terkait. Jika Anda tidak memahami cara Aurora menggunakan grup parameter untuk menerapkan pengaturan konfigurasi secara konsisten di seluruh klaster, lihat . Semua prosedur tersebut untuk membuat, memodifikasi, menerapkan, dan tindakan lain untuk grup parameter akan berlaku untuk Aurora Serverless v2.

Fitur grup parameter beroperasi sama antara klaster terprovisi dan klaster yang berisi instans DB Aurora Serverless v2:

- Nilai parameter default untuk semua instans DB di klaster ditentukan oleh grup parameter klaster.
- Anda dapat mengganti beberapa parameter untuk instans DB tertentu dengan menentukan grup parameter DB kustom untuk instans DB tersebut. Anda dapat melakukannya selama debugging atau penyesuaian performa untuk instans DB tertentu. Misalnya, anggaplah bahwa Anda memiliki klaster yang berisi beberapa instans DB Aurora Serverless v2 dan beberapa instans DB terprovisi. Dalam kasus ini, Anda dapat menentukan beberapa parameter berbeda untuk instans DB terprovisi dengan menggunakan grup parameter DB kustom.
- Untuk Aurora Serverless v2, Anda dapat menggunakan semua parameter yang memiliki nilai `provisioned` dalam atribut `SupportedEngineModes` di grup parameter. Di Aurora Serverless v1, Anda hanya dapat menggunakan subset parameter yang memiliki `serverless` dalam atribut `SupportedEngineModes`.

Topik


- Nilai parameter default
- Koneksi maksimum untuk Aurora Serverless v2
- Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya
- Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2


## Nilai parameter default

Perbedaan penting antara instans DB terprovisi dan instans DB Aurora Serverless v2 adalah bahwa Aurora mengganti nilai parameter kustom apa pun untuk parameter tertentu yang terkait dengan kapasitas instans DB. Nilai parameter kustom masih berlaku untuk instans DB terprovisi di klaster Anda. Untuk detail selengkapnya tentang bagaimana instans DB Aurora Serverless v2 menafsirkan parameter dari grup parameter Aurora, lihat Parameter konfigurasi untuk klaster Aurora. Untuk parameter spesifik yang diganti oleh Aurora Serverless v2, lihat Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya dan Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2.

Anda bisa mendapatkan daftar nilai default untuk grup parameter default untuk berbagai mesin Aurora DB dengan menggunakan perintah describe-db-cluster-parametersCLI dan menanyakan file. Wilayah AWS Berikut ini adalah nilai-nilai yang dapat Anda gunakan untuk opsi `--db-parameter-group-family` dan `-db-parameter-group-name` untuk versi mesin yang kompatibel dengan Aurora Serverless v2.

| Mesin dan versi basis data | Kelompok grup parameter | Nama grup parameter default |
|---|---|---|
| Aurora MySQL versi 3 | `aurora-mysql8.0` | `default.aurora-mysql8.0` |
| Aurora PostgreSQL versi 13.x | `aurora-postgresql13` | `default.aurora-postgresql13` |
| Aurora PostgreSQL versi 14.x | `aurora-postgresql14` | `default.aurora-postgresql14` |
| Aurora PostgreSQL versi 15.x | `aurora-postgresql15` | `default.aurora-postgresql15` |
| Aurora PostgreSQL versi 16.x | `aurora-postgresql16` | `default.aurora-postgresql16` |
| Aurora PostgreSQL versi 17.x | `aurora-postgresql17` | `default.aurora-postgresql17` |

Contoh berikut mendapatkan daftar parameter dari grup klaster DB default untuk Aurora MySQL versi 3 dan Aurora PostgreSQL 13. Ini adalah versi Aurora MySQL dan Aurora PostgreSQL yang Anda gunakan dengan Aurora Serverless v2.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text

aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

## Koneksi maksimum untuk Aurora Serverless v2

Untuk Aurora MySQL dan Aurora PostgreSQL, instans DB Aurora Serverless v2 menjaga parameter `max_connections` tetap konstan sehingga koneksi tidak terputus saat instans DB diturunkan skalanya. Nilai default untuk parameter ini berasal dari rumus berdasarkan ukuran memori instans DB. Untuk detail tentang rumus dan nilai default untuk kelas instans DB terprovisi, lihat Koneksi maksimum ke instans DB Aurora MySQL dan Koneksi maksimum ke instans DB Aurora PostgreSQL.

Ketika Aurora Serverless v2 mengevaluasi rumus, ia menggunakan ukuran memori berdasarkan unit kapasitas Aurora maksimum ACUs () untuk instans DB, bukan nilai ACU saat ini. Jika Anda mengubah nilai default, sebaiknya gunakan variasi rumus alih-alih menentukan nilai konstan. Dengan begitu, Aurora Serverless v2 bisa menggunakan pengaturan yang sesuai berdasarkan kapasitas maksimum.

Saat Anda mengubah kapasitas maksimum klaster DB Aurora Serverless v2, Anda harus mem-boot ulang instans DB Aurora Serverless v2 untuk memperbarui nilai `max_connections`. Ini karena `max_connections` merupakan parameter statis untuk Aurora Serverless v2.

Tabel berikut menunjukkan nilai default untuk `max_connections` bagi Aurora Serverless v2 berdasarkan nilai ACU maksimum.

| Maksimum ACUs | Koneksi maksimum default pada Aurora MySQL | Koneksi maksimum default pada Aurora PostgreSQL |
|---|---|---|
| 1 | 90 | 189 |
| 4 | 135 | 823 |
| 8 | 1.000 | 1,669 |
| 16 | 2.000 | 3,360 |
| 32 | 3.000 | 5.000 |
| 64 | 4.000 | 5.000 |
| 128 | 5.000 | 5.000 |
| 192 | 6.000 | 5.000 |
| 256 | 6.000 | 5.000 |

> **Note**
>
> `max_connections`Nilai untuk instans Aurora Serverless v2 DB didasarkan pada ukuran memori yang berasal dari maksimum ACUs. Namun, ketika Anda menentukan kapasitas minimum 0 atau 0,5 ACUs pada instans DB yang kompatibel dengan PostgreSQL, nilai maksimum dibatasi pada 2.000. `max_connections`

Untuk contoh spesifik yang menunjukkan perubahan `max_connections` dengan nilai ACU maksimum, lihat Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL dan Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL.

## Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya

Selama penskalaan otomatis, Aurora Serverless v2 harus dapat mengubah parameter agar setiap instans DB beroperasi dengan paling efektif untuk peningkatan atau penurunan kapasitas. Dengan demikian, Anda tidak dapat mengganti beberapa parameter yang terkait dengan kapasitas. Untuk

beberapa parameter yang dapat Anda ganti, hindari melakukan hard-coding nilai tetap. Pertimbangan berikut berlaku untuk pengaturan ini yang terkait dengan kapasitas.

Untuk Aurora MySQL, Aurora Serverless v2 mengubah ukuran beberapa parameter secara dinamis selama penskalaan. Untuk parameter berikut, Aurora Serverless v2 tidak menggunakan nilai parameter kustom apa pun yang Anda tentukan:

- `innodb_buffer_pool_size`
- `innodb_purge_threads`
- `table_definition_cache`
- `table_open_cache`

Untuk Aurora PostgreSQL, Aurora Serverless v2 mengubah ukuran parameter berikut secara dinamis selama penskalaan. Untuk parameter berikut, Aurora Serverless v2 tidak menggunakan nilai parameter kustom apa pun yang Anda tentukan:

- `shared_buffers`

Untuk semua parameter selain yang tercantum di sini, instans Aurora Serverless v2 DB beroperasi sama dengan instans DB terprovisi. Nilai parameter default diwarisi dari grup parameter klaster. Anda dapat memodifikasi nilai default untuk seluruh klaster dengan menggunakan grup parameter klaster kustom. Atau Anda dapat memodifikasi nilai default untuk instans DB tertentu dengan menggunakan grup parameter DB kustom. Parameter dinamis akan segera diperbarui. Perubahan parameter statis hanya berlaku setelah Anda mem-boot ulang instans DB.

## Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2

Untuk parameter berikut, Aurora PostgreSQL menggunakan nilai default yang berasal dari ukuran memori berdasarkan pengaturan ACU maksimum, sama seperti dengan `max_connections`:

- `autovacuum_max_workers`
- `autovacuum_vacuum_cost_limit`
- `autovacuum_work_mem`
- `effective_cache_size`
- `maintenance_work_mem`

# Menghindari out-of-memory kesalahan

Jika salah satu instans DB Aurora Serverless v2 Anda secara konsisten mencapai batas kapasitas maksimumnya, Aurora akan menunjukkan kondisi ini dengan mengatur instans DB ke status `incompatible-parameters`. Saat instans DB memiliki status `incompatible-parameters`, beberapa operasi akan diblokir. Misalnya, Anda tidak dapat meningkatkan versi mesin.

Biasanya, instans DB Anda masuk ke status ini ketika sering restart karena out-of-memory kesalahan. Aurora mencatat sebuah peristiwa ketika jenis pengaktifan ulang ini terjadi. Anda dapat melihat peristiwa ini dengan mengikuti prosedur dalam Melihat RDS acara Amazon. Penggunaan memori yang luar biasa tinggi dapat terjadi karena overhead yang timbul saat mengaktifkan pengaturan seperti Wawasan Performa dan autentikasi IAM. Hal ini juga bisa berasal dari beban kerja yang berat pada instans DB Anda atau pengelolaan metadata yang terkait dengan sejumlah besar objek skema.

Jika tekanan memori menjadi lebih rendah sehingga instans DB sangat sering tidak mencapai kapasitas maksimumnya, Aurora akan secara otomatis mengubah status instans DB kembali ke `available`.

Untuk pulih dari kondisi ini, Anda dapat mengambil beberapa atau semua tindakan berikut:

- Tingkatkan batas bawah kapasitas untuk instans DB Aurora Serverless v2 dengan mengubah nilai unit kapasitas Aurora (ACU) minimum untuk klaster. Tindakan ini menghindari masalah saat basis data idle menurunkan skalanya ke kapasitas dengan memori lebih sedikit daripada yang dibutuhkan untuk fitur yang diaktifkan di klaster Anda. Setelah mengubah pengaturan ACU untuk klaster, boot ulang instans DB Aurora Serverless v2. Tindakan ini mengevaluasi apakah Aurora dapat mereset status kembali ke `available`.
- Tingkatkan batas atas kapasitas instans DB Aurora Serverless v2 dengan mengubah nilai ACU maksimum untuk klaster. Tindakan ini menghindari masalah saat basis data yang sibuk tidak dapat menaikkan skalanya ke kapasitas dengan memori yang cukup untuk fitur yang diaktifkan di klaster Anda dan beban kerja basis data. Setelah mengubah pengaturan ACU untuk klaster, boot ulang instans DB Aurora Serverless v2. Tindakan ini mengevaluasi apakah Aurora dapat mereset status kembali ke `available`.
- Nonaktifkan pengaturan konfigurasi yang memerlukan overhead memori. Misalnya, Anda memiliki fitur seperti AWS Identity and Access Management (IAM), Performance Insights, atau replikasi log biner Aurora MySQL yang diaktifkan tetapi tidak menggunakannya. Jika demikian, Anda dapat menonaktifkannya. Atau Anda dapat menambah nilai kapasitas minimum dan maksimum untuk klaster untuk memperhitungkan memori yang digunakan oleh fitur-fitur tersebut. Untuk panduan

tentang memilih pengaturan kapasitas minimum dan maksimum, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora.

- Kurangi beban kerja pada instans DB. Misalnya, Anda dapat menambahkan instans DB pembaca ke klaster untuk menyebarkan beban dari kueri hanya baca ke lebih banyak instans DB.
- Sesuaikan kode SQL yang digunakan oleh aplikasi Anda untuk menggunakan lebih sedikit sumber daya. Misalnya, Anda dapat memeriksa rencana kueri Anda, memeriksa log kueri yang lambat, atau menyesuaikan indeks pada tabel Anda. Anda juga dapat melakukan jenis penyesuaian SQL standar lainnya.

## CloudWatch Metrik Amazon penting untuk Aurora Serverless v2

Untuk memulai Amazon CloudWatch untuk instans Aurora Serverless v2 DB Anda, lihatMelihat Aurora Serverless v2 log di Amazon CloudWatch. Untuk mempelajari lebih lanjut tentang cara memantau klaster Aurora DB, lihat CloudWatch. Memantau peristiwa log di Amazon CloudWatch

Anda dapat melihat instans Aurora Serverless v2 DB Anda CloudWatch untuk memantau kapasitas yang dikonsumsi oleh setiap instans DB dengan `ServerlessDatabaseCapacity` metrik. Anda juga dapat memantau semua CloudWatch metrik Aurora standar, seperti dan. `DatabaseConnections` `Queries` Untuk daftar lengkap CloudWatch metrik yang dapat Anda pantau untuk Aurora, lihat. CloudWatch Metrik Amazon untuk Amazon Aurora Metrik dibagi menjadi metrik tingkat klaster dan tingkat instans, di Metrik tingkat klaster untuk Amazon Aurora dan Metrik tingkat instans untuk Amazon Aurora.

Metrik CloudWatch tingkat instans berikut ini penting untuk dipantau agar Anda memahami bagaimana instans Aurora Serverless v2 DB Anda meningkat dan turun. Semua metrik ini dihitung setiap detik. Dengan begitu, Anda dapat memantau status instans DB Aurora Serverless v2 Anda saat ini. Anda dapat mengatur alarm untuk memberi tahu Anda jika ada instans DB Aurora Serverless v2 yang mendekati ambang batas untuk metrik yang terkait dengan kapasitas. Anda dapat menentukan apakah pengaturan kapasitas minimum dan maksimum sudah sesuai, atau apakah Anda perlu menyesuaikannya. Anda dapat menentukan di mana upaya Anda harus difokuskan untuk mengoptimalkan efisiensi basis data Anda.

- `ServerlessDatabaseCapacity`. Sebagai metrik tingkat instance, ia melaporkan jumlah yang ACUs diwakili oleh kapasitas instans DB saat ini. Sebagai metrik tingkat klaster, metrik ini merepresentasikan rata-rata nilai `ServerlessDatabaseCapacity` milik semua instans DB Aurora Serverless v2 di klaster. Metrik ini adalah satu-satunya metrik tingkat klaster di Aurora

Serverless v1. Di Aurora Serverless v2, metrik ini tersedia di tingkat instans DB dan di tingkat klaster.

- `ACUUtilization`. Ini adalah metrik baru di Aurora Serverless v2. Nilai ini direpresentasikan sebagai persentase. Metrik ini dihitung sebagai nilai metrik `ServerlessDatabaseCapacity` dibagi dengan nilai ACU maksimum klaster DB. Pertimbangkan pedoman berikut untuk menafsirkan metrik ini dan mengambil tindakan:
  - Jika metrik ini mendekati nilai `100.0`, instans DB telah dinaikkan skalanya setinggi mungkin. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Dengan begitu, instans DB penulis dan pembaca dapat diskalakan ke kapasitas yang lebih tinggi.
  - Misalkan beban kerja hanya baca menyebabkan instans DB pembaca mendekati `ACUUtilization` sebesar `100.0`, sedangkan instans DB penulis tidak mendekati kapasitas maksimumnya. Dalam hal ini, pertimbangkan untuk menambahkan instans DB pembaca lainnya ke klaster. Dengan begitu, Anda dapat menyebarkan bagian hanya baca dari beban kerja yang tersebar di lebih banyak instans DB, sehingga mengurangi beban pada setiap instans DB pembaca.
  - Misalkan Anda menjalankan aplikasi produksi, sehingga performa dan skalabilitas adalah pertimbangan utamanya. Dalam hal ini, Anda dapat mengatur nilai ACU maksimum untuk klaster ke angka yang tinggi. Tujuan Anda adalah agar metrik `ACUUtilization` selalu berada di bawah `100.0`. Dengan nilai ACU maksimum yang tinggi, Anda dapat yakin bahwa ada cukup ruang jika ada lonjakan tak terduga dalam aktivitas basis data. Anda hanya dikenai biaya untuk kapasitas basis data yang benar-benar dikonsumsi.
- `CPUUtilization`. Metrik ini ditafsirkan secara berbeda di Aurora Serverless v2 daripada di instans DB terprovisi. Untuk Aurora Serverless v2, nilai ini adalah persentase yang dihitung sebagai jumlah CPU yang saat ini digunakan dibagi dengan kapasitas CPU yang tersedia di bawah nilai ACU maksimum klaster DB. Aurora memantau nilai ini secara otomatis dan menaikkan skala instans DB Aurora Serverless v2 Anda ketika instans DB secara konsisten menggunakan proporsi kapasitas CPU-nya yang tinggi.

  Jika metrik ini mendekati nilai `100.0`, instans DB telah mencapai kapasitas CPU maksimumnya. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Jika metrik ini mendekati nilai `100.0` pada instans DB pembaca, pertimbangkan untuk menambahkan instans DB pembaca lain ke klaster. Dengan begitu, Anda dapat menyebarkan bagian hanya baca dari beban kerja yang tersebar di lebih banyak instans DB, sehingga mengurangi beban pada setiap instans DB pembaca.
- `FreeableMemory`. Nilai ini merepresentasikan jumlah memori yang tidak terpakai yang tersedia ketika instans DB Aurora Serverless v2 diskalakan ke kapasitas maksimumnya. Untuk setiap

ACU yang kapasitasnya saat ini di bawah kapasitas maksimum, nilai ini meningkat sekitar 2 GiB. Dengan demikian, metrik ini tidak mendekati nol sampai instans DB dinaikkan skalanya setinggi mungkin.

Jika metrik ini mendekati nilai `0`, instans DB telah dinaikkan skalanya setinggi mungkin dan mendekati batas memori yang tersedia. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Jika metrik ini mendekati nilai `0` pada instans DB pembaca, pertimbangkan untuk menambahkan instans DB pembaca lain ke klaster. Dengan begitu, bagian hanya baca dari beban kerja dapat disebarkan ke lebih banyak instans DB, sehingga mengurangi penggunaan memori pada setiap instans DB pembaca.

- `TempStorageIOPS`. Jumlah IOPS yang dilakukan pada penyimpanan lokal yang dilampirkan ke instans DB. Ini termasuk IOPS untuk baca dan tulis. Metrik ini merepresentasikan hitungan dan diukur sekali per detik. Ini adalah metrik baru untuk Aurora Serverless v2. Untuk detailnya, lihat Metrik tingkat instans untuk Amazon Aurora.
- `TempStorageThroughput`. Jumlah data yang ditransfer ke dan dari penyimpanan lokal yang terkait dengan instans DB. Metrik ini merepresentasikan byte dan diukur sekali per detik. Ini adalah metrik baru untuk Aurora Serverless v2. Untuk detailnya, lihat Metrik tingkat instans untuk Amazon Aurora.

Biasanya, sebagian besar penskalaan untuk instans DB Aurora Serverless v2 disebabkan oleh penggunaan memori dan aktivitas CPU. Metrik `TempStorageIOPS` dan `TempStorageThroughput` metrik dapat membantu Anda mendiagnosis kasus yang jarang terjadi saat aktivitas jaringan untuk transfer antara instans DB dan perangkat penyimpanan lokal menimbulkan peningkatan kapasitas yang tidak terduga. Untuk memantau aktivitas jaringan lainnya, Anda dapat menggunakan metrik yang ada ini:

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

Anda dapat meminta Aurora menerbitkan beberapa atau semua log database ke Amazon CloudWatch Logs. Untuk petunjuk, lihat yang berikut ini tergantung pada mesin database Anda:

- the section called “Menerbitkan log PostgreSQL Aurora ke Log CloudWatch ”
- the section called “Menerbitkan log Aurora MySQL ke Log CloudWatch ”

## Bagaimana Aurora Serverless v2 metrik berlaku untuk tagihan Anda AWS

Aurora Serverless v2Biaya pada AWS tagihan Anda dihitung berdasarkan `ServerlessDatabaseCapacity` metrik yang sama yang dapat Anda pantau. Mekanisme penagihan dapat berbeda dari CloudWatch rata-rata yang dihitung untuk metrik ini jika Anda menggunakan Aurora Serverless v2 kapasitas hanya sebagian dari satu jam. Ini juga dapat berbeda jika masalah sistem membuat CloudWatch metrik tidak tersedia untuk periode singkat. Dengan demikian, Anda mungkin melihat nilai ACU-jam yang sedikit berbeda pada tagihan Anda daripada jika Anda menghitung sendiri angkanya dari nilai rata-rata `ServerlessDatabaseCapacity`.

## Contoh CloudWatch perintah untuk Aurora Serverless v2 metrik

AWS CLI Contoh berikut menunjukkan bagaimana Anda dapat memantau CloudWatch metrik terpenting yang terkait Aurora Serverless v2 dengannya. Dalam setiap kasus, ganti string `Value=` untuk parameter `--dimensions` dengan pengidentifikasi instans DB Aurora Serverless v2 Anda sendiri.

Contoh Linux berikut menampilkan nilai kapasitas minimum, maksimum, dan rata-rata untuk instans DB, yang diukur setiap 10 menit selama satu jam. Perintah `date` Linux menentukan waktu mulai dan akhir secara relatif terhadap tanggal dan waktu saat ini. Fungsi `sort_by` dalam parameter `--query` mengurutkan hasil secara kronologis berdasarkan bidang `Timestamp`.

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Contoh Linux berikut menunjukkan pemantauan kapasitas setiap instans DB dalam sebuah klaster. Hal ini mengukur pemanfaatan kapasitas minimum, maksimum, dan rata-rata dari setiap instans DB. Pengukuran dilakukan sekali setiap jam selama periode tiga jam. Contoh-contoh ini menggunakan `ACUUtilization` metrik yang mewakili persentase dari batas atas ACUs, alih-alih `ServerlessDatabaseCapacity` mewakili jumlah tetap ACUs. Dengan begitu, Anda tidak perlu mengetahui angka sebenarnya untuk nilai ACU minimum dan maksimum dalam rentang kapasitas. Anda dapat melihat persentase mulai dari 0 hingga 100.

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Contoh Linux berikut melakukan pengukuran yang sama seperti yang sebelumnya. Dalam hal ini, pengukuran ditujukan untuk metrik `CPUUtilization`. Pengukuran dilakukan setiap 10 menit selama periode 1 jam. Angka-angka tersebut merepresentasikan persentase CPU yang tersedia yang digunakan, berdasarkan sumber daya CPU yang tersedia untuk pengaturan kapasitas maksimum untuk instans DB.

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Contoh Linux berikut melakukan pengukuran yang sama seperti yang sebelumnya. Dalam hal ini, pengukuran ditujukan untuk metrik `FreeableMemory`. Pengukuran dilakukan setiap 10 menit selama periode 1 jam.

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

## Memantau performa Aurora Serverless v2 dengan Wawasan Performa

Anda dapat menggunakan Wawasan Performa untuk memantau performa instans DB Aurora Serverless v2. Untuk prosedur Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Penghitung Wawasan Performa baru berikut ini berlaku untuk instans DB Aurora Serverless v2:

- `os.general.serverlessDatabaseCapacity`- Kapasitas saat ini dari instans DB di ACUs. Nilai sesuai dengan `ServerlessDatabaseCapacity` CloudWatch metrik untuk instance DB.
- `os.general.acuUtilization` – Persentase kapasitas saat ini dari kapasitas maksimum yang dikonfigurasi. Nilai sesuai dengan `ACUUtilization` CloudWatch metrik untuk instance DB.
- `os.general.maxConfiguredAcu` – Kapasitas maksimum yang Anda konfigurasikan untuk instans DB Aurora Serverless v2 ini. Itu diukur dalam ACUs.
- `os.general.minConfiguredAcu` – Kapasitas minimum yang Anda konfigurasikan untuk instans DB Aurora Serverless v2 ini. Hal ini diukur dalam ACUs

Untuk daftar lengkap penghitung Wawasan Performa, lihat Metrik penghitung Wawasan Performa.

Ketika nilai `vCPU` ditampilkan untuk instans DB Aurora Serverless v2 di Wawasan Performa, nilai tersebut merepresentasikan perkiraan berdasarkan nilai ACU untuk instans DB ini. Pada interval default satu menit, nilai vCPU fraksional apa pun dibulatkan ke bilangan bulat terdekat. Untuk interval yang lebih lama, nilai vCPU yang ditampilkan adalah rata-rata nilai vCPU bilangan bulat untuk setiap menit.

## Memecahkan masalah kapasitas Aurora Serverless v2

Dalam beberapa kasus, Aurora Serverless v2 tidak diturunkan skalanya ke kapasitas minimum, bahkan tanpa beban pada basis data. Hal ini dapat terjadi karena alasan berikut:

- Fitur tertentu dapat meningkatkan penggunaan sumber daya dan mencegah basis data diturunkan skalanya ke kapasitas minimum. Fitur-fitur ini mencakup hal-hal berikut:
  - Basis data global Aurora
  - Mengekspor CloudWatch Log
  - Mengaktifkan `pg_audit` pada klaster DB yang kompatibel dengan Aurora PostgreSQL
  - Pemantauan yang Ditingkatkan
  - Wawasan Performa

Untuk informasi selengkapnya, lihat Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster.

- Jika instans pembaca tidak menurunkan skalanya ke nilai minimum dan tetap pada kapasitas yang sama atau lebih tinggi dari instans penulis, periksa tingkat prioritas instans pembaca. Instans DB pembaca Aurora Serverless v2 di tingkat 0 atau 1 akan dipertahankan pada kapasitas minimum setidaknya setinggi instans DB penulis. Ubah tingkat prioritas pembaca menjadi 2 atau lebih tinggi agar dinaikkan dan diturunkan skalanya secara independen dari penulis. Untuk informasi selengkapnya, lihat Memilih tingkat promosi untuk Aurora Serverless v2 pembaca.
- Tetapkan parameter basis data apa pun yang memengaruhi ukuran memori bersama ke nilai default-nya. Mengatur nilai yang lebih tinggi dari nilai default akan meningkatkan kebutuhan memori bersama dan mencegah basis data menurunkan skalanya ke kapasitas minimum. Contohnya adalah `max_connections` dan `max_locks_per_transaction`.

> **Note**
>
> Memperbarui parameter memori bersama memerlukan pengaktifan ulang basis data agar perubahan diterapkan.

- Beban kerja basis data yang berat dapat meningkatkan penggunaan sumber daya.
- Volume basis data yang besar dapat meningkatkan penggunaan sumber daya.

  Amazon Aurora menggunakan memori dan sumber daya CPU untuk manajemen klaster DB. Aurora membutuhkan lebih banyak CPU dan memori untuk mengelola klaster DB dengan volume basis data yang lebih besar. Jika kapasitas minimum klaster Anda kurang dari minimum yang diperlukan untuk pengelolaan klaster, klaster Anda tidak akan menurunkan kapasitas minimum.
- Proses latar belakang, seperti purging, juga dapat meningkatkan penggunaan sumber daya.

Jika basis data masih tidak menurunkan skalanya ke kapasitas minimum yang dikonfigurasi, hentikan dan aktifkan ulang basis data untuk mengklaim kembali fragmen memori yang mungkin telah terakumulasi dari waktu ke waktu. Menghentikan dan memulai basis data akan menghasilkan waktu henti, jadi sebaiknya lakukan ini seperlunya.

# Penskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2

Anda dapat menentukan bahwa Aurora Serverless v2 Instans DB menurunkan skala ke nol ACUs dan secara otomatis berhenti, jika mereka tidak memiliki koneksi yang dimulai oleh aktivitas pengguna dalam jangka waktu tertentu. Anda melakukannya dengan menentukan ACU nilai minimum nol untuk cluster DB Anda. Anda tidak dikenakan biaya untuk kapasitas instans saat instance dalam status jeda. Mengaktifkan fitur jeda dan melanjutkan otomatis (jeda otomatis) untuk klaster Aurora yang ringan digunakan atau memiliki periode tidak aktif yang lama dapat membantu Anda mengelola biaya untuk armada database Anda.

> **Note**
>
> Fitur jeda otomatis tersedia untuk Aurora Serverless v2 dengan Aurora Postgre SQL dan Aurora My. SQL Anda mungkin perlu memutakhirkan versi mesin basis data Aurora Anda untuk memanfaatkan fitur ini. Untuk versi mesin di mana kapasitas minimum 0 ACUs tersedia, lihatAurora Serverless v2 kapasitas.

Topik


- Sekilas tentang Aurora Serverless v2 fitur jeda otomatis
- Prasyarat dan Batasan untuk Aurora Serverless v2 fitur jeda otomatis
- Menghidupkan dan menonaktifkan fitur jeda otomatis
- Bagaimana Aurora Serverless v2 fitur jeda otomatis berfungsi
- Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk berbagai jenis cluster Aurora
- Memantau cluster Aurora yang menggunakan jeda otomatis
- Pemecahan masalah untuk Aurora Serverless v2 jeda otomatis
- Pertimbangan desain aplikasi untuk Aurora Serverless v2 fitur jeda otomatis


## Sekilas tentang Aurora Serverless v2 fitur jeda otomatis

Aurora Serverless v2 Instans DB dapat secara otomatis berhenti sejenak setelah periode tanpa koneksi pengguna, dan secara otomatis melanjutkan ketika permintaan koneksi tiba. Bagian Aurora Serverless v2 Fitur jeda/lanjutkan otomatis membantu mengelola biaya untuk sistem yang tidak memiliki tujuan tingkat layanan yang ketat (). SLO Misalnya, Anda dapat mengaktifkan fitur ini untuk

cluster yang digunakan untuk pengembangan dan pengujian, atau untuk aplikasi internal di mana jeda singkat dapat diterima saat database dilanjutkan. Jika beban kerja Anda memiliki periode tidak aktif dan dapat mentolerir sedikit keterlambatan dalam menghubungkan saat instans dilanjutkan, pertimbangkan untuk menggunakan jeda otomatis dengan Aurora Serverless v2 contoh untuk mengurangi biaya.

Anda mengontrol perilaku ini dengan menentukan apakah Aurora Serverless v2 Instans DB di cluster dapat secara otomatis berhenti atau tidak, dan berapa lama setiap instance harus menganggur sebelum berhenti. Untuk mengaktifkan perilaku jeda otomatis untuk semua Aurora Serverless v2 Instans DB di cluster Aurora, Anda menetapkan nilai kapasitas minimum untuk cluster ke nol. ACUs

Jika Anda sebelumnya mengambil keuntungan dari Aurora Serverless v1 fitur yang diskalakan ke nol ACUs setelah periode tidak aktif, Anda dapat meningkatkan ke Aurora Serverless v2 dan gunakan fitur jeda otomatis yang sesuai.

Manfaat penghematan biaya dari fitur jeda otomatis mirip dengan menggunakan fitur stop/start cluster. Jeda otomatis untuk Aurora Serverless v2 memiliki manfaat tambahan dari resume yang lebih cepat daripada memulai cluster yang dihentikan, dan mengotomatiskan proses menentukan kapan harus menjeda dan melanjutkan setiap instans DB.

Fitur jeda otomatis juga memberikan perincian tambahan dalam mengendalikan biaya untuk sumber daya komputasi dalam cluster Anda. Anda dapat mengaktifkan beberapa instance pembaca untuk berhenti sementara instance penulis dan pembaca lain di cluster tetap aktif setiap saat. Anda melakukannya dengan menetapkan instance pembaca yang dapat menjeda secara independen dari instance lain sebagai prioritas failover dalam kisaran 2-15.

Instance penulis dan semua instance pembaca dengan prioritas failover 0 dan 1 selalu berhenti sejenak dan melanjutkan pada saat yang bersamaan. Dengan demikian, instance dalam grup ini berhenti sejenak setelah tidak satupun dari mereka memiliki koneksi untuk interval waktu yang ditentukan.

Cluster Aurora DB dapat berisi kombinasi instance DB penulis dan pembaca dan disediakan dan Aurora Serverless v2 Contoh DB. Oleh karena itu, untuk menggunakan fitur ini secara efektif, akan sangat membantu untuk memahami aspek-aspek berikut dari mekanisme jeda otomatis:

- Keadaan ketika instans DB mungkin secara otomatis berhenti.
- Ketika instance DB dapat dicegah dari jeda. Misalnya, mengaktifkan beberapa fitur Aurora atau melakukan jenis operasi tertentu di cluster dapat mencegah instance berhenti, bahkan tanpa koneksi ke instance tersebut.

- Konsekuensi untuk pemantauan dan penagihan saat instance dijeda.
- Tindakan apa yang menyebabkan instans DB melanjutkan pemrosesan.
- Bagaimana kapasitas instans DB berubah sekitar waktu jeda dan melanjutkan peristiwa.
- Cara mengontrol interval idle sebelum instance DB berhenti.
- Cara mengkodekan logika aplikasi untuk menangani periode saat instance DB melanjutkan pemrosesan.

## Prasyarat dan Batasan untuk Aurora Serverless v2 fitur jeda otomatis

Sebelum menggunakan fitur jeda otomatis, periksa versi mesin mana yang akan digunakan. Juga, periksa apakah jeda otomatis berfungsi dalam kombinasi dengan fitur Aurora lain yang ingin Anda gunakan. Anda tidak dapat mengaktifkan jeda otomatis jika Anda menggunakan versi mesin yang tidak mendukungnya. Untuk fitur yang tidak kompatibel, Anda tidak akan mendapatkan kesalahan jika Anda menggunakannya dalam kombinasi dengan jeda otomatis. Jika cluster menggunakan fitur atau pengaturan yang tidak kompatibel, Aurora Serverless v2 instance tidak akan otomatis berhenti sejenak.

- Jika Anda menggunakan Aurora PostgreSQL, mesin database harus menjalankan setidaknya versi 16.3, 15.7, 14.12, atau 13.15.
- Jika Anda menggunakan Aurora MySQL, mesin database harus menjalankan versi 3.08.0 atau lebih tinggi.
- Untuk daftar lengkap versi mesin dan AWS Wilayah tempat fitur ini tersedia, lihatDaerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2.
- Ketika sebuah Aurora Serverless v2 instance dilanjutkan, kapasitasnya mungkin lebih rendah daripada saat instance dijeda. Untuk detailnya, lihat Perbedaan perilaku jeda otomatis antara Aurora Serverless v2 and Aurora Serverless v1.

Kondisi atau pengaturan tertentu mencegah Aurora Serverless v2 contoh dari jeda otomatis. Untuk informasi selengkapnya, lihat Situasi di mana Aurora Serverless v2 tidak berhenti otomatis.

## Menghidupkan dan menonaktifkan fitur jeda otomatis

Anda dapat mengaktifkan dan menonaktifkan fitur jeda otomatis di tingkat cluster. Untuk melakukannya, Anda menggunakan prosedur yang sama seperti ketika Anda menyesuaikan

kapasitas minimum dan maksimum untuk cluster. Fitur jeda otomatis diwakili oleh kapasitas minimum 0. ACUs

Topik

- Mengaktifkan jeda otomatis untuk Aurora Serverless v2 contoh dalam sebuah cluster
- Dapat dikonfigurasi Aurora Serverless v2 interval batas waktu jeda otomatis
- Melanjutkan jeda otomatis Aurora Serverless v2 instans
- Mematikan jeda otomatis untuk Aurora Serverless v2 contoh dalam sebuah cluster

## Mengaktifkan jeda otomatis untuk Aurora Serverless v2 contoh dalam sebuah cluster

Ikuti prosedur di Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster. Untuk kapasitas minimum, pilih 0ACUs. Bila Anda memilih kapasitas minimum 0ACUs, Anda juga dapat menentukan lamanya waktu instans menjadi idle sebelum dijeda secara otomatis.

CLIContoh berikut menunjukkan bagaimana Anda dapat membuat klaster Aurora dengan fitur jeda otomatis diaktifkan dan interval jeda otomatis disetel ke sepuluh menit (600 detik).

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=4,SecondsUntilAutoPause=600 \
    --master-username myuser \
    --manage-master-user-password
```

CLIContoh berikut menunjukkan bagaimana Anda dapat mengaktifkan fitur jeda otomatis untuk klaster Aurora yang ada. Contoh ini menetapkan interval jeda otomatis menjadi satu jam (3600 detik).

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=80,SecondsUntilAutoPause=3600

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 0,
    "MaxCapacity": 80.0,
```

```
    "SecondsUntilAutoPause": 3600
}
```

## Dapat dikonfigurasi Aurora Serverless v2 interval batas waktu jeda otomatis

Interval batas waktu direpresentasikan dalam `ServerlessV2ScalingConfiguration` atribut cluster Aurora Anda. Anda dapat menentukan interval ini di AWS Management Console saat membuat atau memodifikasi cluster Aurora, jika kapasitas minimum diatur ke nol. ACUs Anda dapat menentukannya di AWS CLI dengan menggunakan `--serverless-v2-scaling-configuration` parameter saat membuat atau memodifikasi cluster Aurora. Anda dapat menentukannya di RDS API dengan menggunakan `ServerlessV2ScalingConfiguration` parameter saat membuat atau memodifikasi cluster Aurora.

Interval minimum yang dapat Anda atur adalah 300 detik (lima menit). Itu default jika Anda tidak menentukan interval. Interval maksimum yang dapat Anda atur adalah 86.400 detik (satu hari).

Misalkan Anda mematikan fitur jeda otomatis untuk klaster dengan mengubah kapasitas minimum cluster menjadi nilai bukan nol. Dalam hal ini, properti interval dihapus dari `ServerlessV2ScalingConfiguration` atribut. Tidak adanya properti tersebut memberikan konfirmasi tambahan bahwa fitur jeda otomatis dimatikan untuk cluster tersebut. Jika nanti Anda mengaktifkan kembali jeda otomatis, Anda dapat menentukan interval khusus apa pun lagi pada saat itu.

## Melanjutkan jeda otomatis Aurora Serverless v2 instans

- Saat Anda terhubung ke jeda Aurora Serverless v2 misalnya, secara otomatis melanjutkan dan menerima koneksi.
- Upaya koneksi yang tidak menyertakan kredensi yang valid masih menyebabkan instans DB dilanjutkan.
- Jika Anda terhubung melalui titik akhir penulis, Aurora melanjutkan instance DB penulis jika dijeda secara otomatis. Pada saat yang sama, Aurora melanjutkan setiap instance pembaca yang dijeda otomatis yang memiliki prioritas failover 0 atau 1, yang berarti kapasitasnya terkait dengan kapasitas instance penulis.
- Jika Anda terhubung melalui titik akhir pembaca, Aurora memilih instance pembaca secara acak. Jika instance pembaca itu dijeda, Aurora melanjutkannya. Aurora juga melanjutkan instance penulis terlebih dahulu, karena instance penulis harus selalu aktif jika ada instance pembaca yang aktif. Ketika Aurora melanjutkan contoh penulis itu, itu juga menyebabkan setiap instance pembaca di tingkat promosi failover nol dan satu dilanjutkan.

- Jika Anda mengirim permintaan ke klaster Anda melalui RDS DataAPI, Aurora melanjutkan instance penulis jika dijeda. Kemudian Aurora memproses permintaan DataAPI.
- Saat Anda mengubah nilai parameter konfigurasi dalam grup parameter cluster DB, Aurora secara otomatis melanjutkan jeda Aurora Serverless v2 instance di semua cluster yang menggunakan grup parameter cluster tersebut. Demikian pula, ketika Anda mengubah nilai parameter dalam grup parameter DB, Aurora secara otomatis melanjutkan setiap jeda Aurora Serverless v2 contoh yang menggunakan grup parameter DB itu. Perilaku resume otomatis yang sama berlaku saat Anda memodifikasi klaster untuk menetapkan grup parameter cluster yang berbeda, atau saat Anda memodifikasi instance untuk menetapkan grup parameter DB yang berbeda.
- Melakukan permintaan backtrack secara otomatis melanjutkan Aurora Serverless v2 penulis misalnya jika dijeda. Aurora memproses permintaan backtrack setelah instance penulis dilanjutkan. Anda dapat mundur ke waktu di mana Aurora Serverless v2 contoh dijeda.
- Mengambil snapshot cluster atau menghapus snapshot tidak menyebabkan apa pun Aurora Serverless v2 contoh untuk melanjutkan.
- Membuat klon Aurora menyebabkan Aurora melanjutkan instance penulis dari cluster yang sedang dikloning.
- Jika instance yang dijeda menerima sejumlah besar permintaan koneksi sebelum selesai dilanjutkan, beberapa sesi mungkin tidak dapat terhubung. Kami merekomendasikan menerapkan logika coba lagi untuk koneksi ke cluster Aurora yang memiliki beberapa Aurora Serverless v2 instance dengan jeda otomatis diaktifkan. Misalnya, Anda dapat mencoba lagi koneksi yang gagal tiga kali.
- Aurora dapat melakukan beberapa jenis pemeliharaan internal minor tanpa membangunkan instance. Namun, beberapa jenis pemeliharaan yang terjadi selama jendela pemeliharaan cluster memang mengharuskan Aurora untuk melanjutkan instance. Ketika pemeliharaan selesai, instance secara otomatis dijeda lagi jika tidak ada lagi aktivitas setelah interval yang ditentukan.

> **Note**
>
> Aurora tidak secara otomatis melanjutkan instance yang dijeda untuk pekerjaan terjadwal khusus mesin, seperti yang ada di ekstensi SQL `pg_cron` Postgre atau penjadwal acara Saya. SQL Untuk memastikan bahwa pekerjaan tersebut berjalan, memulai koneksi secara manual ke instance sebelum waktu yang dijadwalkan. Aurora tidak mengantri pekerjaan apa pun di mana waktu yang dijadwalkan terjadi saat instans DB dijeda. Pekerjaan seperti itu dilewati ketika instance dilanjutkan nanti.

- Jika cluster Aurora mengalami failover sementara Aurora Serverless v2 instance dijeda secara otomatis, Aurora dapat melanjutkan sebuah instance dan kemudian mempromosikan instance itu menjadi penulis. Hal yang sama mungkin terjadi jika satu atau lebih instance DB dihapus dari cluster saat instance dijeda. Dalam hal ini, instance menjadi penulis segera ketika dilanjutkan.
- Operasi yang mengubah properti cluster juga menyebabkan jeda otomatis Aurora Serverless v2 contoh untuk melanjutkan. Misalnya, instance yang dijeda otomatis dilanjutkan untuk operasi seperti berikut ini:
  - Mengubah rentang penskalaan cluster.
  - Memutakhirkan versi mesin cluster.
  - Menjelaskan atau mengunduh file log dari instance yang dijeda. Anda dapat memeriksa data log historis dari instance yang dijeda dengan mengaktifkan unggahan log ke CloudWatch dan menganalisis log. CloudWatch

## Mematikan jeda otomatis untuk Aurora Serverless v2 contoh dalam sebuah cluster

Ikuti prosedur di Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster. Untuk kapasitas minimum, pilih nilai 0,5 atau lebih besar. Saat Anda mematikan fitur jeda otomatis, interval untuk instans menjadi idle diatur ulang. Jika Anda mengaktifkan jeda otomatis lagi, Anda menentukan interval batas waktu baru.

CLIContoh berikut menunjukkan bagaimana Anda dapat mematikan fitur jeda otomatis untuk klaster Aurora yang ada. `describe-db-clusters`Output menunjukkan bahwa `SecondsUntilAutoPause` atribut dihapus ketika kapasitas minimum diatur ke nilai bukan nol.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=2,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 2,
    "MaxCapacity": 80.0
}
```

# Bagaimana Aurora Serverless v2 fitur jeda otomatis berfungsi

Anda dapat menggunakan informasi berikut untuk merencanakan penggunaan fitur jeda otomatis. Memahami keadaan di mana instance berhenti, melanjutkan, atau tetap aktif dapat membantu Anda menyeimbangkan pengorbanan antara ketersediaan, daya tanggap, dan penghematan biaya.

Topik


- Apa yang terjadi ketika Aurora Serverless v2 contoh jeda
- Apa yang terjadi ketika dijeda otomatis Aurora Serverless v2 contoh dilanjutkan
- Cara kerja penagihan instans untuk dijeda otomatis Aurora Serverless v2 kluster
- Situasi di mana Aurora Serverless v2 tidak berhenti otomatis
- Cara kerja jeda otomatis dengan fitur berhenti/mulai cluster
- Cara kerja pemeliharaan dan peningkatan untuk dijeda otomatis Aurora Serverless v2 kluster
- Perbedaan perilaku jeda otomatis antara Aurora Serverless v2 and Aurora Serverless v1


## Apa yang terjadi ketika Aurora Serverless v2 contoh jeda

Ketika sebuah Aurora Serverless v2 Instans DB berhenti setelah periode tanpa koneksi:

- Aurora mulai menjeda instance setelah interval yang ditentukan berlalu tanpa koneksi ke instance, terlepas dari berapa banyak instance yang dimiliki ACUs pada saat itu.
- Mekanisme jeda tidak seketika. Sesi Aurora Serverless v2 contoh yang akan dijeda otomatis mungkin menunggu sebentar untuk mengejar semua perubahan pada penyimpanan Aurora.
- Biaya instance untuk contoh itu ditunda. `ServerlessV2Usage`Metrik memiliki nilai 0 sementara instance dijeda.
- Nilai status untuk instance tidak berubah. Status masih ditampilkan sebagai “tersedia”.
- Instance berhenti menulis ke file log database. Ini berhenti mengirim metrik ke CloudWatch, selain mendaftarkan nol persen untuk `CPUUtilization` dan`ACUUtilization`, dan nol untuk`ServerlessDatabaseCapacity`.
- Aurora memancarkan peristiwa ketika Aurora Serverless v2 Instans DB mulai berhenti, selesai berhenti, dan jika mekanisme jeda terganggu atau tidak berhasil. Untuk detail tentang acara ini, lihatPeristiwa instans DB.

## Apa yang terjadi ketika dijeda otomatis Aurora Serverless v2 contoh dilanjutkan

Ketika sebuah Aurora Serverless v2 Instans DB dilanjutkan setelah dijeda secara otomatis, ketentuan berikut berlaku:

- Setiap perubahan parameter yang ada dalam `pending-reboot` perubahan diterapkan saat instance dilanjutkan.
- Aurora memancarkan peristiwa tingkat instance saat masing-masing Aurora Serverless v2 Instans DB mulai dilanjutkan, selesai dilanjutkan, dan jika instance tidak dapat dilanjutkan karena alasan tertentu. Untuk detail tentang acara ini, lihatPeristiwa instans DB.
- Setiap koneksi yang diminta dibuat setelah instans DB selesai dilanjutkan. Karena waktu tipikal untuk melanjutkan mungkin sekitar 15 detik, kami sarankan Anda menyesuaikan pengaturan batas waktu klien menjadi lebih dari 15 detik. Misalnya, dalam pengaturan JDBC driver Anda, Anda mungkin menyesuaikan nilai untuk `sslResponseTimeout` pengaturan `connectTimeout` dan menjadi lebih dari 15 detik.

> **Note**
>
> Jika sebuah Aurora Serverless v2 Misalnya tetap dijeda lebih dari 24 jam, Aurora dapat menempatkan instance ke dalam tidur yang lebih nyenyak yang membutuhkan waktu lebih lama untuk dilanjutkan. Dalam hal ini, waktu resume bisa 30 detik atau lebih, kira-kira setara dengan melakukan reboot instance. Jika database Anda memiliki periode tidak aktif yang berlangsung lebih dari satu hari, sebaiknya atur batas waktu koneksi menjadi 30 detik atau lebih.

## Cara kerja penagihan instans untuk dijeda otomatis Aurora Serverless v2 kluster

Sementara sebuah Aurora Serverless v2 instance dijeda otomatis, muatan instancenya nol. `ServerlessV2Usage`Metrik adalah nol selama periode itu. AWS masih mengenakan biaya untuk penyimpanan Aurora dan aspek lain dari cluster yang tidak terkait dengan instans DB tertentu.

## Situasi di mana Aurora Serverless v2 tidak berhenti otomatis

- Jika nilai kapasitas minimum untuk cluster DB Anda lebih tinggi dari nolACUs, Aurora Serverless v2 instance di cluster tidak berhenti secara otomatis. Jika Anda memiliki cluster yang ada dengan Aurora Serverless v2 contoh dari sebelum fitur jeda otomatis tersedia, pengaturan kapasitas

minimum terendah adalah 0,5. ACUs Untuk menggunakan fitur jeda otomatis dengan cluster tersebut, ubah pengaturan kapasitas minimum menjadi nol. ACUs

- Jika ada koneksi yang diprakarsai pengguna terbuka untuk Aurora Serverless v2 misalnya, instance tidak akan berhenti sejenak. Instance juga tidak akan berhenti sementara aktivitas seperti patching dan upgrade sedang berlangsung. Koneksi administratif yang digunakan Aurora untuk pemeriksaan kesehatan tidak dihitung sebagai aktivitas dan tidak mencegah instance berhenti.
- Dalam klaster Aurora Postgre mengaktifkan replikasi logis, atau SQL klaster Aurora My SQL yang mengaktifkan replikasi binlog, instance penulis dan instance pembaca apa pun di tingkat promosi failover nol dan satu tidak berhenti secara otomatis. Aurora melakukan aktivitas minimal yang konstan untuk memeriksa kesehatan koneksi replikasi.

  Untuk cluster dengan replikasi diaktifkan, Anda masih dapat memiliki instance pembaca Aurora di cluster sumber yang berhenti otomatis. Untuk melakukannya, tetapkan prioritas failover mereka ke nilai selain nol atau satu. Dengan begitu, mereka dapat dijeda secara independen dari contoh penulis.
- Jika klaster Aurora Anda memiliki RDS Proxy terkait, proxy mempertahankan koneksi terbuka ke setiap instans DB di cluster. Jadi, apapun Aurora Serverless v2 instance di cluster seperti itu tidak akan berhenti secara otomatis.
- Jika cluster Anda adalah cluster utama dalam database global Aurora, Aurora tidak secara otomatis menjeda Aurora Serverless v2 contoh penulis. Itu karena tingkat aktivitas yang konstan diperlukan pada instance penulis untuk mengelola cluster lain dalam database global. Karena contoh penulis tetap aktif, apa saja Aurora Serverless v2 instance pembaca dengan prioritas failover nol atau satu juga tidak jeda otomatis.
- Aurora Serverless v2 instance di cluster sekunder dari Aurora Global Database tidak otomatis berhenti sejenak. Jika cluster DB dipromosikan ke cluster mandiri, maka fitur jeda otomatis menjadi efektif jika semua kondisi lainnya terpenuhi.
- Dalam cluster dengan ETL integrasi nol terkait dengan Redshift, instance penulis dan instance pembaca apa pun di tingkat promosi failover nol dan satu tidak berhenti secara otomatis.
- Selain aktivitas pada port database utama untuk instance, jika instance Aurora Postgre mengaktifkan fitur Babelfish, koneksi dan aktivitas apa pun pada SQL port T mencegah SQL instance dijeda otomatis.

## Cara kerja jeda otomatis dengan fitur berhenti/mulai cluster

Anda dapat menghentikan dan memulai cluster Aurora saat fitur jeda otomatis diaktifkan. Tidak masalah jika beberapa contoh dijeda. Saat Anda memulai cluster lagi, ada yang dijeda Aurora Serverless v2 instance secara otomatis dilanjutkan.

Sementara cluster Aurora dihentikan, ada yang berhenti Aurora Serverless v2 instance tidak secara otomatis dilanjutkan berdasarkan upaya untuk terhubung. Setelah cluster dimulai lagi, mekanisme biasa untuk menjeda dan melanjutkan Aurora Serverless v2 contoh berlaku.

## Cara kerja pemeliharaan dan peningkatan untuk dijeda otomatis Aurora Serverless v2 kluster

- Sementara sebuah Aurora Serverless v2 instance dijeda secara otomatis, jika Anda mencoba memutakhirkan cluster Aurora, Aurora melanjutkan instance dan meningkatkannya.
- Aurora secara berkala melanjutkan jeda otomatis Aurora Serverless v2 instance untuk melakukan pemeliharaan seperti upgrade versi minor dan perubahan properti seperti grup parameter.
- Setelah sebuah Aurora Serverless v2 instance bangun untuk operasi administratif seperti upgrade atau menerapkan pemeliharaan, Aurora menunggu setidaknya 20 menit sebelum menjeda instance itu lagi. Itu untuk memungkinkan operasi latar belakang apa pun selesai. Periode dua puluh menit juga menghindari jeda dan melanjutkan instance beberapa kali jika instance mengalami beberapa operasi administratif berturut-turut.

## Perbedaan perilaku jeda otomatis antara Aurora Serverless v2 and Aurora Serverless v1

- Waktu dimulainya kembali ditingkatkan Aurora Serverless v2 dibandingkan dengan Aurora Serverless v1. Waktu untuk melanjutkan biasanya sekitar 15 detik jika instance dijeda selama kurang dari 24 jam. Jika instance dijeda selama lebih dari 24 jam, waktu resume mungkin lebih lama.
- Cara itu Aurora Serverless v2 berlaku untuk cluster multi-AZ berarti bahwa beberapa instance DB di cluster mungkin dijeda sementara yang lain aktif. Contoh penulis dilanjutkan setiap kali pembaca berjalan, karena penulis diperlukan untuk mengoordinasikan kegiatan tertentu dalam cluster. Karena Aurora Serverless v1 tidak menggunakan instance pembaca, seluruh cluster akan selalu dijeda atau aktif.
- Ketika titik akhir pembaca secara acak memilih instance pembaca untuk terhubung, instance pembaca itu mungkin sudah aktif atau mungkin dijeda secara otomatis. Dengan demikian, waktu

untuk mengakses instance pembaca mungkin bervariasi dan lebih sulit untuk diprediksi. Cluster multi-AZ yang menggunakan Aurora Serverless v2 dan jeda otomatis karena itu mungkin mendapat manfaat dari menyiapkan titik akhir khusus untuk kasus penggunaan hanya-baca tertentu, alih-alih mengarahkan semua sesi hanya-baca ke titik akhir pembaca.

- Aurora Serverless v2 instans menjalani operasi pemeliharaan dengan frekuensi yang sama seperti instans yang disediakan. Karena Aurora secara otomatis melanjutkan instance ketika pemeliharaan seperti itu diperlukan, Anda mungkin menemukannya Aurora Serverless v2 contoh dilanjutkan lebih sering daripada Aurora Serverless v1 cluster melakukannya.

## Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk berbagai jenis cluster Aurora

Pertimbangan untuk fitur jeda otomatis bergantung pada berapa banyak instance yang ada di klaster Aurora Anda, tingkatan promosi failover dari instance pembaca, dan apakah semua instans Aurora Serverless v2 atau kombinasi dari Aurora Serverless v2 dan disediakan.

Topik


- Disarankan tata letak klaster Aurora saat menggunakan jeda otomatis
- Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk instance penulis di cluster DB
- Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk klaster Multi-AZ
- Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk cluster dengan instance yang disediakan


### Disarankan tata letak klaster Aurora saat menggunakan jeda otomatis

Ketika fitur jeda otomatis diaktifkan, Anda dapat mengatur cluster Aurora Anda untuk keseimbangan yang tepat dari ketersediaan tinggi, respons cepat, dan skalabilitas yang sesuai dengan kasus penggunaan Anda. Anda melakukannya dengan memilih kombinasi Aurora Serverless v2 instance, instance yang disediakan, dan tingkatan promosi failover untuk instans DB di klaster Anda.

Jenis konfigurasi berikut menunjukkan pengorbanan yang berbeda antara ketersediaan tinggi dan pengoptimalan biaya untuk klaster Anda:

- Untuk sistem pengembangan dan pengujian, Anda dapat mengatur cluster DB AZ tunggal dengan Aurora Serverless v2 contoh DB. Instance tunggal melayani semua permintaan baca dan tulis.

Ketika cluster tidak digunakan untuk interval waktu yang signifikan, instans DB berhenti. Pada saat itu, biaya komputasi DB untuk cluster Anda juga dijeda.

- Untuk sistem yang menjalankan aplikasi di mana ketersediaan tinggi adalah prioritas, tetapi cluster masih memiliki periode di mana ia sepenuhnya menganggur, Anda dapat mengatur cluster multi-AZ di mana instance DB penulis dan pembaca berada Aurora Serverless v2. Atur instance pembaca ke prioritas failover nol atau satu, sehingga instance penulis dan pembaca berhenti sejenak dan melanjutkan pada saat yang bersamaan. Sekarang Anda mendapatkan manfaat dari failover cepat saat cluster aktif. Ketika cluster tetap menganggur lebih lama dari ambang jeda otomatis, biaya instans DB untuk kedua instance dijeda. Ketika cluster melanjutkan pemrosesan, sesi database pertama membutuhkan waktu singkat untuk terhubung.
- Misalkan klaster Anda terus-menerus aktif dengan jumlah aktivitas minimal, dan memerlukan respons cepat untuk koneksi apa pun. Dalam hal ini, Anda dapat membuat cluster dengan lebih dari satu Aurora Serverless v2 contoh pembaca, dan memisahkan kapasitas beberapa contoh pembaca dari penulis. Tentukan prioritas failover nol atau satu untuk instance penulis dan satu instance pembaca. Tentukan prioritas yang lebih besar dari satu untuk instance pembaca lainnya. Dengan begitu, contoh pembaca di tingkat prioritas yang lebih tinggi dapat berhenti otomatis, bahkan ketika penulis dan salah satu pembaca tetap aktif.

  Dalam hal ini, Anda dapat menggunakan beberapa teknik lain untuk memastikan bahwa klaster tetap tersedia secara terus menerus sambil tetap menurunkan ke kapasitas rendah selama periode idle:

  - Anda dapat menggunakan instance yang disediakan untuk penulis dan pembaca prioritas 0 atau 1. Dengan begitu, dua instance DB tidak pernah berhenti otomatis dan selalu tersedia untuk melayani lalu lintas database dan untuk melakukan failover.
  - Anda dapat mengatur titik akhir kustom yang menyertakan Aurora Serverless v2 contoh di tingkat prioritas yang lebih tinggi, tetapi bukan penulis atau pembaca tingkat promosi 0 atau 1. Dengan begitu, Anda dapat mengarahkan sesi hanya-baca yang tidak sensitif terhadap latensi kepada pembaca yang mungkin dijeda secara otomatis. Anda dapat menghindari penggunaan titik akhir pembaca untuk permintaan tersebut, karena Aurora mungkin mengarahkan koneksi titik akhir pembaca ke instance pembaca yang selalu terjaga, atau ke salah satu instance yang dijeda otomatis. Menggunakan titik akhir kustom memungkinkan Anda mengarahkan koneksi ke berbagai grup instans berdasarkan preferensi Anda untuk respons cepat atau kapasitas penskalaan ekstra.

## Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk instance penulis di cluster DB

Ketika cluster Aurora DB hanya berisi satu instans DB, mekanisme untuk menjeda otomatis dan melanjutkan instance DB sangat mudah. Itu hanya tergantung pada aktivitas pada contoh penulis. Anda mungkin memiliki konfigurasi seperti itu untuk cluster yang digunakan untuk pengembangan dan pengujian, atau untuk menjalankan aplikasi di mana ketersediaan tinggi tidak penting. Perhatikan bahwa dalam cluster instance tunggal, Aurora mengarahkan koneksi melalui titik akhir pembaca ke instance DB penulis. Jadi, untuk cluster DB instance tunggal, mencoba terhubung ke titik akhir pembaca menyebabkan instance DB penulis yang dijeda otomatis dilanjutkan.

Faktor-faktor tambahan berikut berlaku untuk klaster Aurora dengan beberapa instans DB:

- Dalam cluster Aurora DB, instans DB penulis biasanya sering diakses. Oleh karena itu, Anda mungkin menemukan bahwa instans DB penulis tetap aktif meskipun satu atau lebih instance DB pembaca berhenti secara otomatis.
- Aktivitas tertentu pada instans DB pembaca mengharuskan instans DB penulis tersedia. Oleh karena itu, instance DB penulis tidak dapat dijeda sampai semua instance pembaca juga dijeda. Melanjutkan instance pembaca apa pun secara otomatis melanjutkan instance penulis, bahkan jika aplikasi Anda tidak mengakses instance penulis secara langsung.
- Aurora Serverless v2 instance pembaca di tingkat promosi failover nol dan satu skala untuk menjaga kapasitasnya tetap sinkron dengan instance penulis. Dengan demikian, ketika sebuah Aurora Serverless v2 contoh penulis dilanjutkan, begitu juga Aurora Serverless v2 contoh pembaca yang berada di tingkatan promosi nol atau satu.

## Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk klaster Multi-AZ

Dalam cluster Aurora DB yang berisi penulis dan satu atau lebih instance DB pembaca, beberapa Aurora Serverless v2 Instans DB mungkin dijeda sementara instans DB lainnya aktif. Instance penulis dan setiap instance pembaca dengan prioritas failover 0 dan 1 selalu berhenti sejenak dan melanjutkan semuanya pada saat yang bersamaan. Instance pembaca dengan prioritas selain 0 atau 1 dapat menjeda dan melanjutkan secara independen dari instance lainnya.

Saat Anda menggunakan fitur ini untuk cluster dengan beberapa instance pembaca, Anda dapat mengelola biaya tanpa mengorbankan ketersediaan tinggi. Contoh penulis dan satu atau dua instance pembaca lainnya dapat tetap aktif setiap saat, sementara instance pembaca tambahan dapat berhenti ketika mereka tidak diperlukan untuk menangani lalu lintas baca volume tinggi.

Apakah instance pembaca dapat berhenti secara otomatis tergantung pada apakah kapasitasnya dapat diskalakan secara independen, atau apakah kapasitasnya terkait dengan instans DB penulis. Properti penskalaan itu bergantung pada prioritas failover dari instance DB pembaca. Ketika prioritas pembaca adalah nol atau satu, kapasitas pembaca melacak kapasitas instance DB penulis. Oleh karena itu, untuk memungkinkan instans DB pembaca berhenti secara otomatis dalam rentang situasi terluas, tetapkan prioritasnya ke nilai yang lebih tinggi dari satu.

Waktu untuk melanjutkan instance pembaca mungkin sedikit lebih lama daripada melanjutkan instance penulis. Untuk respons tercepat jika instance mungkin dijeda, sambungkan ke titik akhir cluster.

### Bagaimana Aurora Serverless v2 jeda otomatis berfungsi untuk cluster dengan instance yang disediakan

Instans DB apa pun yang disediakan di cluster Aurora DB Anda tidak akan berhenti secara otomatis. Hanya Aurora Serverless v2 Instans DB, dengan kelas `db.serverless` instance, dapat menggunakan fitur jeda otomatis.

Saat klaster Aurora Anda berisi instans DB yang disediakan, apa saja Aurora Serverless v2 instance writer tidak otomatis berhenti sejenak. Itu karena persyaratan bahwa instance penulis tetap tersedia saat setiap instance pembaca aktif. Fakta bahwa Aurora Serverless v2 Penulis tetap aktif juga berarti bahwa Aurora Serverless v2 instance pembaca dengan prioritas failover 0 dan 1 tidak akan berhenti otomatis di klaster hibrida yang berisi instance yang disediakan.

## Memantau cluster Aurora yang menggunakan jeda otomatis

Untuk memantau Aurora, Anda harus sudah terbiasa dengan prosedur pemantauan Memantau Amazon Aurora dengan Amazon CloudWatch dan CloudWatch metrik yang tercantum di dalamnya. Referensi metrik untuk Aurora Ketahuilah bahwa ada beberapa pertimbangan khusus saat Anda memantau klaster Aurora yang menggunakan fitur jeda otomatis:

- Mungkin ada periode waktu ketika Aurora Serverless v2 instance tidak merekam data log dan sebagian besar metrik karena instance dijeda. Satu-satunya metrik yang dikirim ke CloudWatch saat instance dijeda adalah nol persen untuk `CPUUtilization` dan`ACUUtilization`, dan nol untuk. `ServerlessDatabaseCapacity`
- Anda dapat memeriksa apakah Aurora Serverless v2 contoh berhenti lebih atau kurang sering dari yang Anda harapkan. Untuk melakukannya, periksa seberapa sering `ServerlessDatabaseCapacity` metrik berubah dari nilai bukan nol menjadi nol, dan berapa

lama metrik tetap nol. Jika instans tidak berhenti selama yang Anda harapkan, Anda tidak menghemat biaya sebanyak yang Anda bisa. Jika instans berhenti sejenak dan dilanjutkan lebih sering daripada yang Anda inginkan, klaster Anda mungkin memiliki latensi yang tidak perlu saat menanggapi permintaan koneksi. Untuk informasi tentang faktor-faktor yang mempengaruhi apakah dan seberapa sering Aurora Serverless v2 instance dapat menjeda, melihat, Prasyarat dan Batasan untuk Aurora Serverless v2 fitur jeda otomatisSituasi di mana Aurora Serverless v2 tidak berhenti otomatis, dan. Pemecahan masalah untuk Aurora Serverless v2 jeda otomatis

- Anda juga dapat memeriksa file log yang mencatat jeda otomatis dan melanjutkan operasi Aurora Serverless v2 contoh. Jika instance tidak berhenti setelah interval batas waktu berakhir, file log ini juga menyertakan alasan mengapa jeda otomatis tidak terjadi. Untuk informasi selengkapnya, lihat Pemantauan Aurora Serverless v2 jeda dan lanjutkan aktivitas.

Topik

- Memeriksa apakah Aurora Serverless v2 contoh dijeda
- Acara untuk operasi jeda otomatis dan lanjutkan otomatis
- Cara kerja jeda otomatis dengan Performance Insights dan Enhanced Monitoring
- Bagaimana Aurora Serverless v2 fitur jeda otomatis berinteraksi dengan metrik Aurora

## Memeriksa apakah Aurora Serverless v2 contoh dijeda

Untuk menentukan apakah Aurora Serverless v2 instance dalam keadaan jeda, Anda dapat mengamati `ACUUtilization` metrik untuk instance tersebut. Metrik itu memiliki nilai nol sementara instance dijeda.

Sementara sebuah Aurora Serverless v2 instance dijeda, nilai statusnya masih terdaftar sebagai Tersedia. Hal yang sama berlaku saat dijeda Aurora Serverless v2 contoh sedang dalam proses melanjutkan. Itu karena Anda dapat berhasil terhubung ke instance seperti itu, bahkan jika koneksi mengalami sedikit penundaan.

Setiap metrik yang terkait dengan ketersediaan instance Aurora mempertimbangkan periode saat instance dijeda sebagai waktu instans tersedia.

## Acara untuk operasi jeda otomatis dan lanjutkan otomatis

Aurora memancarkan acara untuk Aurora Serverless v2 contoh ketika operasi jeda otomatis dan lanjutkan otomatis mulai, selesai, atau dibatalkan. Peristiwa yang terkait dengan fitur jeda otomatis

telah RDS-EVENT-0370 selesai. RDS-EVENT-0374 Untuk detail tentang acara ini, lihatPeristiwa instans DB.

## Cara kerja jeda otomatis dengan Performance Insights dan Enhanced Monitoring

Sementara sebuah Aurora Serverless v2 Instance dijeda, Aurora tidak mengumpulkan informasi pemantauan untuk instance tersebut baik melalui Performance Insights atau Enhanced Monitoring. Ketika instance dilanjutkan, mungkin ada penundaan singkat sebelum Aurora melanjutkan pengumpulan informasi pemantauan tersebut.

## Bagaimana Aurora Serverless v2 fitur jeda otomatis berinteraksi dengan metrik Aurora

Sementara sebuah Aurora Serverless v2 instance dijeda, tidak memancarkan sebagian besar CloudWatch metrik atau menulis informasi apa pun ke log database-nya. Satu-satunya metrik yang dikirim ke CloudWatch saat instance dijeda adalah nol persen untuk CPUUtilization danACUUtilization, dan nol untuk. ServerlessDatabaseCapacity

CloudWatch Kapan statistik komputasi terkait dengan ketersediaan instance atau cluster dan uptime, Aurora Serverless v2 contoh dianggap tersedia selama waktu mereka dijeda.

Saat Anda memulai RDS API tindakan AWS CLI atau untuk mendeskripsikan atau mengunduh log untuk dijeda Aurora Serverless v2 misalnya, instance dilanjutkan secara otomatis untuk membuat informasi log tersedia.

Contoh CloudWatch metrik

AWS CLI Contoh berikut menunjukkan bagaimana Anda dapat mengamati kapasitas instance yang berubah dari waktu ke waktu. Selama periode waktu, instance otomatis berhenti dan kemudian dilanjutkan. Sementara itu dijeda, ServerlessDatabaseCapacity metrik melaporkan nilai nol. Untuk menentukan apakah instance dijeda pada titik mana pun selama periode waktu, kami memeriksa apakah kapasitas minimum selama periode waktu itu adalah nol.

Contoh Linux berikut mewakili Aurora Serverless v2 contoh yang telah dijeda secara otomatis selama beberapa waktu. Kami mengambil sampel ServerlessDatabaseCapacity setiap menit, selama tiga menit. ACUNilai minimum 0,0 menegaskan bahwa instance dijeda di beberapa titik selama setiap menit.

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '3 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
```

```
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:11:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:12:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
```

Selanjutnya, kami mencoba membuat koneksi ke jeda Aurora Serverless v2 contoh. Dalam contoh ini, kami sengaja menggunakan kata sandi yang salah sehingga upaya koneksi tidak berhasil. Meskipun gagal, upaya koneksi menyebabkan Aurora melanjutkan instance yang dijeda.

```
$ mysql -h my_cluster_endpoint.rds.amazonaws.com -u admin -pwrong-password

ERROR 1045 (28000): Access denied for user 'admin'@'ip_address' (using password: YES)
```

Contoh Linux berikut menunjukkan bahwa instance yang dijeda dilanjutkan, tetap menganggur selama kurang lebih lima menit, dan kemudian kembali ke keadaan jeda. Instans dilanjutkan pada kapasitas 2.0. ACUs Kemudian ditingkatkan sedikit, misalnya untuk melakukan beberapa pembersihan internal. Karena tidak menerima upaya koneksi pengguna dalam periode batas waktu lima menit, itu berubah dari 4.0 ACUs langsung ke status jeda.

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '8 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
DATAPOINTS 2.0 2024-08-02T22:14:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:15:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:16:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:17:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:18:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:19:00+00:00 None
```

```
DATAPOINTS 0.0 2024-08-02T22:20:00+00:00 None
```

Jika laporan dimaksudkan untuk menunjukkan seberapa cepat instans ditingkatkan untuk menangani beban kerja, kami mungkin menentukan `Maximum` statistik alih-alih. `Minimum` Untuk perencanaan kapasitas dan estimasi biaya, kami dapat menentukan periode waktu yang lebih lama dan menggunakan `Average` statistik. Dengan begitu, kita dapat menentukan nilai kapasitas tipikal selama periode aktivitas tinggi, aktivitas rendah, dan keadaan jeda. Untuk memeriksa perilaku selama waktu yang tepat di sekitar jeda dan melanjutkan, kami dapat menentukan periode satu detik dan memeriksa interval waktu yang lebih pendek. Nilai stempel waktu dalam output, seperti`2024-08-02T22:13:00+00:00`, menunjukkan format untuk menentukan parameter yang tepat untuk opsi `--start-time` dan`--end-time`.

## Pemecahan masalah untuk Aurora Serverless v2 jeda otomatis

Jika Anda menemukan itu Aurora Serverless v2 instance tidak berhenti sesering yang Anda harapkan, periksa kemungkinan penyebab berikut:

- Konfirmasikan bahwa versi Aurora yang Anda jalankan mendukung kapasitas minimum nol. ACUs Untuk rentang kapasitas versi Aurora yang berbeda, lihat. Aurora Serverless v2 kapasitas
- Konfirmasikan bahwa nilai kapasitas minimum untuk cluster diatur ke nolACUs.
- Konfirmasikan bahwa instance yang dimaksud sebenarnya menggunakan Aurora Serverless v2 kelas instancedb.`serverless`, bukan salah satu kelas instance yang disediakan.
- Konfirmasikan bahwa klaster tidak menggunakan fitur atau pengaturan yang tidak kompatibel dariPrasyarat dan Batasan untuk Aurora Serverless v2 fitur jeda otomatis.
- Periksa file log yang ditampilkan saat Aurora Serverless v2 contoh dijeda, dilanjutkan, atau Aurora tidak dapat menjeda atau melanjutkan instance karena alasan tertentu. Untuk informasi selengkapnya, lihat Pemantauan Aurora Serverless v2 jeda dan lanjutkan aktivitas.
- Periksa apakah ada klien atau aplikasi yang menjaga koneksi tetap terbuka untuk jangka waktu yang lama. Sebaliknya, periksa apakah ada aplikasi yang menggunakan fungsi RDS Data API atau Lambda mengirim permintaan yang sering sehingga instance tidak pernah menganggur cukup lama untuk berhenti sejenak. Anda dapat memeriksa CloudWatch metrik seperti `ConnectionAttempts` dan`DatabaseConnections`. Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.
- Jika instance pembaca jarang berhenti, periksa prioritas failovernya. Jika pembaca digunakan untuk penskalaan baca dan bukan sebagai siaga jika terjadi failover, atur ke prioritas dalam kisaran 2-15.

- Jika contoh penulis jarang berhenti sejenak, periksa penggunaan instance pembaca Anda. Penulis hanya dapat berhenti sejenak jika seluruh cluster menganggur. Itu karena koneksi ke instance pembaca mana pun menyebabkan penulis melanjutkan.
- Jika aplikasi Anda menerima batas waktu selama permintaan koneksi saat Aurora Serverless v2 instance dilanjutkan, pertimbangkan untuk memperpanjang interval batas waktu yang digunakan oleh kode aplikasi Anda atau kerangka database yang mendasarinya. Batas waktu koneksi yang lebih lama mengurangi kemungkinan koneksi gagal saat Aurora Serverless v2 contoh dilanjutkan. Namun, batas waktu yang lebih lama juga dapat membuat aplikasi Anda lebih lambat untuk mendeteksi masalah ketersediaan di klaster Anda.

  Sebaliknya, pertimbangkan untuk memperpanjang interval jeda otomatis sehingga aplikasi tidak sering menemukan instance yang dijeda.

  Jika tidak ada keseimbangan logis antara perilaku jeda otomatis dan respons klaster untuk aplikasi Anda, klaster tersebut mungkin bukan kandidat yang baik untuk menggunakan fitur jeda otomatis.
- Jika Anda memperkirakan berapa lama Aurora Serverless v2 contoh akan dijeda, ketahuilah bahwa ada faktor-faktor yang membuatnya tidak praktis untuk membuat prediksi yang tepat.
  - Instans mungkin dilanjutkan secara berkala untuk melakukan pemeliharaan, peningkatan versi minor, atau menerapkan perubahan pada grup parameter.
  - Untuk cluster multi-AZ, ada situasi di mana melanjutkan satu instance menyebabkan instance lain dilanjutkan juga. Melanjutkan setiap pembaca selalu menyebabkan penulis melanjutkan. Melanjutkan penulis selalu menyebabkan pembaca dengan prioritas failover 0 dan 1 untuk melanjutkan.

  Kami merekomendasikan untuk mengukur waktu jeda aktual selama beberapa hari, dengan beban kerja yang realistis. Kemudian gunakan pengukuran tersebut untuk menetapkan garis dasar berapa proporsi waktu yang dapat Anda harapkan dari sebuah instance untuk dijeda.
- Anda mungkin menemukan bahwa operasi internal seperti SQL Pembersihan saya, penjadwal SQL acara saya, Postgre SQL autovacuum, atau SQL pekerjaan Postgre yang dijadwalkan melalui ekstensi tidak berjalan atau tidak selesai. `pg_cron` Instance tidak secara otomatis melanjutkan untuk melakukan operasi seperti itu yang tidak melibatkan koneksi pengguna ke database. Jika operasi internal tersebut sedang berlangsung ketika batas waktu jeda otomatis berakhir, tugas pemeliharaan seperti SQL pembersihan Saya dan Postgre autovacuum dibatalkan. SQL Pekerjaan terjadwal dari penjadwal SQL acara Saya atau SQL `pg_cron` ekstensi Postgre juga dibatalkan jika sedang berlangsung saat Aurora memulai operasi jeda.

Jika Anda perlu memastikan bahwa instans secara berkala terjaga untuk melakukan operasi terjadwal, Anda dapat memulai koneksi untuk melanjutkan instance sebelum waktu mulai pekerjaan. Anda juga dapat meningkatkan interval batas waktu jeda otomatis sehingga operasi seperti autovacuum dapat berjalan lebih lama setelah aktivitas pengguna selesai. Anda juga dapat menggunakan mekanisme seperti fungsi Lambda untuk melakukan operasi database sesuai jadwal, dengan cara yang secara otomatis melanjutkan instance jika perlu.

## Pertimbangan desain aplikasi untuk Aurora Serverless v2 fitur jeda otomatis

Ketika sebuah Aurora Serverless v2 Instans DB dilanjutkan setelah dijeda secara otomatis, dimulai dengan kapasitas yang relatif kecil dan ditingkatkan dari sana. Kapasitas awal ini berlaku bahkan jika instans DB memiliki kapasitas yang lebih tinggi segera sebelum dijeda secara otomatis.

Gunakan fitur ini dengan aplikasi yang dapat mentolerir interval sekitar 15 detik saat membuat koneksi. Itu menjelaskan kasus khas di mana Aurora Serverless v2 contoh dilanjutkan karena satu atau sejumlah kecil koneksi masuk. Jika instance dijeda selama lebih dari 24 jam, waktu untuk melanjutkan mungkin lebih lama.

Jika aplikasi Anda sudah menggunakan Aurora Serverless v1 dan fitur jeda otomatisnya, Anda mungkin sudah memiliki interval batas waktu seperti itu untuk upaya koneksi. Jika Anda sudah menggunakan Aurora Serverless v2 dalam kombinasi dengan fitur cluster berhenti/mulai Aurora, waktu dimulainya kembali untuk jeda otomatis Aurora Serverless v2 instance biasanya jauh lebih pendek daripada waktu untuk memulai cluster yang dihentikan.

Saat mengkodekan logika koneksi dalam aplikasi Anda, coba lagi koneksi jika upaya pertama mengembalikan kesalahan yang memiliki penyebab sementara. (Jika kesalahan disebabkan oleh kegagalan otentikasi, perbaiki kredensialnya sebelum mencoba lagi.) Kesalahan yang terjadi segera setelah dilanjutkan mungkin merupakan batas waktu, atau beberapa kesalahan yang terkait dengan batas basis data. Mencoba lagi dapat menangani masalah dalam kasus yang lebih jarang di mana Aurora Serverless v2 instance dilanjutkan karena tingginya jumlah permintaan koneksi simultan. Dalam hal ini, beberapa koneksi mungkin membutuhkan waktu lebih lama dari biasanya untuk diproses, atau mungkin melebihi batas koneksi simultan pada upaya pertama.

Selama pengembangan aplikasi dan debugging, jangan biarkan sesi klien atau alat pemrograman dengan koneksi terbuka ke database. Aurora tidak akan menjeda instance jika ada koneksi yang dimulai pengguna yang terbuka, terlepas dari apakah koneksi tidak menjalankan pernyataan atau transaksi apa pun. SQL Ketika satu Aurora Serverless v2 instance di cluster Aurora tidak dapat

berhenti sejenak, contoh lain di cluster mungkin juga dicegah untuk berhenti. Untuk informasi selengkapnya, lihat Situasi di mana Aurora Serverless v2 tidak berhenti otomatis.

Aurora memancarkan peristiwa ketika Aurora Serverless v2 Instans DB mulai dilanjutkan, selesai dilanjutkan, dan jika instance tidak dapat dilanjutkan karena alasan tertentu. Untuk detail tentang acara ini, lihatPeristiwa instans DB.

# Migrasi ke Aurora Serverless v2

Untuk mengonversi cluster DB yang ada untuk digunakan Aurora Serverless v2, Anda dapat melakukan hal berikut:

- Tingkatkan dari klaster DB Aurora terprovisi.
- Upgrade dari Aurora Serverless v1 klaster.
- Migrasi dari database lokal ke Aurora Serverless v2 klaster.

Saat klaster yang ditingkatkan menjalankan versi mesin yang sesuai seperti yang tercantum dalamPersyaratan dan batasan untuk Aurora Serverless v2, Anda dapat mulai menambahkan Aurora Serverless v2 Instans DB untuk itu. Instans DB pertama yang Anda tambahkan ke klaster yang ditingkatkan harus berupa instans DB terprovisi. Kemudian Anda dapat mengalihkan pemrosesan untuk beban kerja tulis, beban kerja baca, atau keduanya ke Aurora Serverless v2 Instans DB.

Daftar Isi


- Memutakhirkan atau mengalihkan cluster yang ada untuk digunakan Aurora Serverless v2
  - Tingkatkan jalur untuk kluster yang kompatibel dengan MySQL untuk digunakan Aurora Serverless v2
  - Tingkatkan jalur untuk kluster yang kompatibel dengan PostgreSQL untuk digunakan Aurora Serverless v2
- Beralih dari klaster yang disediakan ke Aurora Serverless v2
- Perbandingan Aurora Serverless v2 and Aurora Serverless v1
  - Perbandingan Aurora Serverless v2 and Aurora Serverless v1 persyaratan
  - Perbandingan Aurora Serverless v2 and Aurora Serverless v1 penskalaan dan ketersediaan
  - Perbandingan Aurora Serverless v2 and Aurora Serverless v1 dukungan fitur
  - Beradaptasi Aurora Serverless v1 menggunakan kasus untuk Aurora Serverless v2
- Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2

- Klaster DB yang kompatibel dengan Aurora MySQL
- Klaster DB yang kompatibel dengan Aurora PostgreSQL
- Migrasi dari database lokal ke Aurora Serverless v2

> **Note**
>
> Topik-topik ini menjelaskan cara mengonversi klaster DB yang ada. Untuk informasi tentang membuat yang baru Aurora Serverless v2 Cluster DB, lihatMembuat cluster DB yang menggunakan Aurora Serverless v2.

## Memutakhirkan atau mengalihkan cluster yang ada untuk digunakan Aurora Serverless v2

Jika klaster yang Anda sediakan memiliki versi engine yang mendukung Aurora Serverless v2, beralih ke Aurora Serverless v2 tidak memerlukan upgrade. Dalam hal ini, Anda dapat menambahkan Aurora Serverless v2 Instans DB ke cluster asli Anda. Anda dapat mengganti cluster untuk menggunakan semua Aurora Serverless v2 Instans DB. Anda juga dapat menggunakan kombinasi Aurora Serverless v2 dan menyediakan instance DB di cluster DB yang sama. Untuk versi mesin Aurora yang mendukung Aurora Serverless v2, lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Serverless v2.

Jika Anda menjalankan versi mesin yang lebih rendah yang tidak mendukung Aurora Serverless v2, Anda mengambil langkah-langkah umum ini:

1. Tingkatkan klaster.
2. Buat instans DB penulis terprovisi untuk klaster yang ditingkatkan.
3. Memodifikasi cluster untuk digunakan Aurora Serverless v2 Instans DB.

> **Important**
>
> Saat Anda melakukan upgrade versi utama ke Aurora Serverless v2-versi yang kompatibel dengan menggunakan snapshot restore atau kloning, instans DB pertama yang Anda tambahkan ke cluster baru harus berupa instance DB yang disediakan. Penambahan ini akan memulai tahap akhir proses peningkatan.

Sampai tahap akhir itu terjadi, cluster tidak memiliki infrastruktur yang diperlukan untuk Aurora Serverless v2 dukungan. Dengan demikian, klaster yang ditingkatkan ini selalu dimulai dengan instans DB penulis terprovisi. Kemudian Anda dapat mengonversi atau gagal melalui instance DB yang disediakan ke Aurora Serverless v2 satu.

Upgrade dari Aurora Serverless v1 kepada Aurora Serverless v2 melibatkan pembuatan cluster yang disediakan sebagai langkah perantara. Kemudian, Anda melakukan langkah peningkatan yang sama seperti ketika Anda memulai dengan klaster terprovisi.

## Tingkatkan jalur untuk kluster yang kompatibel dengan MySQL untuk digunakan Aurora Serverless v2

Jika klaster asli Anda menjalankan Aurora MySQL, pilih prosedur yang sesuai tergantung pada versi mesin dan mode mesin klaster Anda.

| Jika klaster Aurora MySQL asli Anda adalah yang ini | Lakukan ini untuk beralih ke Aurora Serverless v2 |
| --- | --- |
| Klaster terprovisi yang menjalankan Aurora MySQL versi 3, kompatibel dengan MySQL 8.0 | Ini adalah tahap akhir untuk semua konversi dari klaster Aurora MySQL yang ada.<br><br>Jika perlu, lakukan peningkatan versi minor ke versi 3.02.0 atau lebih tinggi. Gunakan instans DB terprovisi untuk instans DB penulis. Tambahkan satu Aurora Serverless v2 pembaca contoh DB. Lakukan failover untuk menjadikan instans DB ini sebagai instans DB penulis.<br><br>(Opsional) Konversi instance DB lain yang disediakan di cluster menjadi Aurora Serverless v2. Atau tambahkan baru Aurora Serverless v2 Instans DB dan hapus instans DB yang disediakan. |

| Jika klaster Aurora MySQL asli Anda adalah yang ini | Lakukan ini untuk beralih ke Aurora Serverless v2 |
| --- | --- |
| | Untuk prosedur dan contoh lengkap, lihat Beralih dari klaster yang disediakan ke Aurora Serverless v2. |
| Klaster terprovisi yang menjalankan Aurora MySQL versi 2, kompatibel dengan MySQL 5.7 | Lakukan peningkatan versi mayor ke Aurora MySQL versi 3.02.0 atau lebih tinggi. Kemudian ikuti prosedur untuk Aurora MySQL versi 3 untuk mengganti cluster yang akan digunakan Aurora Serverless v2 Instans DB. |
| Aurora Serverless v1 cluster yang menjalank an Aurora MySQL versi 2, kompatibel dengan MySQL 5.7 | Untuk membantu merencanakan konversi Anda dari Aurora Serverless v1, konsultasikan Perbandingan Aurora Serverless v2 and Aurora Serverless v1 dulu.<br><br>Kemudian, ikuti prosedur dalam Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2. |

## Tingkatkan jalur untuk kluster yang kompatibel dengan PostgreSQL untuk digunakan Aurora Serverless v2

Jika klaster asli Anda menjalankan Aurora PostgreSQL, pilih prosedur yang sesuai tergantung pada versi mesin dan mode mesin klaster Anda.

| Jika klaster Aurora PostgreSQL asli Anda adalah yang ini | Lakukan ini untuk beralih ke Aurora Serverless v2 |
| --- | --- |
| Klaster terprovisi yang menjalankan Aurora PostgreSQL versi 13 | Ini adalah tahap akhir untuk semua konversi dari klaster Aurora PostgreSQL yang ada.<br><br>Jika perlu, lakukan peningkatan versi minor ke versi 13.6 atau lebih tinggi. Tambahkan satu instans DB terprovisi untuk instans DB penulis. Tambahkan satu Aurora Serverless v2 |

| Jika klaster Aurora PostgreSQL asli Anda adalah yang ini | Lakukan ini untuk beralih ke Aurora Serverless v2 |
| --- | --- |
| | pembaca contoh DB. Lakukan failover untuk membuatnya Aurora Serverless v2 contoh contoh penulis DB.<br><br>(Opsional) Konversi instance DB lain yang disediakan di cluster menjadi Aurora Serverles s v2. Atau tambahkan baru Aurora Serverles s v2 Instans DB dan hapus instans DB yang disediakan.<br><br>Untuk prosedur dan contoh lengkap, lihat Beralih dari klaster yang disediakan ke Aurora Serverless v2. |
| Cluster yang disediakan menjalankan Aurora PostgreSQL versi 11 atau 12 | Lakukan peningkatan versi mayor ke Aurora PostgreSQL versi 13.6 atau lebih tinggi. Kemudian ikuti prosedur untuk Aurora PostgreSQL versi 13 untuk mengganti cluster yang akan digunakan Aurora Serverless v2 Instans DB. |
| Aurora Serverless v1 cluster yang menjalankan Aurora PostgreSQL versi 11 atau 13 | Untuk membantu merencanakan konversi Anda dari Aurora Serverless v1, konsultasikan Perbandingan Aurora Serverless v2 and Aurora Serverless v1 dulu.<br><br>Kemudian, ikuti prosedur dalam Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2. |

## Beralih dari klaster yang disediakan ke Aurora Serverless v2

Untuk mengganti klaster yang disediakan untuk digunakan Aurora Serverless v2, ikuti langkah-langkah ini:

1. Periksa apakah cluster yang disediakan perlu ditingkatkan untuk digunakan Aurora Serverless v2 Instans DB. Untuk versi Aurora yang kompatibel dengan Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2.

   Jika cluster yang disediakan menjalankan versi mesin yang tidak tersedia untuk Aurora Serverless v2, tingkatkan versi mesin cluster:

   - Jika Anda memiliki klaster terprovisi yang kompatibel dengan MySQL 5.7, ikuti petunjuk peningkatan untuk Aurora MySQL versi 3. Gunakan prosedur dalam Cara melakukan peningkatan di tempat.
   - Jika Anda memiliki klaster terprovisi yang kompatibel dengan PostgreSQL dan menjalankan PostgreSQL versi 11 atau 12, ikuti petunjuk peningkatan untuk Aurora PostgreSQL versi 13. Gunakan prosedur dalam Melakukan upgrade versi utama.
2. Konfigurasikan properti cluster lainnya agar sesuai dengan Aurora Serverless v2 persyaratan dariPersyaratan dan batasan untuk Aurora Serverless v2.
3. Konfigurasikan konfigurasi penskalaan untuk klaster. Ikuti prosedur dalam Mengatur Aurora Serverless v2 rentang kapasitas untuk sebuah cluster.
4. Tambahkan satu atau lebih Aurora Serverless v2 Instans DB ke cluster. Ikuti prosedur umum dalam Menambahkan Replika Aurora ke klaster DB. Untuk setiap instans DB baru, tentukan nama kelas instans DB khusus Tanpa Server di AWS Management Console, atau `db.serverless` di API RDS AWS CLI Amazon atau Amazon.

   Dalam beberapa kasus, Anda mungkin sudah memiliki satu atau beberapa instans DB pembaca terprovisi di klaster. Jika demikian, Anda dapat mengonversi salah satu pembaca menjadi Aurora Serverless v2 Instans DB alih-alih membuat instance DB baru. Untuk melakukannya, ikuti prosedurnya di Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2.
5. Lakukan operasi failover untuk membuat salah satu Aurora Serverless v2 DB membuat instance DB penulis untuk cluster.
6. (Opsional) Konversi instans DB apa pun yang disediakan menjadi Aurora Serverless v2, atau hapus mereka dari cluster. Ikuti prosedur umum dalam Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2 atau Menghapus instans dari klaster DB Aurora.

   > **Tip**
   >
   > Menghapus instans DB terprovisi tidaklah wajib. Anda dapat mengatur cluster yang berisi keduanya Aurora Serverless v2 dan instans DB yang disediakan. Namun, sampai Anda

terbiasa dengan kinerja dan karakteristik penskalaan Aurora Serverless v2 Instans DB, kami menyarankan Anda mengonfigurasi cluster Anda dengan instans DB semua jenis yang sama.

AWS CLI Contoh berikut menunjukkan proses peralihan menggunakan klaster yang disediakan yang menjalankan Aurora MySQL versi 3.02.0. Klasternya bernama `mysql-80`. Klaster ini dimulai dengan dua instans DB terprovisi bernama `provisioned-instance-1` dan `provisioned-instance-2`, satu penulis dan satu pembaca. Keduanya menggunakan kelas instans DB `db.r6g.large`.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

Kita membuat tabel dengan beberapa data. Dengan demikian, kita dapat mengonfirmasi bahwa data ini dan operasi klaster sama sebelum dan setelah switchover.

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
Query OK, 1 row affected (0.02 sec)
```

Pertama, kita menambahkan rentang kapasitas ke klaster ini. Jika tidak, kami mendapatkan kesalahan saat menambahkan Aurora Serverless v2 Instans DB ke cluster. Jika kita menggunakan AWS Management Console untuk prosedur ini, langkah itu otomatis ketika kita menambahkan yang pertama Aurora Serverless v2 contoh DB.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
```

```
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

Kami membuat dua Aurora Serverless v2 pembaca untuk menggantikan instans DB asli. Kita melakukannya dengan menentukan kelas instans DB `db.serverless` untuk instans DB baru.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
```

```
  "DBInstanceStatus": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

Kami melakukan failover untuk membuat salah satu Aurora Serverless v2 DB membuat instance penulis baru untuk cluster.

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

Perlu beberapa detik agar perubahan tersebut diterapkan. Pada saat itu, kita memiliki Aurora Serverless v2 Penulis dan seorang Aurora Serverless v2 pembaca. Jadi, kita tidak memerlukan instans DB asli terprovisi mana pun.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

Langkah terakhir dalam prosedur switchover adalah menghapus kedua instans DB terprovisi.

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-1",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

Sebagai pemeriksaan terakhir, kami mengonfirmasi bahwa tabel asli dapat diakses dan dapat ditulis dari Aurora Serverless v2 penulis contoh DB.

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
```

```
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
+--------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

Kami juga terhubung ke Aurora Serverless v2 pembaca DB instance dan konfirmasikan bahwa data yang baru ditulis juga tersedia di sana.

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

## Perbandingan Aurora Serverless v2 and Aurora Serverless v1

Jika Anda sudah menggunakan Aurora Serverless v1, Anda dapat mempelajari perbedaan utama antara Aurora Serverless v1 and Aurora Serverless v2. Perbedaan arsitektur, seperti dukungan untuk instans DB pembaca, membuka jenis kasus penggunaan baru.

Anda dapat menggunakan tabel berikut untuk membantu memahami perbedaan paling penting antara Aurora Serverless v2 and Aurora Serverless v1.

Topik

- Perbandingan Aurora Serverless v2 and Aurora Serverless v1 persyaratan

- Perbandingan Aurora Serverless v2 and Aurora Serverless v1 penskalaan dan ketersediaan
- Perbandingan Aurora Serverless v2 and Aurora Serverless v1 dukungan fitur
- Beradaptasi Aurora Serverless v1 menggunakan kasus untuk Aurora Serverless v2

## Perbandingan Aurora Serverless v2 and Aurora Serverless v1 persyaratan

Tabel berikut merangkum persyaratan yang berbeda untuk menjalankan database Anda menggunakan Aurora Serverless v2 atau Aurora Serverless v1. Aurora Serverless v2 menawarkan versi yang lebih tinggi dari mesin Aurora MySQL dan Aurora PostgreSQL DB daripada Aurora Serverless v1 tidak.

| Fitur | Aurora Serverless v2 kebutuhan | Aurora Serverless v1 kebutuhan |
|---|---|---|
| Mesin DB | Aurora MySQL, Aurora PostgreSQL | Aurora MySQL, Aurora PostgreSQL |
| Versi Aurora MySQL yang didukung | Lihat Aurora Serverless v2 dengan Aurora MySQL. | Lihat Aurora Serverless v1 dengan Aurora MySQL. |
| Versi Aurora PostgreSQL yang didukung | Lihat Aurora Serverless v2 dengan Aurora PostgreSQL. | Lihat Aurora Serverless v1 dengan Aurora PostgreSQL. |
| Meningkatkan klaster DB | Demikian pula dengan klaster DB terprovisi, Anda dapat melakukan peningkat an secara manual tanpa menunggu Aurora meningkat kan klaster DB untuk Anda. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.<br><br>**Note**<br>Untuk melakukan upgrade versi utama | Peningkatan versi minor diterapkan secara otomatis saat tersedia. Untuk informasi selengkapnya, lihat Aurora Serverless v1 dan versi mesin basis data Aurora.<br><br>Anda dapat melakukan peningkatan versi mayor secara manual. Untuk informasi selengkapnya, lihat Memodifikasi Aurora Serverless v1 Klaster DB. |

| Fitur | Aurora Serverless v2 kebutuhan | Aurora Serverless v1 kebutuhan |
| --- | --- | --- |
| | dari 13.x ke 14.x atau 15.x untuk cluster DB yang kompatibel dengan Aurora PostgreSQL, kapasitas maksimum cluster Anda harus minimal 2. ACUs | |

| Fitur | Aurora Serverless v2 kebutuhan | Aurora Serverless v1 kebutuhan |
| --- | --- | --- |
| Mengonversi dari klaster DB terprovisi | Anda dapat menggunakan metode berikut:<br>• Tambahkan satu atau lebih Aurora Serverless v2 instans DB pembaca ke cluster yang disediaka n yang ada. Untuk menggunakan Aurora Serverless v2 untuk penulis, lakukan failover ke salah satu Aurora Serverless v2 Instans DB. Untuk seluruh cluster untuk digunakan Aurora Serverless v2 Instans DB, hapus instans DB penulis yang disediaka n setelah mempromos ikan Aurora Serverless v2 Instans DB untuk penulis.<br>• Buat klaster baru dengan mesin DB dan versi mesin yang sesuai. Gunakan salah satu metode standar. Misalnya, pulihkan snapshot klaster atau buat klon dari klaster yang ada. Pilih Aurora Serverless v2 untuk beberapa atau semua instance DB di cluster baru.<br>Jika Anda membuat klaster baru melalui kloning, Anda tidak dapat meningkat | Kembalikan snapshot dari cluster yang disediakan untuk membuat yang baru Aurora Serverless v1 klaster. |

| Fitur | Aurora Serverless v2 kebutuhan | Aurora Serverless v1 kebutuhan |
| --- | --- | --- |
| | kan versi mesin secara bersamaan. Pastikan bahwa cluster asli sudah menjalankan versi mesin yang kompatibel dengan Aurora Serverless v2. | |
| Konversi dari Aurora Serverless v1 cluster | Ikuti prosedur dalam Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2. | Tidak berlaku |
| Kelas instans DB yang tersedia | Kelas instans DB khusus `db.serverless` . Di AWS Management Console, itu diberi label sebagai Tanpa Server. | Tidak berlaku. Aurora Serverless v1 menggunakan mode `serverless` mesin. |
| Port | Port apa pun yang kompatibel dengan MySQL atau PostgreSQL | Port MySQL atau PostgreSQL default saja |
| Alamat IP publik diizinkan? | Ya | Tidak |
| Cloud privat virtual (VPC) diperlukan? | Ya | Ya. Masing-masing Aurora Serverless v1 cluster menggunakan 2 antarmuka dan titik akhir Load Balancer Gateway yang dialokasikan ke VPC Anda. |

## Perbandingan Aurora Serverless v2 and Aurora Serverless v1 penskalaan dan ketersediaan

Tabel berikut merangkum perbedaan antara Aurora Serverless v2 and Aurora Serverless v1 untuk skalabilitas dan ketersediaan.

Aurora Serverless v2 penskalaan lebih responsif, lebih terperinci, dan tidak terlalu mengganggu daripada penskalaan Aurora Serverless v1. Aurora Serverless v2 dapat menskalakan keduanya dengan mengubah ukuran instans DB dan dengan menambahkan lebih banyak instance DB ke cluster DB. Hal ini juga dapat skala dengan menambahkan cluster lain Wilayah AWS ke database global Aurora. Sebaliknya, Aurora Serverless v1 hanya menskalakan dengan menambah atau mengurangi kapasitas penulis. Semua komputasi untuk Aurora Serverless v1 cluster berjalan dalam satu Availability Zone dan satu Wilayah AWS.

| Fitur penskalaan dan ketersediaan tinggi | Aurora Serverless v2 tingkah laku | Aurora Serverless v1 tingkah laku |
|---|---|---|
| Unit kapasitas Aurora minimum (ACUs) (Aurora MySQL) | 0.5 saat cluster berjalan, 0 saat cluster dijeda. | 1 saat klaster berjalan, 0 saat klaster dijeda. |
| Minimum ACUs (Aurora PostgreSQL) | 0.5 saat cluster berjalan, 0 saat cluster dijeda. | 2 saat klaster berjalan, 0 saat klaster dijeda. |
| Maksimum ACUs (Aurora MySQL) | 256 | 256 |
| Maksimum ACUs (Aurora PostgreSQL) | 256 | 384 |
| Menghentikan klaster | Anda dapat menghentikan dan memulai klaster secara manual dengan menggunakan fitur hentikan dan mulai klaster yang sama dengan klaster terprovisi. | Klaster berhenti secara otomatis setelah batas waktu habis. Klaster memerlukan beberapa waktu untuk tersedia ketika aktivitas dilanjutkan. |

| Fitur penskalaan dan ketersediaan tinggi | Aurora Serverless v2 tingkah laku | Aurora Serverless v1 tingkah laku |
|---|---|---|
| Penskalaan untuk instans DB | Skala naik dan turun dengan kenaikan minimum 0,5 ACUs. | Skala naik dan turun dengan menggandakan atau membagi dua. ACUs |
| Jumlah instans DB | Sama seperti klaster terprovisi: 1 instans DB penulis, maksimal 15 instans DB pembaca. | 1 instans DB menangani baca dan tulis. |
| Penskalaan dapat terjadi saat pernyataan SQL berjalan? | Ya. Aurora Serverless v2 tidak perlu menunggu titik tenang. | Tidak. Misalnya, penskalaan menunggu penyelesaian transaksi, tabel sementara, dan kunci tabel yang berjalan lama. |
| Instans DB pembaca diskalakan bersama dengan penulis | Opsional | Tidak berlaku |
| Penyimpanan maksimum | 128 TiB | 128 TiB |
| Cache buffer dipertahankan saat penskalaan | Ya. Cache buffer diubah ukurannya secara dinamis. | Tidak. Rewarming cache buffer dilakukan setelah penskalaan. |
| Failover | Ya, sama seperti untuk klaster terprovisi. | Upaya terbaik saja, tergantung pada ketersediaan kapasitas. Lebih lambat dari pada Aurora Serverless v2. |

| Fitur penskalaan dan ketersediaan tinggi | Aurora Serverless v2 tingkah laku | Aurora Serverless v1 tingkah laku |
|---|---|---|
| Kemampuan Multi-AZ | Ya, sama seperti untuk klaster terprovisi. Klaster Multi-AZ memerlukan instans DB pembaca di Zona Ketersediaan (AZ) kedua. Untuk klaster Multi-AZ, Aurora melakukan failover Multi-AZ jika terjadi kegagalan AZ. | Aurora Serverless v1 cluster menjalankan semua komputasi mereka dalam satu AZ. Pemulihan jika terjadi kegagalan AZ adalah upaya terbaik saja dan tergantung pada ketersediaan kapasitas. |
| Basis data global Aurora | Ya | Tidak |
| Penskalaan berdasarkan tekanan memori | Ya | Tidak |
| Penskalaan berdasarkan beban CPU | Ya | Ya |
| Penskalaan berdasarkan lalu lintas jaringan | Ya, berdasarkan overhead memori dan CPU untuk lalu lintas jaringan. Parameter `max_connections` tetap konstan untuk menghindari terputusnya koneksi saat menurunkan skala. | Ya, berdasarkan jumlah koneksi. |
| Tindakan batas waktu untuk peristiwa penskalaan | Tidak | Ya |
| Menambahkan instans DB baru ke cluster melalui AWS Auto Scaling | Tidak berlaku. Anda dapat membuat Aurora Serverless v2 instans DB pembaca di tingkatan promosi 2—15 dan membuatnya diperkecil ke kapasitas rendah. | Tidak. Instans DB pembaca tidak tersedia. |

## Perbandingan Aurora Serverless v2 and Aurora Serverless v1 dukungan fitur

Tabel berikut merangkum hal ini:

- Fitur yang tersedia di Aurora Serverless v2 tapi tidak Aurora Serverless v1
- Fitur yang bekerja secara berbeda antara Aurora Serverless v1 and Aurora Serverless v2
- Fitur yang saat ini tidak tersedia di Aurora Serverless v2

Aurora Serverless v2 menyertakan banyak fitur dari kluster yang disediakan yang tidak tersedia untuk Aurora Serverless v1.

| Fitur | Aurora Serverless v2 dukungan | Aurora Serverless v1 dukungan |
|---|---|---|
| Topologi klaster | Aurora Serverless v2 adalah properti dari instans DB individu. Sebuah cluster dapat berisi beberapa Aurora Serverless v2 Instans DB, atau kombinasi Aurora Serverless v2 dan instans DB yang disediakan. | Aurora Serverless v1 cluster tidak menggunakan gagasan instance DB. Anda tidak dapat mengubah Aurora Serverless v1 properti setelah Anda membuat cluster. |
| Parameter konfigurasi | Hampir semua parameter yang sama dapat dimodifikasi seperti pada klaster terprovisi. Untuk detailnya, lihat Menggunakan grup parameter untuk Aurora Serverless v2. | Hanya sebagian parameter yang dapat dimodifikasi. |
| Grup parameter | Grup parameter klaster dan grup parameter DB. Parameter dengan nilai `provisioned` dalam atribut `SupportedEngineModes` tersedia. Itu lebih banyak | Grup parameter klaster saja. Parameter dengan nilai `serverless` dalam atribut `SupportedEngineModes` tersedia. |

| Fitur | Aurora Serverless v2 dukungan | Aurora Serverless v1 dukungan |
| --- | --- | --- |
| | parameter daripada di Aurora Serverless v1. | |
| Enkripsi untuk volume klaster | Opsional | Wajib. Batasan Batasan klaster DB terenkripsi Amazon Aurora berlaku untuk semua Aurora Serverless v1 klaster. |
| Snapshot lintas Wilayah | Ya | Snapshot harus dienkripsi dengan kunci AWS Key Management Service ()AWS KMS Anda sendiri. |
| Cadangan otomatis dipertahankan setelah penghapusan klaster DB | Ya | Tidak |
| TLS/SSL | Ya. Dukungannya sama dengan klaster terprovisi. Untuk informasi penggunaan, lihat Menggunakan TLS/SSL dengan Aurora Serverless v2. | Ya. Ada beberapa perbedaan dari dukungan TLS untuk klaster terprovisi. Untuk informasi penggunaan, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1. |
| Kloning | Hanya dari dan ke versi mesin DB yang kompatibel dengan Aurora Serverless v2. Anda tidak dapat menggunakan kloning untuk meningkatkan dari Aurora Serverless v1 atau dari versi sebelumnya dari cluster yang disediakan. | Hanya dari dan ke versi mesin DB yang kompatibel dengan Aurora Serverless v1. |
| Integrasi dengan Amazon S3 | Ya | Ya |

| Fitur | Aurora Serverless v2 dukungan | Aurora Serverless v1 dukungan |
|---|---|---|
| Integrasi dengan AWS Secrets Manager | Ya | Tidak |
| Mengekspor snapshot klaster DB ke S3 | Ya | Tidak |
| Mengaitkan peran IAM | Ya | Tidak |
| Mengunggah log ke Amazon CloudWatch | Opsional. Anda memilih log mana yang akan diaktifkan dan log mana yang akan diunggah CloudWatch. | Semua log yang dihidupkan diunggah secara CloudWatch otomatis. |
| Data API tersedia | Ya | Ya |
| Editor kueri tersedia | Ya | Ya |
| Wawasan Performa | Ya | Tidak |
| Proksi Amazon RDS tersedia | Ya | Tidak |
| Babelfish for Aurora PostgreSQL tersedia | Ya. Didukung untuk versi Aurora PostgreSQL yang kompatibel dengan Babelfish dan Aurora Serverless v2. | Tidak |

## Beradaptasi Aurora Serverless v1 menggunakan kasus untuk Aurora Serverless v2

Tergantung pada kasus penggunaan Anda untuk Aurora Serverless v1, Anda mungkin menyesuaikan pendekatan itu untuk memanfaatkan Aurora Serverless v2 fitur sebagai berikut.

Misalkan Anda memiliki Aurora Serverless v1 cluster yang dimuat ringan dan prioritas Anda adalah menjaga ketersediaan berkelanjutan sambil meminimalkan biaya. Dengan Aurora Serverless v2, Anda dapat mengonfigurasi pengaturan ACU minimum yang lebih kecil 0,5, dibandingkan dengan minimal 1 ACU untuk Aurora Serverless v1. Anda dapat meningkatkan ketersediaan dengan membuat konfigurasi Multi-AZ, dengan instans DB pembaca juga memiliki minimal 0,5 ACUs.

Misalkan Anda memiliki Aurora Serverless v1 cluster yang Anda gunakan dalam skenario pengembangan dan pengujian. Dalam hal ini, biaya juga merupakan prioritas tinggi, tetapi klaster tidak perlu tersedia setiap saat. Aurora Serverless v2 dapat secara otomatis menjeda setiap instance saat benar-benar menganggur. Anda melakukannya dengan menentukan kapasitas minimum 0 ACUs untuk cluster, seperti yang dijelaskan dalamPenskalaan ke Nol ACUs dengan jeda otomatis dan lanjutkan Aurora Serverless v2. Anda juga dapat menghentikan cluster secara manual saat tidak diperlukan, dan memulainya saat waktunya untuk siklus pengujian atau pengembangan berikutnya.

Misalkan Anda memiliki Aurora Serverless v1 cluster dengan beban kerja yang berat. Cluster yang setara menggunakan Aurora Serverless v2 dapat menskalakan dengan lebih granularitas. Misalnya, Aurora Serverless v1 skala dengan menggandakan kapasitas, misalnya dari 64 menjadi 128 ACUs. Sebaliknya, Anda Aurora Serverless v2 Instans DB dapat menskalakan peningkatan 0,5-ACU.

Misalkan beban kerja Anda membutuhkan kapasitas total yang lebih tinggi daripada yang tersedia di Aurora Serverless v1. Anda dapat menggunakan beberapa Aurora Serverless v2 instans DB pembaca untuk menurunkan bagian beban kerja yang intensif baca dari instans DB penulis. Anda juga dapat membagi beban kerja yang sarat baca di antara beberapa instans DB pembaca.

Untuk beban kerja sarat tulis, Anda dapat mengonfigurasi klaster dengan instans DB besar terprovisi sebagai penulis. Anda dapat melakukannya bersama satu atau lebih Aurora Serverless v2 instans DB pembaca.

## Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Proses upgrade cluster DB dari Aurora Serverless v1 kepada Aurora Serverless v2 memiliki beberapa langkah. Itu karena Anda tidak dapat mengonversi langsung dari Aurora Serverless v1 kepada Aurora Serverless v2. Selalu ada langkah perantara yang melibatkan konversi Aurora Serverless v1 Cluster DB ke cluster yang disediakan.

## Klaster DB yang kompatibel dengan Aurora MySQL

Anda dapat mengonversi Aurora Serverless v1 Cluster DB ke cluster DB yang disediakan, lalu gunakan penerapan biru/hijau untuk memutakhirkannya dan mengubahnya menjadi Aurora Serverless v2 klaster DB. Kami merekomendasikan prosedur ini untuk lingkungan produksi. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

Untuk menggunakan penerapan biru/hijau untuk memutakhirkan Aurora Serverless v1 cluster yang menjalankan Aurora MySQL versi 2 (MySQL 5.7 — kompatibel)

1. Konversikan Aurora Serverless v1 Cluster DB ke cluster Aurora MySQL versi 2 yang disediakan. Ikuti prosedur di Konversi dari Aurora Serverless v1 untuk disediakan.
2. Buat penyebaran biru/hijau. Ikuti prosedur di Membuat penyebaran biru/hijau di Amazon RDS Aurora.
3. Pilih versi Aurora MySQL untuk cluster hijau yang kompatibel dengan Aurora Serverless v2, misalnya 3.04.1.

   Untuk versi yang kompatibel, lihat Aurora Serverless v2 dengan Aurora MySQL.
4. Ubah instance DB penulis dari cluster hijau untuk menggunakan kelas instans DB Serverless v2 (db.serverless).

   Lihat perinciannya di Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2.
5. Saat Anda ditingkatkan Aurora Serverless v2 Cluster DB tersedia, beralih dari cluster biru ke cluster hijau.

## Klaster DB yang kompatibel dengan Aurora PostgreSQL

Anda dapat mengonversi Aurora Serverless v1 Cluster DB ke cluster DB yang disediakan, lalu gunakan penerapan biru/hijau untuk memutakhirkannya dan mengubahnya menjadi Aurora Serverless v2 klaster DB. Kami merekomendasikan prosedur ini untuk lingkungan produksi. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Blue/Green Deployment untuk pembaruan database.

Untuk menggunakan penerapan biru/hijau untuk memutakhirkan Aurora Serverless v1 cluster yang menjalankan Aurora PostgreSQL versi 11

1. Konversikan Aurora Serverless v1 Cluster DB ke cluster Aurora PostgreSQL yang disediakan. Ikuti prosedur di Konversi dari Aurora Serverless v1 untuk disediakan.
2. Buat penyebaran biru/hijau. Ikuti prosedur di Membuat penyebaran biru/hijau di Amazon RDS Aurora.
3. Pilih versi Aurora PostgreSQL untuk cluster hijau yang kompatibel dengan Aurora Serverless v2, misalnya 15.3.

   Untuk versi yang kompatibel, lihat Aurora Serverless v2 dengan Aurora PostgreSQL.
4. Ubah instance DB penulis dari cluster hijau untuk menggunakan kelas instans DB Serverless v2 (db.serverless).

   Lihat perinciannya di Mengonversi penulis atau pembaca yang disediakan menjadi Aurora Serverless v2.
5. Saat Anda ditingkatkan Aurora Serverless v2 Cluster DB tersedia, beralih dari cluster biru ke cluster hijau.

Anda juga dapat meng-upgrade Aurora Serverless v1 Cluster DB langsung dari Aurora PostgreSQL versi 11 ke versi 13, mengubahnya menjadi cluster DB yang disediakan, dan kemudian mengonversi cluster yang disediakan menjadi Aurora Serverless v2 klaster DB.

Untuk meng-upgrade, kemudian mengkonversi Aurora Serverless v1 cluster yang menjalankan Aurora PostgreSQL versi 11

1. Konversikan Aurora Serverless v1 Cluster DB ke cluster Aurora PostgreSQL yang disediakan. Ikuti prosedur di Konversi dari Aurora Serverless v1 untuk disediakan.
2. Tingkatkan Aurora Serverless v1 cluster ke Aurora PostgreSQL versi 13 versi yang kompatibel dengan Aurora Serverless v2, misalnya, 13.12. Ikuti prosedur dalam Meningkatkan versi mayor.

   Untuk versi yang kompatibel, lihat Aurora Serverless v2 dengan Aurora PostgreSQL.
3. Menambahkan Aurora Serverless v2 pembaca instance DB ke cluster. Untuk informasi selengkapnya, lihat Menambahkan Aurora Serverless v2 pembaca.
4. Gagal ke Aurora Serverless v2 contoh DB:

   a. Pilih instance DB penulis dari cluster DB.

b. Untuk Tindakan, pilih Failover.

c. Pada halaman konfirmasi, pilih Failover.

Untuk Aurora Serverless v1 Cluster DB yang menjalankan Aurora PostgreSQL versi 13, Anda mengonversi Aurora Serverless v1 cluster ke cluster DB yang disediakan, dan kemudian mengonversi cluster yang disediakan menjadi Aurora Serverless v2 klaster DB.

Untuk meng-upgrade Aurora Serverless v1 cluster yang menjalankan Aurora PostgreSQL versi 13

1. Konversikan Aurora Serverless v1 Cluster DB ke cluster Aurora PostgreSQL yang disediakan. Ikuti prosedur di Konversi dari Aurora Serverless v1 untuk disediakan.
2. Menambahkan Aurora Serverless v2 pembaca instance DB ke cluster. Untuk informasi selengkapnya, lihat Menambahkan Aurora Serverless v2 pembaca.
3. Gagal ke Aurora Serverless v2 contoh DB:

   a. Pilih instance DB penulis dari cluster DB.

   b. Untuk Tindakan, pilih Failover.

   c. Pada halaman konfirmasi, pilih Failover.
4. Hapus instance pembaca.

## Migrasi dari database lokal ke Aurora Serverless v2

Anda dapat memigrasikan database lokal Anda ke Aurora Serverless v2, sama seperti Aurora MySQL dan Aurora PostgreSQL yang disediakan.

- Untuk basis data MySQL, Anda dapat menggunakan perintah `mysqldump`. Untuk informasi selengkapnya, lihat Mengimpor data ke instans DB MySQL atau MariaDB dengan waktu henti yang lebih singkat.
- Untuk basis data PostgreSQL, Anda dapat menggunakan perintah `pg_dump` dan `pg_restore`. Untuk informasi selengkapnya, lihat postingan blog Praktik terbaik untuk memigrasi basis data PostgreSQL ke Amazon RDS dan Amazon Aurora.

# Menggunakan Amazon Aurora Serverless v1

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi kluster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Amazon Aurora Serverless v1 (Amazon Aurora Serverless versi 1) adalah konfigurasi penskalaan otomatis sesuai permintaan untuk Amazon Aurora. Sebuah Aurora Serverless v1 DB klaster adalah klaster DB yang meningkatkan dan menurunkan kapasitas komputasi berdasarkan kebutuhan aplikasi Anda. Ini kontras dengan cluster DB yang disediakan Aurora, di mana Anda mengelola kapasitas secara manual. Aurora Serverless v1 menyediakan opsi yang relatif sederhana dan hemat biaya untuk beban kerja yang jarang, terputus-putus, atau tidak dapat diprediksi. Ini hemat biaya karena secara otomatis dimulai, menskalakan kapasitas komputasi agar sesuai dengan penggunaan aplikasi Anda, dan dimatikan saat tidak digunakan.

Untuk mempelajari lebih lanjut tentang harga, lihat Harga Tanpa Server di bawah Edisi Kompatibel dengan MySQL atau Edisi yang kompatibel dengan PostgreSQL di Amazon Aurora pricing halaman.

Aurora Serverless v1 klaster memiliki jenis volume penyimpanan berkapasitas tinggi, terdistribusi, dan sangat tersedia yang sama dengan yang digunakan oleh klaster DB yang disediakan.

Untuk Aurora Serverless v1 cluster, volume cluster selalu dienkripsi. Anda dapat memilih kunci enkripsi, tetapi tidak dapat menonaktifkan enkripsi. Artinya, Anda dapat melakukan operasi yang sama pada Aurora Serverless v1 yang Anda bisa pada snapshot terenkripsi. Untuk informasi selengkapnya, lihat Aurora Serverless v1 dan snapshot.

Topik


- Ketersediaan wilayah dan versi untuk Aurora Serverless v1
- Keuntungan dari Aurora Serverless v1

- Kasus penggunaan untuk Aurora Serverless v1
- Pembatasan Aurora Serverless v1
- Persyaratan konfigurasi untuk Aurora Serverless v1
- Menggunakan TLS/SSL dengan Aurora Serverless v1
- Bagaimana Aurora Serverless v1 cara kerja
- Membuat Aurora Serverless v1 klaster DB.
- Memulihkan Aurora Serverless v1 klaster DB
- Memodifikasi Aurora Serverless v1 Klaster DB
- Penskalaan Aurora Serverless v1 kapasitas klaster DB secara manual
- Melihat Aurora Serverless v1 klaster DB
- Menghapus sebuah Aurora Serverless v1 Klaster DB
- Aurora Serverless v1 dan versi mesin basis data Aurora


## Ketersediaan wilayah dan versi untuk Aurora Serverless v1

Ketersediaan fitur dan dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi lebih lanjut tentang versi dan ketersediaan Wilayah dengan Aurora dan Aurora Serverless v1, lihat Aurora Serverless v1.

## Keuntungan dari Aurora Serverless v1

Aurora Serverless v1 memberikan keuntungan berikut:

- Lebih sederhana dari yang disediakan - Aurora Serverless v1 menghilangkan banyak kerumitan dalam mengelola instans dan kapasitas DB.
- Dapat diskalakan - Aurora Serverless v1 menskalakan komputasi dan kapasitas memori dengan mulus sesuai kebutuhan, tanpa gangguan pada koneksi klien.
- Hemat biaya — Saat Anda menggunakan Aurora Serverless v1, Anda hanya membayar untuk sumber daya basis data yang Anda konsumsi, setiap detik.
- Penyimpanan yang sangat tersedia - Aurora Serverless v1 menggunakan sistem penyimpanan terdistribusi dengan toleransi kesalahan yang sama dengan replikasi enam arah seperti Aurora untuk melindungi dari kehilangan data.

# Kasus penggunaan untuk Aurora Serverless v1

Aurora Serverless v1 dirancang untuk kasus penggunaan berikut:

- Aplikasi yang jarang digunakan – Anda memiliki aplikasi yang hanya digunakan selama beberapa menit beberapa kali per hari atau minggu, seperti situs blog volume rendah. Dengan Aurora Serverless v1, Anda hanya membayar sumber daya basis data yang Anda konsumsi per detik.
- Aplikasi baru – Anda men-deploy aplikasi baru dan Anda tidak yakin dengan ukuran instans yang Anda butuhkan. Dengan menggunakan Aurora Serverless v1, Anda dapat membuat titik akhir basis data dan membuat skala otomatis basis data sesuai dengan persyaratan kapasitas aplikasi Anda.
- Beban kerja variabel – Anda menjalankan aplikasi yang jarang digunakan, dengan beban puncak selama 30 menit hingga beberapa jam beberapa kali setiap hari, atau beberapa kali per tahun. Contohnya adalah aplikasi untuk sumber daya manusia, penganggaran, dan aplikasi pelaporan operasional. Dengan Aurora Serverless v1, Anda tidak perlu lagi menyediakan kapasitas puncak atau rata-rata.
- Beban kerja yang tidak dapat diprediksi – Anda menjalankan beban kerja harian yang mengalami peningkatan aktivitas secara tiba-tiba dan tidak dapat diprediksi. Contohnya adalah situs lalu lintas yang melihat lonjakan aktivitas saat hujan mulai turun. Dengan Aurora Serverless v1, kapasitas skala otomatis basis data Anda untuk memenuhi kebutuhan beban puncak aplikasi dan menurunkan skala kembali saat lonjakan aktivitas selesai.
- Basis data pengembangan dan pengujian – Developer Anda menggunakan basis data selama jam kerja, tetapi tidak memerlukannya pada malam hari atau akhir pekan. Dengan Aurora Serverless v1, basis data Anda secara otomatis mati saat tidak digunakan.
- Aplikasi multi-penyewa - Dengan Aurora Serverless v1, Anda tidak perlu mengelola kapasitas basis data satu per satu untuk setiap aplikasi dalam armada Anda. Aurora Serverless v1 mengelola kapasitas basis data individu untuk Anda.

# Pembatasan Aurora Serverless v1

Batasan berikut berlaku untuk Aurora Serverless v1:

-

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika

pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi kluster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

- Aurora Serverless v1 tidak mendukung fitur-fitur berikut:
  - Basis data global Aurora
  - Aurora Replica
  - AWS Identity and Access Management (IAM) otentikasi basis data
  - Pelacakan mundur di Aurora
  - Stream aktivitas basis data
  - Autentikasi Kerberos
  - Wawasan Performa
  - Proksi RDS
  - Melihat log di AWS Management Console
- Koneksi ke Aurora Serverless v1 Klaster DB tertutup secara otomatis jika ditahan agar terbuka lebih lama dari satu hari.
- Semua Aurora Serverless v1 Klaster DB memiliki batasan berikut:
  - Anda tidak dapat mengekspor Aurora Serverless v1 snapshot ke bucket Amazon S3
  - Anda tidak dapat menggunakan AWS Database Migration Service dan Mengubah Data Capture (CDC) dengan Aurora Serverless v1 klaster DB. Hanya cluster Aurora DB yang disediakan yang mendukung CDC sebagai sumber. AWS DMS
  - Anda tidak dapat menyimpan data ke file teks di Amazon S3 atau memuat data file teks Aurora Serverless v1 dari S3.
  - Anda tidak dapat melampirkan peran IAM ke Aurora Serverless v1 klaster DB. Namun, Anda dapat memuat data ke Aurora Serverless v1 dari Amazon S3 dengan menggunakan ekstensi `aws_s3` dengan fungsi `aws_s3.table_import_from_s3` dan parameter `credentials`. Untuk informasi selengkapnya, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL.
  - Saat menggunakan editor kueri, rahasia Secrets Manager dibuat untuk kredensial DB untuk mengakses basis data. Jika Anda menghapus kredensial dari editor kueri, rahasia terkait juga dihapus dari Secrets Manager. Anda tidak dapat memulihkan rahasia ini setelah dihapus.
- Kluster DB berbasis Aurora MySQL berjalan Aurora Serverless v1 tidak mendukung yang berikut:

- Memanggil AWS Lambda fungsi dari dalam cluster Aurora MySQL DB Anda. Namun, AWS Lambda fungsi dapat melakukan panggilan ke Aurora Serverless v1 klaster DB.
- Memulihkan snapshot dari instans DB yang bukan merupakan instans DB Aurora MySQL atau RDS for MySQL.
- Mereplikasi data menggunakan replikasi berdasarkan log biner (binlog). Batasan ini benar terlepas dari apakah cluster DB berbasis Aurora MySQL Anda Aurora Serverless v1 adalah sumber atau target replikasi. Untuk mereplikasi data menjadi Aurora Serverless v1 Cluster DB dari instans MySQL DB di luar Aurora, seperti yang berjalan di Amazon, pertimbangkan untuk menggunakan. EC2 AWS Database Migration Service Untuk informasi selengkapnya, silakan lihat Panduan Pengguna AWS Database Migration Service.
- Membuat pengguna dengan akses berbasis host (`'username'@'IP_address'`). Ini karena Aurora Serverless v1 menggunakan armada router antara klien dan host database untuk penskalaan yang mulus. Alamat IP yang Aurora Serverless Cluster DB melihat adalah host router dan bukan klien Anda. Untuk informasi selengkapnya, lihat Aurora Serverless v1 arsitektur.

  Sebagai gantinya, gunakan wildcard (`'username'@'%'`).

- Kluster DB berbasis Aurora PostgreSQL berjalan Aurora Serverless v1 memiliki batasan berikut:
  - Manajemen rencana kueri Aurora PostgreSQL (ekstensi `apg_plan_management`) tidak didukung.
  - Fitur replikasi logis yang tersedia di Amazon RDS PostgreSQL dan Aurora PostgreSQL tidak didukung.
  - Komunikasi keluar seperti yang dimungkinkan oleh ekstensi Amazon RDS for PostgreSQL tidak didukung. Misalnya, Anda tidak dapat mengakses data eksternal dengan ekstensi `postgres_fdw/dblink`. Untuk informasi selengkapnya tentang ekstensi RDS PostgreSQL, lihat PostgreSQL di Amazon RDS dalam Panduan Pengguna RDS.
  - Saat ini, kueri dan perintah SQL tertentu tidak disarankan. Hal ini termasuk kunci advisory tingkat sesi, relasi sementara, notifikasi asinkron (`LISTEN`), dan kursor dengan penahanan (`DECLARE name ... CURSOR WITH HOLD FOR query`). Selain itu, perintah `NOTIFY` akan mencegah penskalaan dan tidak disarankan.

    Untuk informasi selengkapnya, lihat Penskalaan otomatis untuk Aurora Serverless v1.

- Anda tidak dapat menyetel jendela pencadangan otomatis pilihan untuk Aurora Serverless v1 klaster DB.

- Anda dapat mengatur jendela pemeliharaan untuk Aurora Serverless v1 klaster DB. Untuk informasi selengkapnya, lihat Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.

## Persyaratan konfigurasi untuk Aurora Serverless v1

Saat Anda membuat Aurora Serverless v1 Cluster DB, perhatikan persyaratan berikut:

- Gunakan nomor port spesifik ini untuk setiap mesin DB:
  - Aurora MySQL – `3306`
  - Aurora PostgreSQL – `5432`
- Buat Anda Aurora Serverless v1 Cluster DB di cloud pribadi virtual (VPC) berdasarkan layanan Amazon VPC. Saat Anda membuat Aurora Serverless v1 Cluster DB di VPC Anda, Anda menggunakan dua (2) dari lima puluh (50) titik akhir Antarmuka dan Gateway Load Balancer yang dialokasikan ke VPC Anda. Titik akhir ini dibuat secara otomatis untuk Anda. Untuk menambah kuota Anda, Anda dapat menghubungi Dukungan. Untuk informasi selengkapnya, lihat kuota Amazon VPC.
- Anda tidak bisa memberikan Aurora Serverless v1 Klaster DB adalah alamat IP publik. Anda dapat mengakses Aurora Serverless v1 Cluster DB hanya dari dalam VPC.
- Buat subnet di Availability Zone yang berbeda untuk grup subnet DB yang Anda gunakan untuk Aurora Serverless v1 klaster DB. Dengan kata lain, Anda tidak dapat memiliki lebih dari satu subnet di Availability Zone yang sama.
- Perubahan ke grup subnet yang digunakan oleh Aurora Serverless v1 Cluster DB tidak diterapkan ke cluster.
- Anda dapat mengakses Aurora Serverless v1 Cluster DB dari AWS Lambda. Untuk melakukannya, Anda harus mengonfigurasi fungsi Lambda Anda agar berjalan dalam VPC yang sama seperti Aurora Serverless v1 klaster DB. Untuk informasi selengkapnya tentang bekerja dengan AWS Lambda, lihat Mengonfigurasi fungsi Lambda untuk mengakses sumber daya di VPC Amazon di Panduan Pengembang.AWS Lambda

## Menggunakan TLS/SSL dengan Aurora Serverless v1

Secara default, Aurora Serverless v1 menggunakan protokol Transport LayerSecurity/Secure Sockets Layer (TLS/SSL) untuk mengenkripsi komunikasi antara klien dan Aurora Serverless v1 klaster DB. Layanan ini mendukung protokol TLS/SSL versi 1.0, 1.1, dan 1.2. Anda tidak perlu mengkonfigurasi Aurora Serverless v1 Cluster DB untuk menggunakan TLS/SSL.

Namun, batasan berikut berlaku:

- Dukungan TLS/SSL untuk Aurora Serverless v1 Cluster DB saat ini tidak tersedia di Tiongkok (Beijing). Wilayah AWS
- Saat Anda membuat pengguna database untuk berbasis Aurora MySQL Aurora Serverless v1 Cluster DB, jangan gunakan `REQUIRE` klausa untuk izin SSL. Tindakan ini akan mencegah pengguna terhubung ke instans Aurora DB.
- Untuk utilitas Klien MySQL dan PostgreSQL Client, variabel sesi yang mungkin Anda gunakan di lingkungan lain tidak berpengaruh saat menggunakan TLS/SSL antara klien dan Aurora Serverless v1.
- Untuk Klien MySQL, ketika menghubungkan dengan mode `VERIFY_IDENTITY` TLS/SSL, saat ini Anda perlu menggunakan perintah `mysql` yang kompatibel dengan MySQL 8.0. Untuk informasi selengkapnya, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Tergantung pada klien yang Anda gunakan untuk terhubung Aurora Serverless v1 Cluster DB, Anda mungkin tidak perlu menentukan TLS/SSL untuk mendapatkan koneksi terenkripsi. Misalnya, untuk menggunakan PostgreSQL Client untuk terhubung ke Aurora Serverless v1 Cluster DB yang menjalankan Aurora PostgreSQL Compatible Edition, sambungkan seperti biasa.

```
psql -h endpoint -U user
```

Setelah Anda memasukkan kata sandi, Klien PostgreSQL akan menunjukkan detail koneksi, termasuk versi dan cipher TLS/SSL.

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

> **Important**
>
> Aurora Serverless v1 menggunakan Security/Secure Sockets Layer (TLS/SSL) protocol to encrypt connections by default unless SSL/TLS is disabled by the client application. The TLS/SSL koneksi Transport Layer berakhir di armada router. Komunikasi antara armada router dan Anda Aurora Serverless v1 Cluster DB terjadi dalam batas jaringan internal layanan. Anda dapat memeriksa status koneksi klien untuk memeriksa apakah koneksi ke Aurora Serverless v1 TLS/SSL dienkripsi. `pg_stat_ssl`PostgreSQL `pg_stat_activity` dan

tabel dan `ssl_is_used` fungsinya tidak menampilkan status TLS/SSL untuk komunikasi antara aplikasi klien dan Aurora Serverless v1. Demikian pula, status TLS/SSL tidak dapat diturunkan dari pernyataan MySQL. `status`
Parameter klaster Aurora untuk `force_ssl` PostgreSQL dan untuk MySQL sebelumnya tidak didukung untuk `require_secure_transport` Aurora Serverless v1. Parameter ini tersedia sekarang untuk Aurora Serverless v1. Untuk daftar lengkap parameter yang didukung oleh Aurora Serverless v1, hubungi operasi API. DescribeEngineDefaultClusterParameters Untuk informasi tentang grup parameter dan Aurora Nirserver v1, lihat Kelompok parameter untuk Aurora Serverless v1.

Untuk menggunakan Klien MySQL untuk terhubung ke Aurora Serverless v1 Cluster DB yang menjalankan Aurora MySQL-Compatible Edition, Anda menentukan TLS/SSL dalam permintaan Anda. Contoh berikut mencakup trust store Amazon root CA 1 yang diunduh dari Amazon Trust Services, yang diperlukan agar koneksi ini berhasil.

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

Saat diminta, masukkan kata sandi Anda. Kemudian, monitor MySQL terbuka. Anda dapat mengonfirmasi bahwa sesi dienkripsi menggunakan perintah `status`.

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

Untuk mempelajari selengkapnya tentang menghubungkan ke basis data Aurora MySQL dengan Klien MySQL, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Aurora Serverless v1 mendukung semua mode TLS/SSL yang tersedia untuk MySQL Client (`mysql`) dan PostgreSQL Client (), termasuk yang tercantum dalam tabel berikut. `psql`

| Deskripsi mode TLS/SSL | mysql | psql |
|---|---|---|
| Hubungkan tanpa menggunakan TLS/SSL. | DISABLED | disable |
| Mencoba koneksi menggunakan TLS/SSL terlebih dahulu, tetapi kembali ke non-SSL jika diperlukan. | PREFERRED | prefer (default) |
| Memberlakukan penggunaan TLS/SSL. | REQUIRED | require |
| memberlakukan TLS/SSL dan memverifikasi CA. | VERIFY_CA | verify-ca |
| Memberlakukan TLS/SSL, memverifikasi CA, dan memverifikasi nama host CA. | VERIFY_IDENTITY | verify-full |

Aurora Serverless v1 menggunakan sertifikat wildcard. Jika Anda menentukan opsi "memverifikasi CA" atau "memverifikasi CA dan nama host CA" saat menggunakan TLS/SSL, pertama-tama unduh trust store Amazon root CA 1 dari Amazon Trust Services. Setelah melakukannya, Anda dapat mengidentifikasi file berformat PEM ini dalam perintah klien Anda. Untuk melakukannya menggunakan Klien PostgreSQL:

Untuk Linux, macOS, atau Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Untuk mempelajari selengkapnya tentang mengoperasikan basis data Aurora PostgreSQL menggunakan Klien Postgres, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data PostgreSQL.

Untuk informasi selengkapnya tentang menghubungkan ke klaster DB Aurora secara umum, lihat Menghubungkan ke klaster DB Amazon Aurora.

## Suite cipher yang didukung untuk koneksi ke Aurora Serverless v1 Klaster DB

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data Anda. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS/SSL klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Tindakan ini mencegah penggunaan cipher yang tidak aman atau yang sudah dihentikan.

Aurora Serverless v1 Cluster DB yang didasarkan pada Aurora MySQL mendukung suite cipher yang sama dengan cluster DB yang disediakan Aurora MySQL. Untuk informasi tentang cipher suite ini, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL.

Aurora Serverless v1 Cluster DB yang didasarkan pada Aurora PostgreSQL tidak mendukung cipher suite.

# Bagaimana Aurora Serverless v1 cara kerja

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Setelah itu, Anda dapat mempelajari caranya Aurora Serverless v1 bekerja.

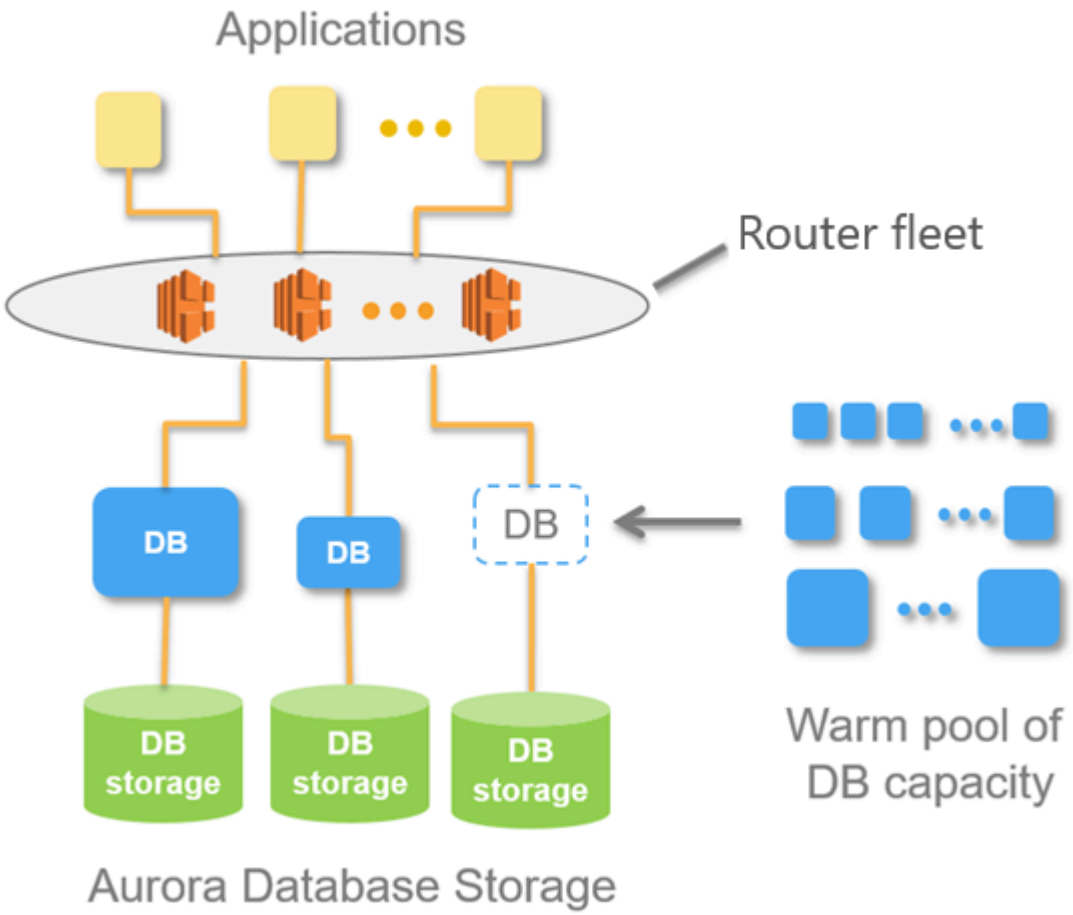
Topik

- Aurora Serverless v1 arsitektur
- Penskalaan otomatis untuk Aurora Serverless v1
- Tindakan batas waktu habis untuk perubahan kapasitas
- Jeda dan lanjutkan untuk Aurora Serverless v1

- Menentukan jumlah maksimum koneksi database untuk Aurora Serverless v1
- Kelompok parameter untuk Aurora Serverless v1
- Logging untuk Aurora Serverless v1
- Aurora Serverless v1 dan pemeliharaan
- Aurora Serverless v1 dan failover
- Aurora Serverless v1 dan snapshot

## Aurora Serverless v1 arsitektur

Gambar berikut menunjukkan ikhtisar dari Aurora Serverless v1 arsitektur.




Alih-alih menyediakan dan mengelola server database, Anda menentukan unit kapasitas Aurora (). ACUs Setiap ACU adalah kombinasi dari sekitar 2 gigabyte (GB) memori, CPU yang sesuai, dan jaringan. Penyimpanan basis data secara otomatis menskalakan dari 10 gibibyte (GiB) hingga 128 tebibyte (TiB), sama seperti penyimpanan dalam klaster DB Aurora standar.

Anda dapat menentukan ACU minimum dan maksimum. Unit kapasitas Aurora minimum adalah ACU terendah yang dapat dicapai saat menurunkan skala klaster DB. Unit kapasitas Aurora maksimum adalah ACU tertinggi yang dapat ditingkatkan skala klaster DB-nya. Berdasarkan pengaturan Anda, Aurora Serverless v1 secara otomatis membuat aturan penskalaan untuk ambang penggunaan CPU, koneksi, dan memori yang tersedia.

Aurora Serverless v1 mengelola kumpulan sumber daya yang hangat Wilayah AWS untuk meminimalkan waktu penskalaan. Saat Aurora Serverless v1 menambahkan sumber daya baru ke klaster Aurora DB, menggunakan router fleet untuk mengalihkan koneksi klien aktif ke sumber daya baru. Pada waktu tertentu, Anda hanya dikenakan biaya untuk ACUs yang sedang digunakan secara aktif di cluster Aurora DB Anda.

## Penskalaan otomatis untuk Aurora Serverless v1

Kapasitas yang dialokasikan ke Aurora Serverless v1 Cluster DB dengan mulus menskalakan naik dan turun berdasarkan beban yang dihasilkan oleh aplikasi klien Anda. Di sini, beban adalah pemanfaatan CPU dan jumlah koneksi. Ketika kapasitas dibatasi oleh salah satu dari ini, Aurora Serverless v1 skala. Aurora Serverless v1 juga meningkat ketika mendeteksi masalah kinerja yang dapat diselesaikan dengan melakukannya.

Anda dapat melihat acara penskalaan untuk Aurora Serverless v1 cluster di AWS Management Console. Selama penskalaan otomatis, Aurora Serverless v1 mereset metrik `EngineUptime`. Nilai nilai metrik reset tidak berarti bahwa penskalaan yang mulus memiliki masalah atau itu Aurora Serverless v1 koneksi terputus. Ini hanyalah titik awal untuk waktu aktif pada kapasitas baru. Untuk mempelajari selengkapnya tentang metrik, lihat Memantau metrik di klaster Amazon Aurora.

Saat Aurora Serverless v1 Cluster DB tidak memiliki koneksi aktif, dapat diturunkan ke kapasitas nol (0 ACUs). Untuk mempelajari selengkapnya, lihat Jeda dan lanjutkan untuk Aurora Serverless v1.

Ketika itu perlu melakukan operasi penskalaan, Aurora Serverless v1 pertama kali mencoba mengidentifikasi titik penskalaan, saat ketika tidak ada kueri yang sedang diproses. Aurora Serverless v1 mungkin tidak dapat menemukan titik penskalaan karena alasan berikut:

- Kueri yang berjalan lama
- Transaksi yang sedang berlangsung
- Tabel atau kunci tabel sementara

Untuk meningkatkan Aurora Serverless v1 Tingkat keberhasilan klaster DB saat menemukan titik penskalaan, kami sarankan Anda menghindari kueri yang berjalan lama dan transaksi yang

berjalan lama. Untuk mempelajari lebih lanjut tentang operasi yang memblokir penskalaan dan cara menghindarinya, lihat Praktik terbaik untuk bekerja dengan Aurora Serverless v1.

Secara default, Aurora Serverless v1 mencoba menemukan titik penskalaan selama 5 menit (300 detik). Anda dapat menentukan periode waktu tunggu yang berbeda saat Anda membuat atau memodifikasi klaster. Periode batas waktu bisa antara 60 detik dan 10 menit (600 detik). Jika Aurora Serverless v1 tidak dapat menemukan titik penskalaan dalam periode yang ditentukan, waktu operasi penskalaan otomatis habis.

Secara default, jika penskalaan otomatis tidak menemukan titik penskalaan sebelum waktu habis, Aurora Serverless v1 menjaga cluster pada kapasitas saat ini. Anda dapat mengubah perilaku default ini saat membuat atau memodifikasi Aurora Serverless v1 Cluster DB dengan memilih opsi Paksa perubahan kapasitas. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.

## Tindakan batas waktu habis untuk perubahan kapasitas

Jika batas waktu penskalaan otomatis habis tanpa menemukan titik penskalaan, Aurora akan secara default mempertahankan kapasitas saat ini. Anda dapat memilih agar Aurora memaksa perubahan dengan memilih opsi Paksa perubahan kapasitas. Opsi ini tersedia di bagian Waktu habis dan tindakan Penskalaan Otomatis pada halaman Buat basis data saat Anda membuat klaster.

Secara default, opsi Paksa perubahan kapasitas tidak dipilih. Jaga agar opsi ini jelas untuk memiliki Aurora Serverless v1 Kapasitas cluster DB tetap tidak berubah jika waktu operasi penskalaan habis tanpa menemukan titik penskalaan.

Memilih opsi ini menyebabkan Anda Aurora Serverless v1 Cluster DB untuk menegakkan perubahan kapasitas, bahkan tanpa titik penskalaan. Sebelum memilih opsi ini, ketahui konsekuensinya:

- Setiap transaksi dalam proses akan terinterupsi, dan pesan kesalahan berikut akan muncul.

  Aurora MySQL versi 2 – ERROR 1105 (HY000): Transaksi terakhir dibatalkan karena Seamless Scaling. Silakan coba lagi.

  Anda dapat mengirimkan kembali transaksi segera setelah Anda Aurora Serverless v1 klaster DB tersedia.
- Koneksi ke tabel dan kunci sementara terputus.

  Kami menyarankan Anda memilih opsi Paksa perubahan kapasitas hanya jika aplikasi Anda dapat dipulihkan dari koneksi yang terputus atau transaksi yang tidak selesai.

Pilihan yang Anda buat AWS Management Console saat Anda membuat Aurora Serverless v1 Cluster DB disimpan dalam `ScalingConfigurationInfo` objek, di `SecondsBeforeTimeout` dan `TimeoutAction` properti. Nilai properti `TimeoutAction` diatur ke salah satu nilai berikut saat Anda membuat klaster:

- `RollbackCapacityChange` – Nilai ini diatur saat Anda memilih opsi Kembalikan perubahan kapasitas. Ini adalah perilaku default.
- `ForceApplyCapacityChange` – Nilai ini diatur saat Anda memilih opsi Paksa perubahan kapasitas.

Anda bisa mendapatkan nilai properti ini pada yang sudah ada Aurora Serverless v1 DB cluster dengan menggunakan describe-db-clusters AWS CLI perintah, seperti yang ditunjukkan berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

Untuk Windows:

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

Sebagai contoh, berikut ini menunjukkan query dan respon untuk Aurora Serverless v1 Cluster DB dinamai `west-coast-sles` di Wilayah AS Barat (California N.).

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
```

```
            "TimeoutAction": "RollbackCapacityChange"
        }
    }
]
```

Seperti yang ditunjukkan oleh tanggapan, ini Aurora Serverless v1 Cluster DB menggunakan pengaturan default.

Untuk informasi selengkapnya, lihat Membuat Aurora Serverless v1 klaster DB.. Setelah membuat Aurora Serverless v1, Anda dapat memodifikasi tindakan batas waktu dan pengaturan kapasitas lainnya kapan saja. Untuk mempelajari caranya, lihat Memodifikasi Aurora Serverless v1 Klaster DB.

## Jeda dan lanjutkan untuk Aurora Serverless v1

Anda dapat memilih untuk menjeda Aurora Serverless v1 klaster DB setelah jangka waktu tertentu tanpa aktivitas. Anda menentukan jumlah waktu tanpa aktivitas sebelum klaster DB dihentikan sementara. Saat Anda memilih opsi ini, waktu tanpa aktivitas default adalah lima menit, tetapi Anda dapat mengubah nilai ini. Ini adalah pengaturan opsional.

Ketika klaster DB dijeda, tidak ada komputasi atau aktivitas memori yang terjadi, dan Anda hanya dikenai biaya untuk penyimpanan. Jika koneksi basis data diminta saat Aurora Serverless v1 klaster DB dijeda, klaster DB secara otomatis melanjutkan dan melayani permintaan koneksi.

Saat klaster DB melanjutkan aktivitas, kapasitasnya sama dengan saat Aurora menjeda klaster. Jumlah ACUs tergantung pada seberapa banyak Aurora menskalakan cluster ke atas atau ke bawah sebelum menjedanya.

> **Note**
>
> Jika klaster DB dijeda selama lebih dari tujuh hari, klaster DB ini mungkin akan dicadangkan dengan snapshot. Dalam kasus ini, Aurora memulihkan klaster DB dari snapshot ketika ada permintaan untuk terhubung ke klaster DB tersebut.

## Menentukan jumlah maksimum koneksi database untuk Aurora Serverless v1

Contoh berikut adalah untuk Aurora Serverless v1 Cluster DB yang kompatibel dengan MySQL 5.7. Anda dapat menggunakan klien MySQL atau editor kueri, jika aksesnya telah Anda konfigurasi. Untuk informasi selengkapnya, lihat Menjalankan kueri di editor kueri.

Untuk menemukan jumlah maksimum koneksi basis data

1. Temukan rentang kapasitas untuk Anda Aurora Serverless v1 Cluster DB menggunakan file AWS CLI.

```
aws rds describe-db-clusters \
    --db-cluster-identifier my-serverless-57-cluster \
    --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
```

   Hasilnya menunjukkan bahwa rentang kapasitasnya adalah 1-4 ACUs.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

2. Jalankan kueri SQL berikut untuk menemukan jumlah maksimum koneksi.

```
select @@max_connections;
```

   Hasil yang ditampilkan ditujukan untuk kapasitas minimum klaster, 1 ACU.

```
@@max_connections
90
```

3. Skala cluster ke 8-32 ACUs.

   Untuk informasi selengkapnya tentang penskalaan, lihat Memodifikasi Aurora Serverless v1 Klaster DB.

4. Konfirmasikan rentang kapasitas.

```
{
    "MinCapacity": 8,
    "AutoPause": true,
    "MaxCapacity": 32,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
```

```
}
```

5. Temukan jumlah maksimum koneksi.

```
select @@max_connections;
```

Hasil yang ditampilkan adalah untuk kapasitas minimum cluster, 8 ACUs.

```
@@max_connections
1000
```

6. Skala cluster semaksimal mungkin, 256-256 ACUs.
7. Konfirmasikan rentang kapasitas.

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. Temukan jumlah maksimum koneksi.

```
select @@max_connections;
```

Hasil yang ditampilkan adalah untuk 256 ACUs.

```
@@max_connections
6000
```

> **Note**
>
> `max_connections`Nilai tidak menskalakan secara linier dengan jumlah. ACUs

9. Skala cluster kembali ke 1-4 ACUs.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
```

```
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

Kali ini, `max_connections` nilainya adalah untuk 4 ACUs.

```
@@max_connections
270
```

10. Biarkan skala cluster turun menjadi 2 ACUs.

```
@@max_connections
180
```

Jika Anda telah mengonfigurasi klaster untuk berhenti sejenak setelah waktu idle tertentu, klaster akan turun menjadi 0. ACUs Namun, `max_connections` tidak akan turun di bawah nilai untuk 1 ACU.

```
@@max_connections
90
```

## Kelompok parameter untuk Aurora Serverless v1

Saat Anda membuat Aurora Serverless v1 Cluster DB, Anda memilih mesin Aurora DB tertentu dan grup parameter cluster DB terkait. Tidak seperti cluster Aurora DB yang disediakan, sebuah Aurora Serverless v1 Cluster DB memiliki satu instance DB baca/tulis yang dikonfigurasi dengan grup parameter cluster DB saja—tidak memiliki grup parameter DB terpisah. Selama penskalaan otomatis, Aurora Serverless v1 harus dapat mengubah parameter agar cluster bekerja paling baik untuk peningkatan atau penurunan kapasitas. Dengan demikian, dengan Aurora Serverless v1 Cluster DB, beberapa perubahan yang mungkin Anda buat pada parameter untuk jenis mesin DB tertentu mungkin tidak berlaku.

Misalnya, berbasis Aurora PostgreSQL Aurora Serverless v1 Cluster DB tidak dapat digunakan `apg_plan_mgmt.capture_plan_baselines` dan parameter lain yang mungkin digunakan pada klaster Aurora PostgreSQL DB yang disediakan untuk manajemen rencana kueri.

Anda bisa mendapatkan daftar nilai default untuk grup parameter default untuk berbagai mesin Aurora DB dengan menggunakan perintah describe-engine-default-cluster-parameter CLI dan menanyakan file. Wilayah AWS Berikut ini adalah nilai yang dapat Anda gunakan untuk opsi `--db-parameter-group-family`.

| | |
|---|---|
| Aurora MySQL versi 2 | `aurora-mysql5.7` |
| Aurora PostgreSQL versi 11 | `aurora-postgresql11` |
| Aurora PostgreSQL versi 13 | `aurora-postgresql13` |

Kami menyarankan Anda mengonfigurasi AWS CLI dengan ID kunci AWS akses dan kunci akses AWS rahasia, dan Anda mengatur Wilayah AWS sebelum menggunakan AWS CLI perintah. Dengan menyediakan Wilayah untuk konfigurasi CLI, Anda tidak perlu memasukkan parameter `--region` saat menjalankan perintah. Untuk mempelajari lebih lanjut tentang mengonfigurasi AWS CLI, lihat Dasar-dasar konfigurasi di Panduan AWS Command Line Interface Pengguna.

Contoh berikut mendapatkan daftar parameter dari grup klaster DB default untuk Aurora MySQL versi 2.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
  --output text
```

Untuk Windows:

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

## Memodifikasi nilai parameter untuk Aurora Serverless v1

Seperti yang dijelaskan dalam , Anda tidak dapat langsung mengubah nilai dalam grup parameter default, terlepas dari jenisnya (grup parameter klaster DB, grup parameter DB). Sebagai gantinya, Anda membuat grup parameter kustom berdasarkan grup parameter klaster DB default untuk mesin Aurora DB Anda dan mengubah pengaturan yang diperlukan pada grup parameter tersebut. Misalnya, Anda mungkin ingin mengubah beberapa pengaturan untuk Aurora Serverless v1 Cluster DB untuk mencatat kueri atau mengunggah log khusus mesin DB ke Amazon CloudWatch.

Untuk membuat grup parameter klaster DB kustom

1. Masuk ke AWS Management Console dan kemudian buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Grup parameter.
3. Pilih Buat grup parameter untuk membuka panel detail Grup parameter.
4. Pilih grup klaster DB default yang sesuai untuk engine DB yang ingin Anda gunakan untuk berkas Aurora Serverless v1 klaster DB. Pastikan Anda memilih opsi berikut:

    a. Untuk Rangkaian grup parameter, pilih rangkaian grup parameter yang sesuai untuk mesin DB pilihan Anda. Pastikan bahwa pilihan Anda memiliki awalan `aurora-` dalam namanya.

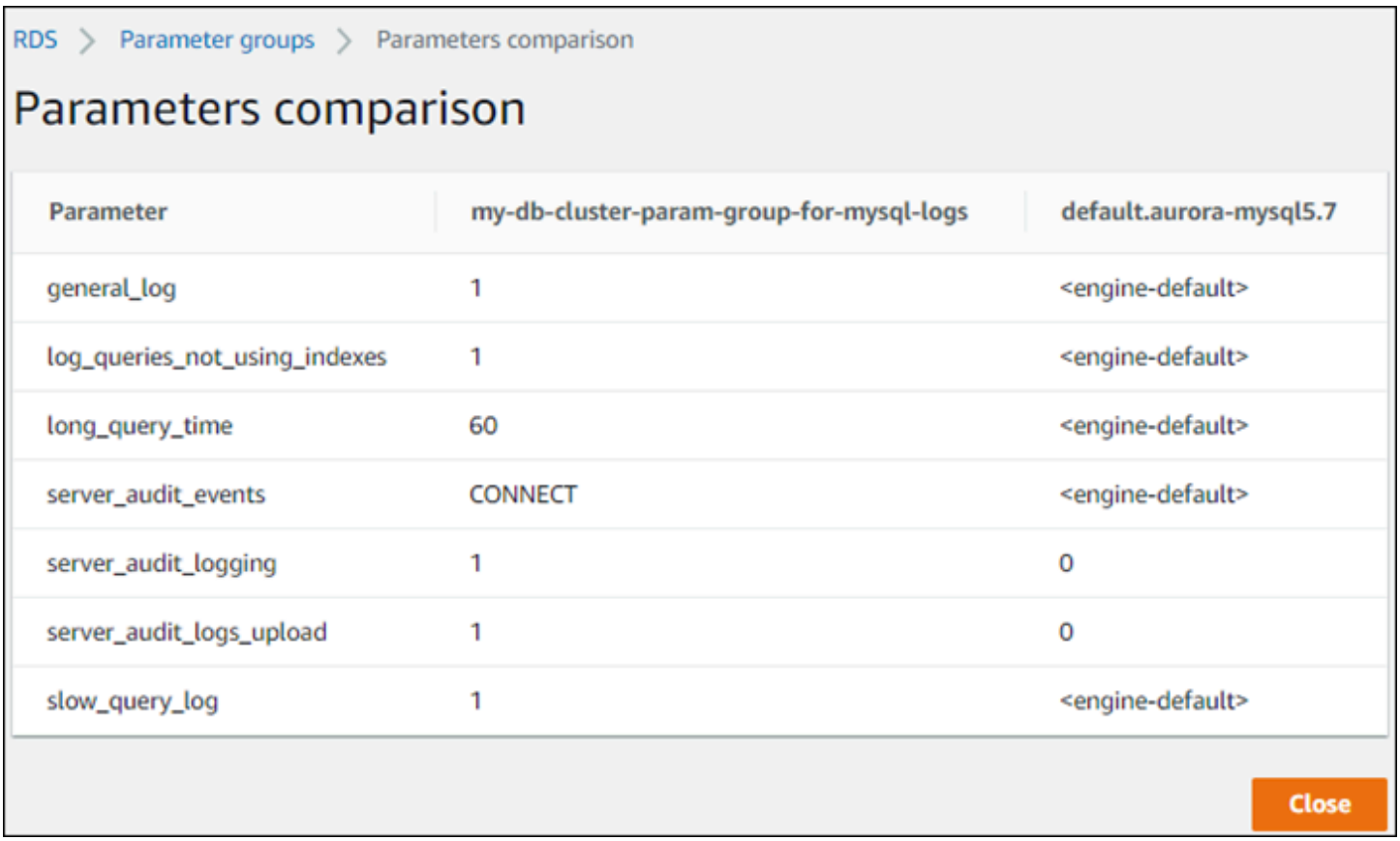
    b. Untuk Jenis, pilih Grup Parameter Klaster DB.

    c. Untuk Nama grup dan Deskripsi, masukkan nama yang bermakna untuk Anda atau orang lain yang mungkin perlu bekerja dengan Aurora Serverless v1 klaster DB dan parameternya.

    d. Pilih Buat.

Grup parameter klaster DB kustom Anda ditambahkan ke daftar grup parameter yang tersedia di Wilayah AWS Anda. Anda dapat menggunakan grup parameter cluster DB kustom Anda saat Anda membuat yang baru Aurora Serverless v1 klaster DB. Anda juga dapat memodifikasi yang sudah ada Aurora Serverless v1 Cluster DB untuk menggunakan grup parameter cluster DB kustom Anda. Setelah Anda Aurora Serverless v1 Cluster DB mulai menggunakan grup parameter cluster DB kustom Anda, Anda dapat mengubah nilai untuk parameter dinamis menggunakan salah satu AWS Management Console atau AWS CLI.

Anda juga dapat menggunakan konsol untuk melihat side-by-side perbandingan nilai dalam grup parameter cluster DB kustom Anda dibandingkan dengan grup parameter cluster DB default, seperti yang ditunjukkan pada gambar berikut.




Saat Anda mengubah nilai parameter pada cluster DB aktif, Aurora Serverless v1 memulai skala mulus untuk menerapkan perubahan parameter. Jika Aurora Serverless v1 Cluster DB dalam keadaan jeda, ia melanjutkan dan mulai menskalakan sehingga dapat membuat perubahan. Operasi penskalaan untuk perubahan grup parameter selalu memaksa perubahan kapasitas, jadi ketahui bahwa memodifikasi parameter dapat mengakibatkan koneksi terputus jika titik penskalaan tidak dapat ditemukan selama periode penskalaan.

## Logging untuk Aurora Serverless v1

Secara default, log kesalahan untuk Aurora Serverless v1 diaktifkan dan diunggah secara otomatis ke Amazon CloudWatch. Anda juga dapat memiliki Aurora Serverless v1 Cluster DB mengunggah log khusus mesin database Aurora ke. CloudWatch Untuk melakukannya, aktifkan parameter konfigurasi di grup parameter klaster DB kustom Anda. Klaster Aurora Serverless v1 Cluster DB kemudian mengunggah semua log yang tersedia ke Amazon CloudWatch. Pada titik ini, Anda dapat menggunakan CloudWatch untuk menganalisis data log, membuat alarm, dan melihat metrik.

Untuk Aurora MySQL, tabel berikut menunjukkan log yang dapat Anda aktifkan. Ketika diaktifkan, mereka secara otomatis diunggah dari Aurora Serverless v1 Cluster DB ke Amazon CloudWatch.

| Log Aurora MySQL | Deskripsi |
| --- | --- |
| `general_log` | Membuat log umum. Atur ke 1 untuk mengaktif kan. Default-nya adalah nonaktif (0). |
| `log_queries_not_using_indexes` | Mencatat setiap kueri ke log kueri lambat yang tidak menggunakan indeks. Default-nya adalah nonaktif (0). Atur ke 1 untuk mengaktifkan log ini. |
| `long_query_time` | Mencegah kueri yang berjalan cepat agar tidak dicatat dalam log kueri lambat. Dapat diatur ke angka float antara 0 dan 31536000. Default-nya adalah 0 (tidak aktif). |
| `server_audit_events` | Daftar peristiwa untuk dicatat dalam log. Nilai yang didukung adalah `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` , dan `TABLE`. |
| `server_audit_logging` | Atur ke 1 untuk mengaktifkan pencatatan log audit server. Jika Anda mengaktifkan ini, Anda dapat menentukan peristiwa audit yang akan dikirim CloudWatch dengan mencantumkannya di `server_audit_events` parameter. |
| `slow_query_log` | Membuat log kueri lambat. Atur ke 1 untuk mengaktifkan log kueri lambat. Default-nya adalah nonaktif (0). |

Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster Amazon Aurora My DB SQL.

Untuk Aurora PostgreSQL, tabel berikut menunjukkan log yang dapat Anda aktifkan. Ketika diaktifkan, mereka secara otomatis diunggah dari Aurora Serverless v1 Cluster DB ke Amazon CloudWatch bersama dengan log kesalahan biasa.

| Log Aurora PostgreSQL | Deskripsi |
| --- | --- |
| `log_connections` | Diaktifkan secara default dan tidak dapat diubah. Log ini mencatat log detail untuk semua koneksi klien baru. |
| `log_disconnections` | Diaktifkan secara default dan tidak dapat diubah. Mencatat log semua pemutusan koneksi klien. |
| `log_hostname` | Dimatikan secara default dan tidak dapat diubah. Nama host tidak dicatat. |
| `log_lock_waits` | Default-nya adalah 0 (nonaktif). Atur ke 1 untuk mencatat log peristiwa tunggu kunci. |
| `log_min_duration_statement` | Durasi minimum (dalam milidetik) untuk menjalankan pernyataan sebelum dicatat lognya. |
| `log_min_messages` | Mengatur tingkat pesan yang dicatat lognya. Nilai yang didukung adalah `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`.<br><br>Untuk mencatat data performa ke log `postgres`, tetapkan nilainya ke `debug1`. |
| `log_temp_files` | Mencatat log penggunaan file sementara yang melampaui kilobyte (kB) yang ditentukan. |
| `log_statement` | Mengontrol pernyataan SQL tertentu yang dicatat lognya. Nilai yang didukung adalah `none`, `ddl`, `mod`, dan `all`. Default-nya adalah `none`. |

Setelah Anda mengaktifkan log untuk Aurora MySQL atau Aurora PostgreSQL untuk Anda Aurora Serverless v1 Cluster DB, Anda dapat melihat log masuk CloudWatch.

## Melihat Aurora Serverless v1 log dengan Amazon CloudWatch

Aurora Serverless v1 secara otomatis mengunggah ("menerbitkan") ke Amazon CloudWatch semua log yang diaktifkan di grup parameter cluster DB kustom Anda. Anda tidak perlu memilih atau menentukan jenis log. Mengunggah log dimulai segera setelah Anda mengaktifkan parameter konfigurasi log. Jika Anda menonaktifkan parameter log nanti, pengunggahan lainnya akan berhenti. Namun, semua log yang telah diterbitkan CloudWatch tetap ada sampai Anda menghapusnya.

Untuk informasi lebih lanjut tentang penggunaan CloudWatch dengan log Aurora MySQL, lihat. Memantau peristiwa log di Amazon CloudWatch

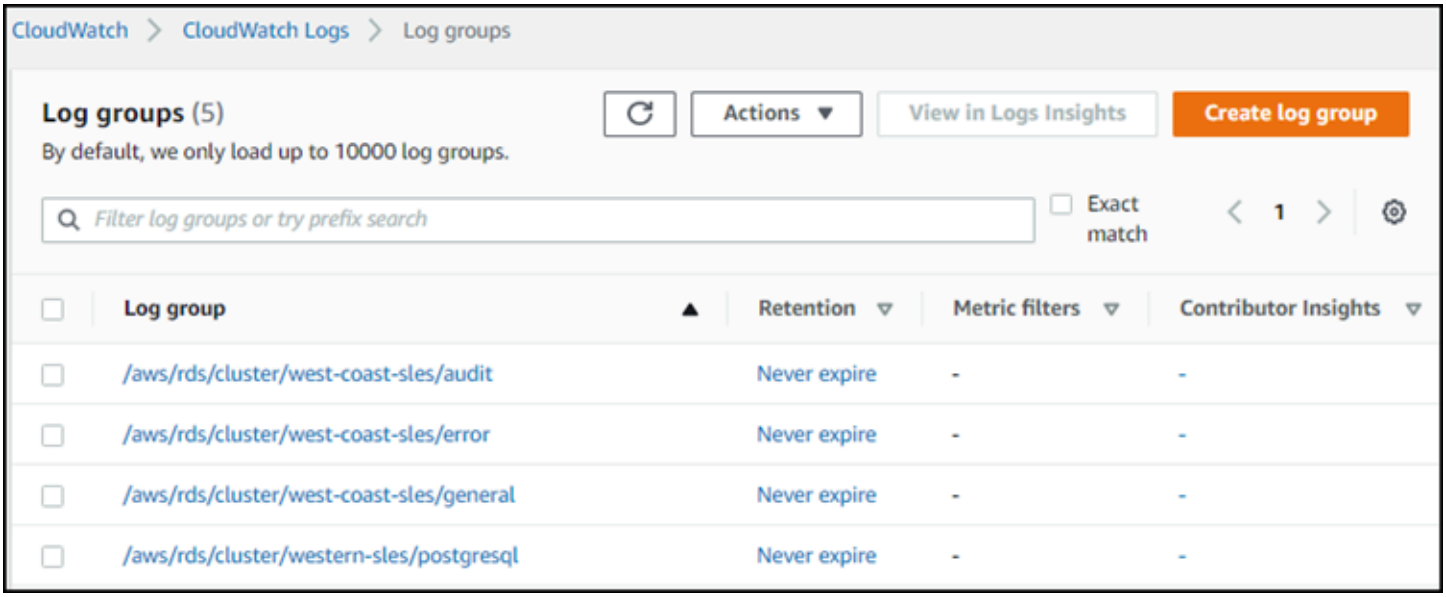
Untuk informasi lebih lanjut tentang CloudWatch dan Aurora PostgreSQL, lihat. Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch

Untuk melihat log untuk Anda Aurora Serverless v1 Klaster DB

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Anda Wilayah AWS.
3. Pilih Grup log.
4. Pilih Anda Aurora Serverless v1 Log cluster DB dari daftar. Untuk log kesalahan, pola penamaannya adalah sebagai berikut.

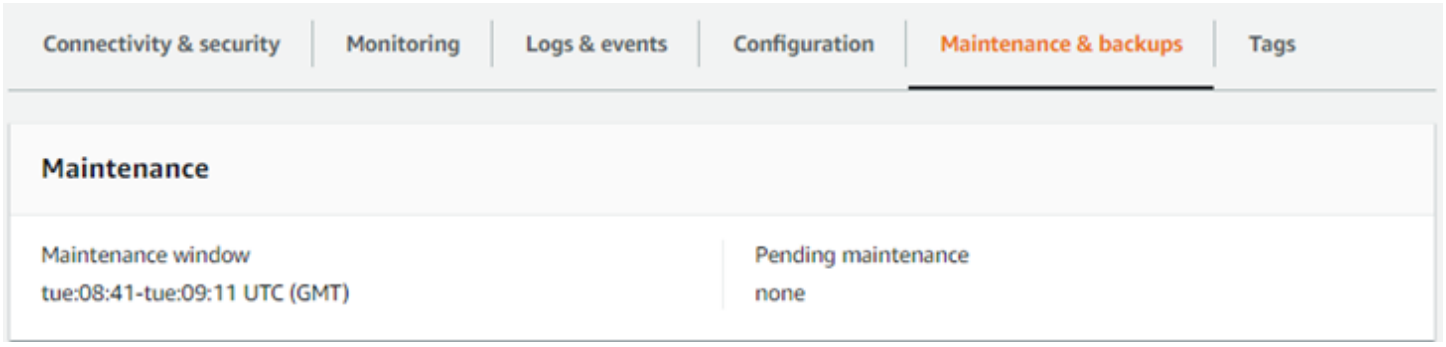
   ```
   /aws/rds/cluster/cluster-name/error
   ```

Misalnya, dalam tangkapan layar berikut Anda dapat menemukan daftar untuk log yang diterbitkan untuk PostgreSQL Aurora Aurora Serverless v1 Cluster DB bernama`western-sles`. Anda juga dapat menemukan beberapa daftar untuk Aurora MySQL Aurora Serverless v1 Kluster DB,`west-coast-sles`. Pilih log yang Anda minati untuk mulai mengkaji kontennya.




## Aurora Serverless v1 dan pemeliharaan

Pemeliharaan untuk Aurora Serverless v1 klaster DB, seperti menerapkan fitur terbaru, perbaikan, dan pembaruan keamanan, dilakukan secara otomatis untuk Anda. Aurora Serverless v1 memiliki jendela pemeliharaan yang dapat Anda lihat di AWS Management Console dalam Pemeliharaan & cadangan untuk Aurora Serverless v1 klaster DB. Anda dapat menemukan tanggal dan waktu pemeliharaan yang mungkin dilakukan dan jika ada pemeliharaan yang tertunda untuk Anda Aurora Serverless v1 DB cluster, seperti yang ditunjukkan pada gambar berikut.




Anda dapat mengatur jendela pemeliharaan saat Anda membuat Aurora Serverless v1 DB cluster, dan Anda dapat memodifikasi jendela nanti. Untuk informasi selengkapnya, lihat Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.

Jendela pemeliharaan digunakan untuk peningkatan versi mayor terjadwal. Upgrade versi minor dan patch diterapkan segera selama penskalaan. Penskalaan terjadi sesuai dengan pengaturan Anda untuk`TimeoutAction`:

- `ForceApplyCapacityChange`Perubahan diterapkan segera.
- `RollbackCapacityChange`— Aurora secara paksa memperbarui cluster setelah 3 hari dari upaya patch pertama.

Seperti halnya perubahan apa pun yang dipaksakan tanpa titik penskalaan yang tepat, hal ini dapat mengganggu beban kerja Anda.

Sebisa mungkin, Aurora Serverless v1 melakukan pemeliharaan dengan cara yang tidak mengganggu. Saat pemeliharaan diperlukan, Aurora Serverless v1 klaster DB menskalakan kapasitasnya untuk menangani operasi yang diperlukan. Sebelum melakukan penskalaan, Aurora Serverless v1 mencari titik penskalaan. Itu dilakukan hingga tiga hari jika perlu.

Di akhir setiap hari itu yang Aurora Serverless v1 tidak dapat menemukan titik penskalaan, itu menciptakan peristiwa cluster. Peristiwa ini memberi tahu Anda tentang pemeliharaan yang tertunda dan kebutuhan penskalaan untuk melakukan pemeliharaan. Pemberitahuan termasuk tanggal kapan Aurora Serverless v1 dapat memaksa klaster DB untuk menskalakan.

Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.

## Aurora Serverless v1 dan failover

Jika instans DB untuk Aurora Serverless v1 Cluster DB menjadi tidak tersedia atau Availability Zone (AZ) gagal, Aurora membuat ulang instans DB di AZ yang berbeda. Namun, Aurora Serverless v1 cluster bukanlah cluster multi-AZ. Hal ini karena klaster tersebut terdiri dari satu instans DB dalam satu AZ. Dengan demikian, mekanisme failover ini membutuhkan waktu lebih lama daripada cluster Aurora dengan provisioned atau Aurora Serverless v2 contoh. Bagian Aurora Serverless v1 waktu failover tidak ditentukan karena tergantung pada permintaan dan ketersediaan kapasitas di tempat lain AZs dalam yang diberikan. Wilayah AWS

Karena Aurora memisahkan kapasitas komputasi dan penyimpanan, volume penyimpanan untuk cluster tersebar di beberapa. AZs Data Anda tetap tersedia meskipun pemadaman memengaruhi instans DB atau AZ terkait.

## Aurora Serverless v1 dan snapshot

Volume klaster untuk Aurora Serverless v1 klaster selalu dienkripsi. Anda dapat memilih kunci enkripsi, tetapi tidak dapat menonaktifkan enkripsi. Untuk menyalin atau berbagi snapshot Aurora Serverless v1 cluster, enkripsi snapshot menggunakan milik Anda sendiri. AWS KMS key Untuk

informasi selengkapnya, lihat Penyalinan snapshot cluster DB. Untuk mempelajari selengkapnya tentang enkripsi dan Amazon Aurora, lihat Mengenkripsi klaster DB Amazon Aurora

# Membuat Aurora Serverless v1 klaster DB.

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Prosedur berikut menciptakan Aurora Serverless v1 cluster tanpa objek skema atau data Anda. Jika Anda ingin membuat Aurora Serverless v1 cluster yang merupakan duplikat dari yang sudah ada atau Aurora Serverless v1 cluster, Anda dapat melakukan pemulihan snapshot atau operasi kloning sebagai gantinya. Untuk detailnya, lihat Memulihkan dari snapshot klaster DB dan Mengkloning volume untuk klaster DB Amazon Aurora. Anda tidak dapat mengonversi kluster yang sudah ada menjadi Aurora Serverless v1. Anda juga tidak dapat mengonversi yang sudah ada Aurora Serverless v1 cluster kembali ke cluster yang disediakan.

Saat Anda membuat Aurora Serverless v1 klaster DB, Anda dapat mengatur kapasitas minimum dan maksimum untuk klaster. Unit kapasitas setara dengan komputasi dan konfigurasi memori tertentu. Aurora Serverless v1 membuat aturan penskalaan untuk ambang batas untuk pemanfaatan CPU, koneksi, dan memori yang tersedia dan menskalakan dengan mulus ke berbagai unit kapasitas sesuai kebutuhan untuk aplikasi Anda. Untuk mengetahui informasi selengkapnya, lihat Aurora Serverless v1 arsitektur.

Anda dapat mengatur nilai spesifik berikut untuk Aurora Serverless v1 klaster DB.

- unit kapasitas Aurora minimum – Aurora Serverless v1 dapat mengurangi kapasitas turun hingga unit kapasitas ini.
- unit kapasitas Aurora maksimum – Aurora Serverless v1 dapat meningkatkan kapasitas naik untuk unit kapasitas ini.

Anda juga dapat memilih opsi konfigurasi penskalaan opsional berikut:

- Paksa penskalaan kapasitas ke nilai yang ditentukan saat waktu habis tercapai – Anda dapat memilih pengaturan ini jika Anda mau Aurora Serverless v1 memaksakan Aurora Serverless v1 menskalakan meskipun tidak dapat menemukan titik penskalaan sebelum waktu habis. Jika Anda ingin Aurora Serverless v1 untuk membatalkan perubahan kapasitas jika tidak dapat menemukan titik skala, jangan pilih pengaturan ini. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda kapasitas komputasi setelah beberapa menit berturut-turut tidak aktif – Anda dapat memilih pengaturan ini jika Anda mau Aurora Serverless v1 untuk menskalakan ke nol saat tidak ada aktivitas di klaster DB Anda selama jumlah waktu yang Anda tentukan. Dengan pengaturan ini diaktifkan, Aurora Serverless v1 klaster DB secara otomatis melanjutkan pemrosesan dan menskalakan ke kapasitas yang diperlukan untuk menangani beban kerja saat lalu lintas basis data dilanjutkan. Untuk mempelajari selengkapnya, lihat Jeda dan lanjutkan untuk Aurora Serverless v1.

Sebelum Anda dapat membuat Aurora Serverless v1 Cluster DB, Anda memerlukan AWS akun. Anda juga perlu menyelesaikan tugas penyiapan untuk menggunakan Amazon Aurora. Untuk informasi selengkapnya, lihat Menyiapkan lingkungan Anda untuk Amazon Aurora. Anda juga perlu menyelesaikan langkah awal lainnya untuk membuat klaster DB Aurora. Untuk mempelajari selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Aurora Serverless v1 tersedia dalam versi tertentu Wilayah AWS dan untuk Aurora MySQL dan Aurora PostgreSQL tertentu saja. Untuk informasi selengkapnya, lihat Aurora Serverless v1.

> **Note**
>
> Volume klaster untuk Aurora Serverless v1 klaster selalu dienkripsi. Saat Anda membuat Aurora Serverless v1 klaster DB, Anda tidak dapat mematikan enkripsi, tetapi Anda dapat memilih untuk menggunakan kunci enkripsi Anda sendiri. Dengan Aurora Serverless v2, Anda dapat memilih apakah akan mengenkripsi volume cluster.

Anda dapat membuat Aurora Serverless v1 Cluster DB dengan AWS CLI atau RDS API.

> **Note**
>
> Jika Anda menerima pesan kesalahan berikut saat mencoba membuat klaster Anda, akun Anda memerlukan izin tambahan.

```
Unable to create the resource. Verify that you have permission to
create service linked role. Otherwise wait and try again later.
```

Lihat Menggunakan peran terkait layanan untuk Amazon Aurora untuk informasi selengkapnya.

Anda tidak dapat langsung terhubung ke instans DB di Aurora Serverless v1 klaster DB. Untuk terhubung ke Aurora Serverless v1 klaster DB, Anda menggunakan titik akhir basis data. Anda dapat menemukan titik akhir untuk Aurora Serverless v1 Cluster DB pada tab Konektivitas & keamanan untuk cluster Anda di AWS Management Console. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Amazon Aurora.

### AWS CLI

Untuk membuat Aurora Serverless v1 Cluster DB dengan AWS CLI, jalankan create-db-clusterperintah dan tentukan `serverless` untuk `--engine-mode` opsi.

Secara opsional, Anda dapat menentukan opsi `--scaling-configuration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi.

Contoh perintah berikut membuat klaster DB Nirserver baru dengan mengatur opsi `--engine-mode` ke `serverless`. Contoh-contoh tersebut juga menentukan nilai untuk opsi `--scaling-configuration`.

Contoh untuk Aurora MySQL

Perintah berikut membuat klaster DB Nirserver yang kompatibel dengan Aurora MySQL baru. Nilai kapasitas yang valid untuk Aurora MySQL adalah 1, 2, 4, 8, 16, 32, 64, 128, dan 256.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
```

```
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

Contoh untuk Aurora PostgreSQL

Perintah berikut membuat klaster DB Nirserver yang kompatibel dengan PostgreSQL 13.9 baru. Nilai kapasitas yang valid untuk Aurora PostgreSQL adalah 2, 4, 8, 16, 32, 64, 192, dan 384.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

## API RDS

Untuk membuat Aurora Serverless v1 Cluster DB dengan API RDS, jalankan DBCluster operasi Create dan tentukan `serverless` `EngineMode` parameternya.

Secara opsional, Anda dapat menentukan parameter `ScalingConfiguration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

# Memulihkan Aurora Serverless v1 klaster DB

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Anda dapat mengonfigurasi Aurora Serverless v1 Cluster DB saat Anda memulihkan snapshot cluster DB yang disediakan dengan AWS CLI atau RDS API.

Saat Anda memulihkan snapshot ke Aurora Serverless v1 klaster DB, Anda dapat mengatur nilai spesifik berikut:

- unit kapasitas Aurora minimum – Aurora Serverless v1 dapat mengurangi kapasitas turun hingga unit kapasitas ini.
- unit kapasitas Aurora maksimum – Aurora Serverless v1 dapat meningkatkan kapasitas naik untuk unit kapasitas ini.
- Timeout action – Tindakan yang harus diambil jika waktu modifikasi kapasitas habis karena tidak dapat menemukan titik skala. Aurora Serverless v1 Klaster DB dapat memaksa klaster DB Anda ke pengaturan kapasitas baru jika mengatur opsi Paksa penskalaan kapasitas ke nilai yang ditentukan....  Atau, hal ini dapat melakukan rollback perubahan kapasitas untuk membatalkannya jika Anda tidak memilih opsi tersebut. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda setelah tidak aktif – Jumlah waktu tanpa lalu lintas basis data untuk menskalakan ke kapasitas pemrosesan nol. Saat lalu lintas basis data dilanjutkan, Aurora secara otomatis melanjutkan kapasitas dan skala pemrosesan untuk menangani lalu lintas.

Untuk informasi umum tentang memulihkan klaster DB dari snapshot, lihat Memulihkan dari snapshot klaster DB.

## AWS CLI

Anda dapat mengonfigurasi Aurora Serverless Cluster DB saat Anda memulihkan snapshot cluster DB yang disediakan dengan AWS Management Console, the AWS CLI, atau RDS API.

Saat Anda memulihkan snapshot ke Aurora Serverless klaster DB, Anda dapat mengatur nilai spesifik berikut:

- unit kapasitas Aurora minimum – Aurora Serverless dapat mengurangi kapasitas turun hingga unit kapasitas ini.
- unit kapasitas Aurora maksimum – Aurora Serverless dapat meningkatkan kapasitas naik untuk unit kapasitas ini.
- Timeout action – Tindakan yang harus diambil jika waktu modifikasi kapasitas habis karena tidak dapat menemukan titik skala. Aurora Serverless v1 Klaster DB dapat memaksa klaster DB Anda ke pengaturan kapasitas baru jika mengatur opsi Paksa penskalaan kapasitas ke nilai yang ditentukan....  Atau, hal ini dapat melakukan rollback perubahan kapasitas untuk membatalkannya jika Anda tidak memilih opsi tersebut. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda setelah tidak aktif – Jumlah waktu tanpa lalu lintas basis data untuk menskalakan ke kapasitas pemrosesan nol. Saat lalu lintas basis data dilanjutkan, Aurora secara otomatis melanjutkan kapasitas dan skala pemrosesan untuk menangani lalu lintas.

> **Note**
>
> Versi snapshot cluster DB harus kompatibel dengan Aurora Serverless v1. Untuk daftar versi yang didukung, lihatAurora Serverless v1.

Untuk memulihkan snapshot ke Aurora Serverless v1 cluster dengan MySQL 5.7 kompatibilitas, termasuk parameter tambahan berikut:

- `--engine` **`aurora-mysql`**
- `--engine-version` **`5.7`**

`--engine-version`Parameter `--engine` dan memungkinkan Anda membuat MySQL 5.7 kompatibel Aurora Serverless v1 cluster dari MySQL 5.6 yang kompatibel dengan Aurora atau

Aurora Serverless v1 snapshot. Contoh berikut mengembalikan snapshot dari cluster MySQL 5.6 kompatibel bernama MySQL 5.7 kompatibel *mydbclustersnapshot* Aurora Serverless v1 cluster bernama*mynewdbcluster*.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

Secara opsional, Anda dapat menentukan opsi `--scaling-configuration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Dalam contoh berikut, Anda memulihkan dari snapshot cluster DB yang dibuat sebelumnya bernama *mydbclustersnapshot* ke cluster DB baru bernama*mynewdbcluster*. Anda mengatur `--scaling-configuration` sehingga yang baru Aurora Serverless v1 Cluster DB dapat menskalakan dari 8 ACUs hingga 64 ACUs (unit kapasitas Aurora) sesuai kebutuhan untuk memproses beban kerja. Setelah pemrosesan selesai dan setelah 1000 detik tanpa koneksi yang didukung, klaster akan dinonaktifkan hingga permintaan koneksi memintanya untuk diaktifkan ulang.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
```

```
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

### API RDS

Untuk mengonfigurasi Aurora Serverless v1 Cluster DB saat Anda memulihkan dari cluster DB menggunakan RDS API, jalankan DBCluster FromSnapshot operasi Restore dan tentukan `serverless` `EngineMode` parameternya.

Secara opsional, Anda dapat menentukan parameter `ScalingConfiguration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

## Memodifikasi Aurora Serverless v1 Klaster DB

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Setelah Anda mengonfigurasi Aurora Serverless v1 DB cluster, Anda dapat memodifikasi properti tertentu dengan AWS CLI,, atau RDS API. AWS Management Console Sebagian besar properti yang dapat Anda ubah sama dengan jenis klaster Aurora lainnya.

Perubahan yang paling relevan untuk Aurora Serverless v1 adalah sebagai berikut:

- Memodifikasi konfigurasi penskalaan
- Meningkatkan versi mayor
- Konversi dari Aurora Serverless v1 untuk disediakan

## Memodifikasi konfigurasi penskalaan sebuah Aurora Serverless v1 Klaster DB

Anda dapat mengatur kapasitas minimum dan maksimum untuk klaster DB. Setiap unit kapasitas setara dengan komputasi dan konfigurasi memori tertentu. Aurora Serverless secara otomatis membuat aturan penskalaan untuk ambang penggunaan CPU, koneksi, dan memori yang tersedia. Anda juga dapat mengatur apakah Aurora Serverless menjeda basis data saat tidak ada aktivitas lalu melanjutkan saat aktivitas dimulai lagi.

Anda dapat mengatur nilai spesifik berikut untuk konfigurasi penskalaan:

- unit kapasitas Aurora minimum – Aurora Serverless dapat mengurangi kapasitas turun hingga unit kapasitas ini.
- unit kapasitas Aurora maksimum – Aurora Serverless dapat meningkatkan kapasitas naik untuk unit kapasitas ini.
- Batas waktu dan tindakan penskalaan otomatis - Bagian ini menentukan berapa lama Aurora Serverless menunggu untuk menemukan titik penskalaan sebelum waktu habis. Bagian ini juga menentukan tindakan yang harus diambil jika batas waktu modifikasi kapasitas habis karena tidak dapat menemukan titik penskalaan. Aurora dapat memaksa perubahan kapasitas untuk mengatur kapasitas ke nilai yang ditentukan sesegera mungkin. Atau, layanan ini dapat melakukan rollback perubahan kapasitas untuk membatalkannya. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda setelah tidak aktif — Gunakan Skala kapasitas opsional ke 0 ACUs saat klaster dalam pengaturan idle untuk menskalakan database ke nol kapasitas pemrosesan saat klaster tidak aktif. Saat lalu lintas basis data dilanjutkan, Aurora secara otomatis melanjutkan kapasitas dan skala pemrosesan untuk menangani lalu lintas.

> **Note**
>
> Saat Anda memodifikasi rentang kapasitas untuk sebuah Aurora Serverless Cluster DB, perubahan terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama jendela pemeliharaan terjadwal berikutnya.

Konsol

Anda dapat mengubah konfigurasi penskalaan klaster DB Aurora dengan AWS Management Console.

Untuk mengubah Aurora Serverless v1 Klaster DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih Aurora Serverless v1 klaster DB yang ingin Anda ubah.
4. Untuk Actions, pilih Modify cluster.
5. Di bagian Pengaturan kapasitas, ubah konfigurasi penskalaan.
6. Pilih Lanjutkan.
7. Pada halaman Modifikasi klaster DB, tinjau modifikasi Anda, lalu pilih kapan harus menerapkannya.
8. Pilih Ubah klaster.

AWS CLI

Untuk mengubah konfigurasi penskalaan Aurora Serverless v1 Cluster DB menggunakan AWS CLI, jalankan modify-db-cluster AWS CLI perintah. Tentukan opsi `--scaling-configuration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: `1`, `2`, `4`, `8`, `16`, `32`, `64`, `128`, dan `256`.
- Aurora PostgreSQL: `2`, `4`, `8`, `16`, `32`, `64`, `192`, dan `384`.

Dalam contoh ini, Anda memodifikasi konfigurasi penskalaan Aurora Serverless v1 Cluster DB bernama*`sample-cluster`*.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

API RDS

Anda dapat memodifikasi konfigurasi penskalaan cluster Aurora DB dengan operasi DBCluster Modify API. Tentukan parameter `ScalingConfiguration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

## Memutakhirkan versi utama dari sebuah Aurora Serverless v1 Klaster DB

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Anda dapat meng-upgrade versi utama untuk Aurora Serverless v1 Cluster DB kompatibel dengan PostgreSQL 11 ke versi PostgreSQL 13 yang kompatibel dengan PostgreSQL yang sesuai.

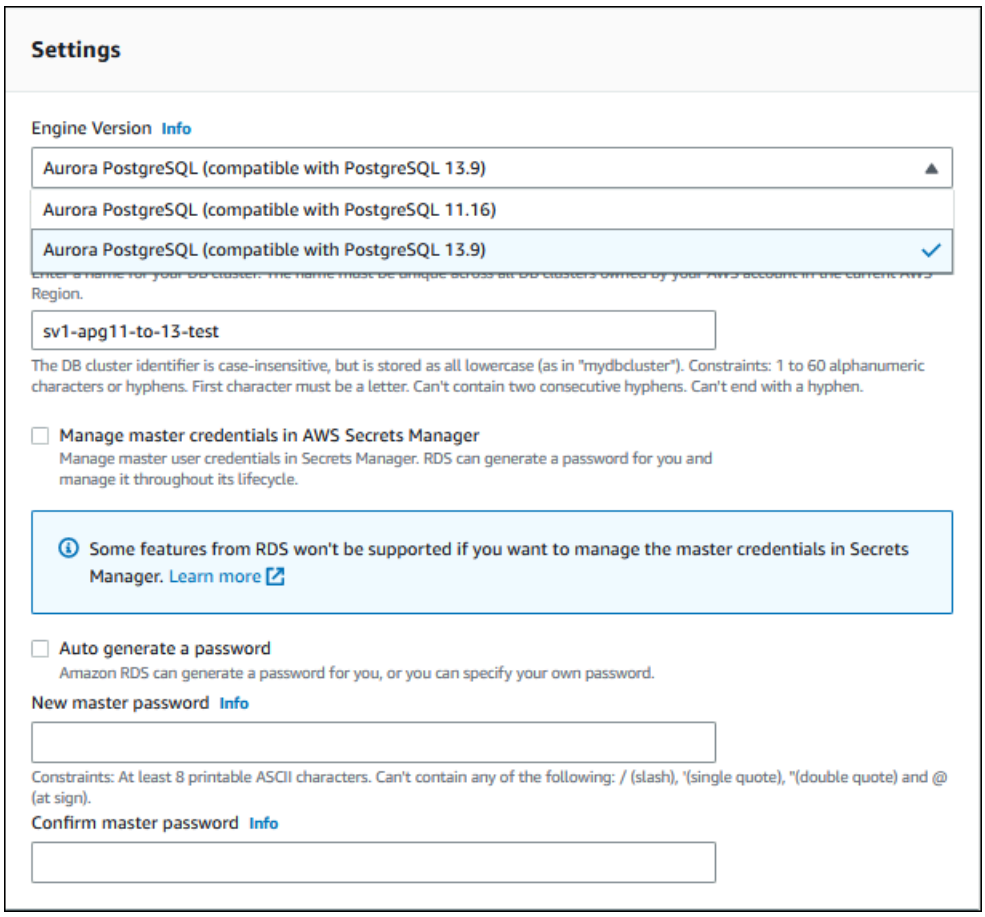
Konsol

Anda dapat melakukan upgrade di tempat dari sebuah Aurora Serverless v1 Cluster DB menggunakan file AWS Management Console.

Untuk meng-upgrade sebuah Aurora Serverless v1 Klaster DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih Aurora Serverless v1 Cluster DB yang ingin Anda tingkatkan.
4. Untuk Tindakan, pilih Ubah klaster.
5. Untuk Versi, pilih nomor versi Aurora PostgreSQL versi 13.

   Contoh berikut menunjukkan peningkatan di tempat dari Aurora PostgreSQL 11.16 ke 13.9.




   Jika Anda melakukan peningkatan versi mayor, biarkan semua properti lainnya tetap sama. Untuk mengubah properti lainnya, lakukan operasi Modifikasi lain setelah peningkatan selesai.
6. Pilih Lanjutkan.

7. Pada halaman Modifikasi klaster DB, tinjau modifikasi Anda, lalu pilih kapan harus menerapkannya.
8. Pilih Ubah klaster.

AWS CLI

Untuk melakukan upgrade di tempat dari PostgreSQL 11 yang kompatibel Aurora Serverless v1 Cluster DB ke PostgreSQL 13 — yang kompatibel dengan PostgreSQL, tentukan parameter `--engine-version` dengan nomor versi Aurora PostgreSQL versi 13 yang kompatibel dengan Aurora Serverless v1. Juga termasuk `--allow-major-version-upgrade` parameter.

Dalam contoh ini, Anda memodifikasi versi utama dari PostgreSQL 11 yang kompatibel Aurora Serverless v1 Cluster DB bernama`sample-cluster`. Melakukan hal itu melakukan upgrade di tempat ke PostgreSQL 13 yang kompatibel Aurora Serverless v1 klaster DB.

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 13.serverless_12 \
    --allow-major-version-upgrade
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.serverless_12 ^
    --allow-major-version-upgrade
```

API RDS

Untuk melakukan upgrade di tempat dari PostgreSQL 11 yang kompatibel Aurora Serverless v1 Cluster DB ke PostgreSQL 13 — yang kompatibel dengan PostgreSQL, tentukan parameter `EngineVersion` dengan nomor versi Aurora PostgreSQL versi 13 yang kompatibel dengan Aurora Serverless v1. Juga termasuk `AllowMajorVersionUpgrade` parameter.

## Mengonversi sebuah Aurora Serverless v1 Cluster DB untuk disediakan

Anda dapat mengonversi sebuah Aurora Serverless v1 Cluster DB ke cluster DB yang disediakan. Untuk melakukan konversi, gunakan AWS CLI atau Amazon RDS API untuk mengubah kelas instans

DB ke Provisioned. Gunakan langkah-langkah di bawah ini untuk memodifikasi kelas instans DB Anda.

Contoh berikut menunjukkan bagaimana menggunakan AWS CLI untuk mengkonversi Aurora Serverless v1 Cluster DB ke cluster yang disediakan.

AWS CLI

Untuk mengonversi sebuah Aurora Serverless v1 Cluster DB ke cluster yang disediakan, jalankan perintah. modify-db-cluster AWS CLI

Parameter-parameter berikut diperlukan:

- `--db-cluster-identifier`— The Aurora Serverless v1 Cluster DB yang Anda konversi ke provisioned.
- `--engine-mode` – Gunakan nilai `provisioned`.
- `--allow-engine-mode-change`
- `--db-cluster-instance-class`— Pilih kelas instans DB untuk cluster DB yang disediakan berdasarkan kapasitas Aurora Serverless v1 klaster DB.

Dalam contoh ini, Anda mengonversi sebuah Aurora Serverless v1 DB cluster bernama `sample-cluster` dan menggunakan kelas instans `db.r5.xlarge` DB.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--engine-mode provisioned \
--allow-engine-mode-change \
--db-cluster-instance-class db.r5.xlarge
```

Untuk Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine-mode provisioned ^
--allow-engine-mode-change ^
--db-cluster-instance-class db.r5.xlarge
```

Contoh berikut menunjukkan cara menggunakan Amazon RDS API untuk mengonversi Aurora Serverless v1 Cluster DB ke cluster yang disediakan.

API RDS

Untuk mengonversi sebuah Aurora Serverless v1 Cluster DB ke cluster yang disediakan, gunakan operasi Modify DBCluster API.

Parameter-parameter berikut diperlukan:

- `DBClusterIdentifier`— The Aurora Serverless v1 Cluster DB yang Anda konversi ke provisioned.
- `EngineMode` – Gunakan nilai `provisioned`.
- `AllowEngineModeChange`
- `DBClusterInstanceClass`— Pilih kelas instans DB untuk cluster DB yang disediakan berdasarkan kapasitas Aurora Serverless v1 klaster DB.

## Pertimbangan saat mengonversi dari Aurora Serverless v1 Cluster DB ke cluster yang disediakan

Pertimbangan berikut berlaku ketika Aurora Serverless v1 Cluster DB dikonversi ke cluster yang disediakan:

- Anda dapat menggunakan konversi ini sebagai bagian dari peningkatan cluster DB Anda dari Aurora Serverless v1 kepada Aurora Serverless v2Untuk informasi selengkapnya, lihat Upgrade dari Aurora Serverless v1 cluster ke Aurora Serverless v2.
- Proses konversi membuat instans DB pembaca di cluster DB, mempromosikan instance pembaca ke instance penulis, dan kemudian menghapus aslinya Aurora Serverless v1 contoh. Saat Anda mengonversi klaster DB, Anda tidak dapat melakukan modifikasi lain secara bersamaan, seperti mengubah versi mesin DB atau grup parameter klaster DB. Operasi konversi akan diterapkan segera, dan tidak dapat dibatalkan.
- Selama konversi, snapshot klaster DB cadangan akan diambil dari klaster DB jika terjadi kesalahan. Pengidentifikasi untuk snapshot klaster DB memiliki form `pre-modify-engine-mode-`*`DB_cluster_identifier`*`-`*`timestamp`*.
- Aurora menggunakan versi mesin minor DB default saat ini untuk klaster DB terprovisi.

- Jika Anda tidak menyediakan kelas instans DB untuk cluster DB yang dikonversi, Aurora merekomendasikan satu berdasarkan kapasitas maksimum yang asli Aurora Serverless v1 klaster DB. Kapasitas yang disarankan untuk pemetaan kelas instans ditunjukkan dalam tabel berikut.

| Serverless kapasitas maksimum (ACUs) | Kelas instans DB terprovisi |
|---|---|
| 1 | db.t3.small |
| 2 | db.t3.medium |
| 4 | db.t3.large |
| 8 | db.r5.large |
| 16 | db.r5.xlarge |
| 32 | db.r5.2xlarge |
| 64 | db.r5.4xlarge |
| 128 | db.r5.8xlarge |
| 192 | db.r5.12xlarge |
| 256 | db.r5.16xlarge |
| 384 | db.r5.24xlarge |

> **Note**
>
> Bergantung pada kelas instans DB yang Anda pilih, dan penggunaan database Anda, Anda mungkin melihat biaya yang berbeda untuk kluster DB yang disediakan dibandingkan dengan Aurora Serverless v1.
>
> Jika Anda mengonversi Aurora Serverless v1 Cluster DB ke kelas instans DB burstable (db.t*), Anda mungkin dikenakan biaya tambahan untuk menggunakan cluster DB. Lihat informasi yang lebih lengkap di Jenis kelas instans DB.

# Penskalaan Aurora Serverless v1 kapasitas klaster DB secara manual

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Biasanya, Aurora Serverless v1 skala klaster DB berjalan mulus berdasarkan beban kerja. Namun, kapasitas mungkin tidak selalu diskalakan cukup cepat untuk memenuhi kondisi ekstrem yang tiba-tiba, seperti peningkatan transaksi secara eksponensial. Dalam kasus seperti itu, Anda dapat memulai operasi penskalaan secara manual dengan menyetel nilai kapasitas baru. Setelah Anda mengatur kapasitas secara eksplisit, Aurora Serverless v1 secara otomatis menskalakan klaster DB. Itu dilakukan berdasarkan periode pendinginan untuk menskalakan ke bawah.

Anda dapat secara eksplisit mengatur kapasitas Aurora Serverless v1 DB cluster ke nilai tertentu dengan AWS Management Console, AWS CLI, atau RDS API.

## Konsol

Anda dapat mengatur kapasitas klaster DB Aurora dengan AWS Management Console.

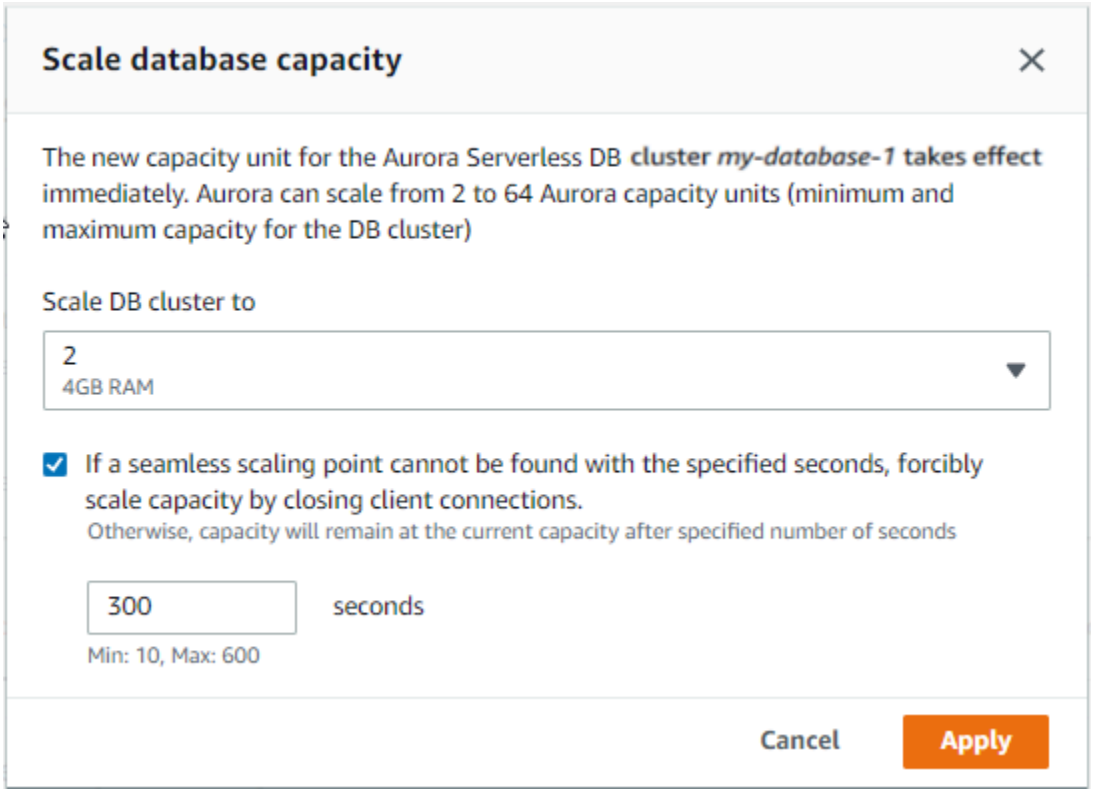
Untuk mengubah Aurora Serverless v1 Klaster DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih Aurora Serverless v1 klaster DB yang ingin Anda ubah.
4. Untuk Actions, pilih Set capacity.
5. Di jendela Skalakan kapasitas basis data, pilih yang berikut:
    a. Untuk pemilih drop-down Skalakan klaster DB ke, pilih kapasitas baru yang Anda inginkan untuk klaster DB Anda.

b. Untuk kotak centang Jika titik penskalaan yang mulus tidak dapat ditemukan, pilih perilaku yang Anda inginkan untuk Anda Aurora Serverless v1 pengaturan `TimeoutAction` klaster DB, sebagai berikut:

   - Hapus opsi ini jika Anda ingin kapasitas Anda tetap tidak berubah jika Aurora Serverless v1 tidak dapat menemukan titik skala sebelum waktu habis.
   - Pilih opsi ini jika Anda ingin memaksakan Aurora Serverless v1 klaster DB mengubah kapasitasnya meskipun tidak dapat menemukan titik penskalaan sebelum waktu habis. Opsi ini dapat mengakibatkan Aurora Serverless v1 menjatuhkan koneksi yang mencegahnya menemukan titik skala.

c. Untuk beberapa detik, masukkan jumlah waktu yang ingin Anda izinkan Aurora Serverless v1 klaster DB Anda untuk mencari titik penskalaan sebelum waktu habis. Anda dapat menentukan di mana saja dari 10 detik hingga 600 detik (10 menit). Default-nya adalah lima menit (300 detik). Contoh berikut ini memaksa Aurora Serverless v1 Cluster DB untuk menurunkan skala ke 2 ACUs bahkan jika tidak dapat menemukan titik penskalaan dalam waktu lima menit.




6. Pilih Terapkan.

Untuk mempelajari selengkapnya tentang titik penskalaan, `TimeoutAction`, dan periode pendinginan, lihat Penskalaan otomatis untuk Aurora Serverless v1.

### AWS CLI

Untuk mengatur kapasitas Aurora Serverless v1 Cluster DB menggunakan AWS CLI, jalankan AWS CLI perintah modify-current-db-cluster-capacity, dan tentukan `--capacity` opsi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Dalam contoh ini, Anda mengatur kapasitas Aurora Serverless v1 Cluster DB dinamai *`sample-cluster64`*.

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

### API RDS

Anda dapat mengatur kapasitas cluster Aurora DB dengan operasi ModifyCurrentDBClusterCapacity API. Tentukan parameter `Capacity`. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

## Melihat Aurora Serverless v1 klaster DB

**Important**

AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi kluster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora

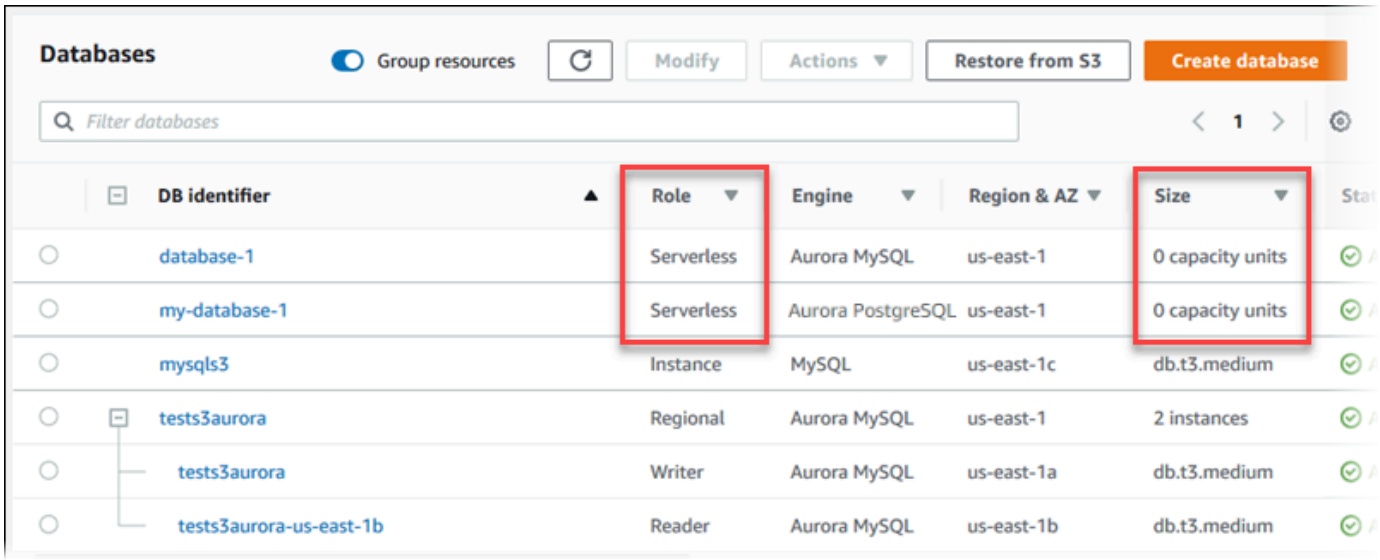
akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Setelah Anda membuat satu atau lebih Aurora Serverless v1 klaster DB, Anda dapat melihat cluster DB mana yang berjenis Serverless dan yang mana berjenis Instance. Anda juga dapat melihat jumlah unit kapasitas Aurora saat ini () masing-masing ACUs Aurora Serverless v1 yang digunakan setiap klaster DB. Setiap ACU merupakan kombinasi dari kapasitas pemrosesan (CPU) dan memori (RAM).

Untuk melihat Aurora Serverless v1 Klaster DB

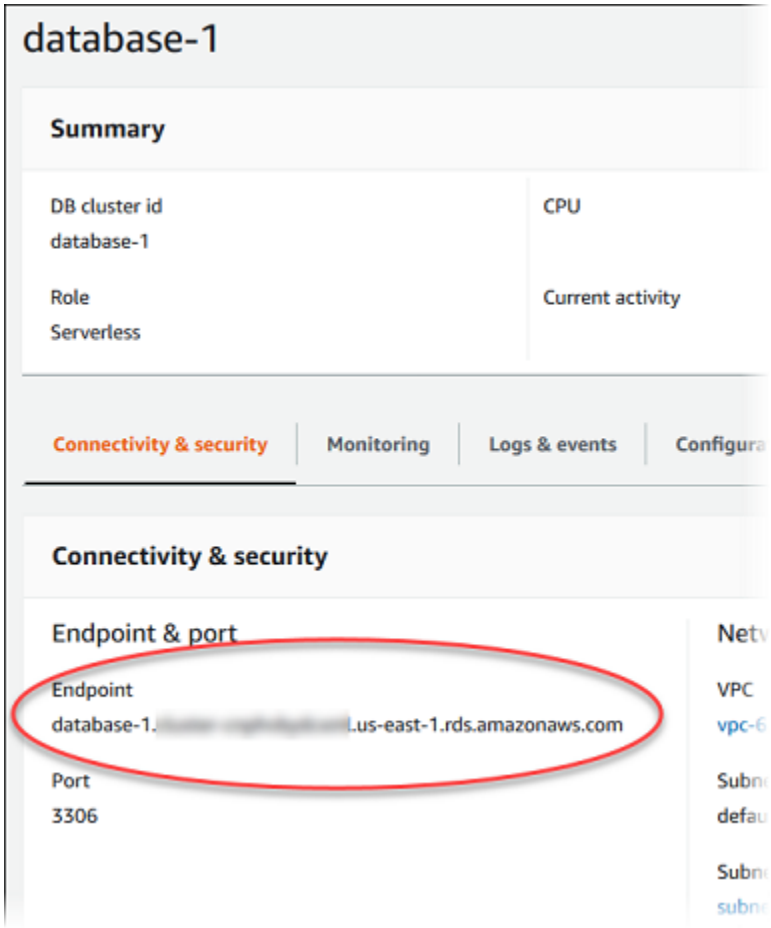
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda membuat Aurora Serverless v1 klaster DB.
3. Di panel navigasi, pilih Basis data.

   Untuk setiap klaster DB, jenis klaster DB ditampilkan pada Role. Titik Aurora Serverless v1 DB memperlihatkan Serverless sebagai tipe. Anda dapat melihat Aurora Serverless v1 kapasitas klaster DB saat ini pada Size.

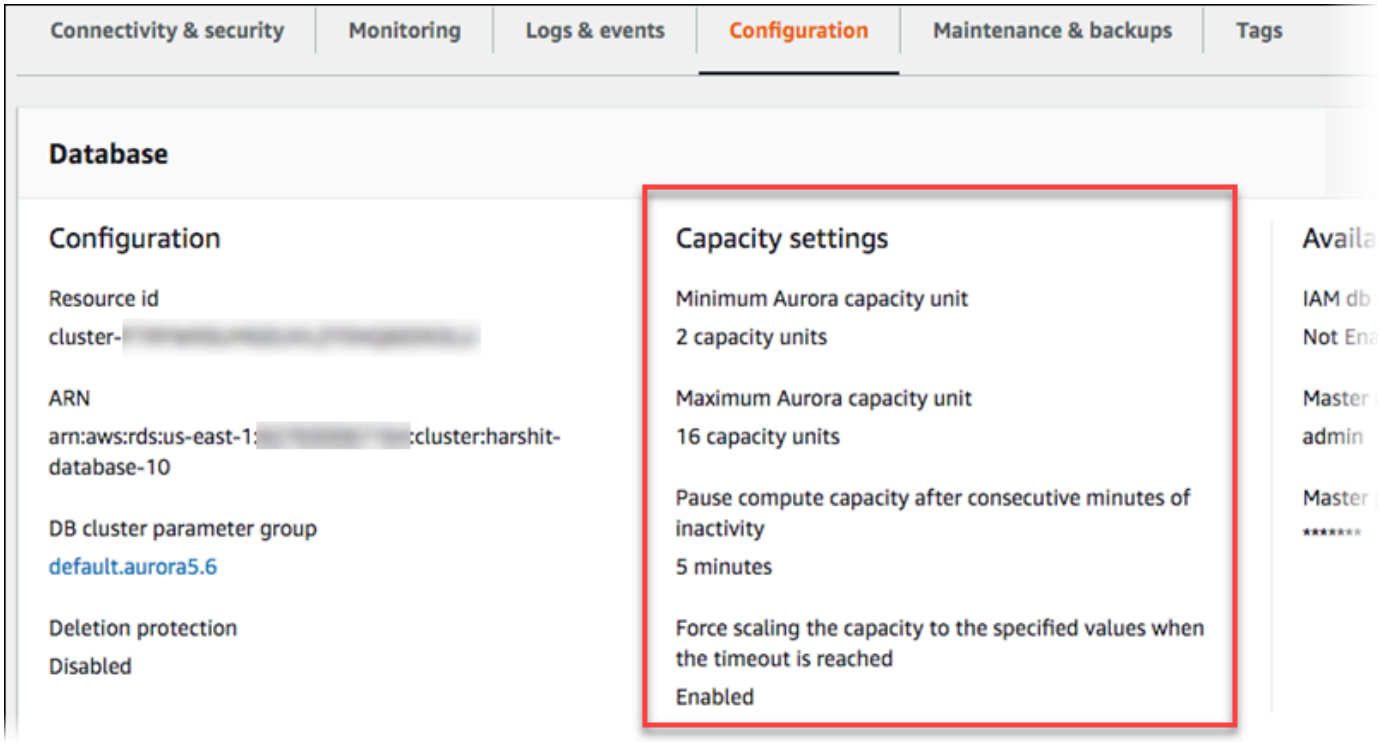



4. Pilih nama Aurora Serverless v1 klaster DB untuk menampilkan detailnya.

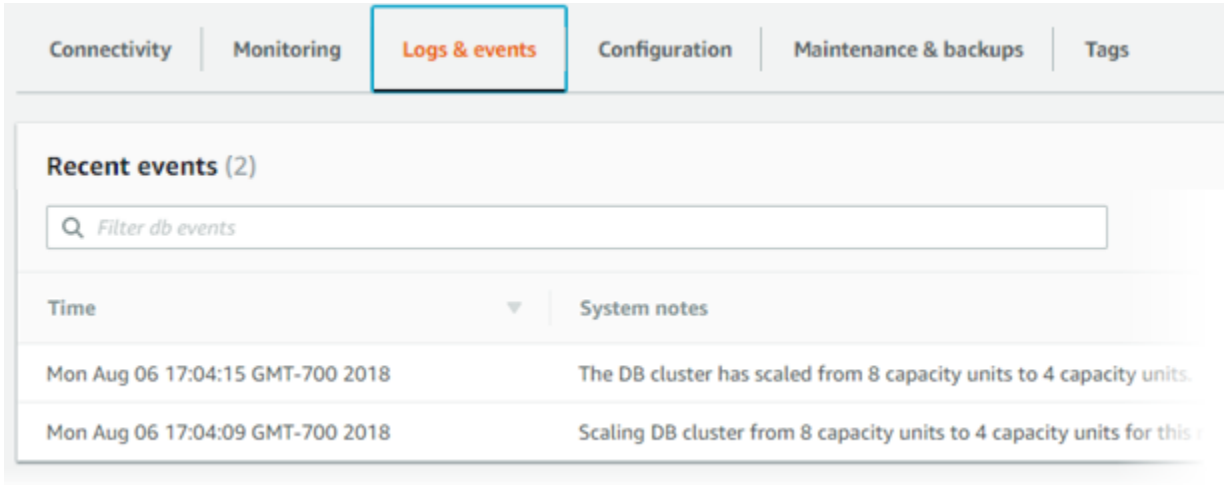
   Di tab Connectivity & security, perhatikan titik akhir basis data. Gunakan titik akhir ini untuk terhubung ke Aurora Serverless v1 klaster DB.




Pilih tab Configuration untuk melihat pengaturan kapasitas.




Suatu scaling event dihasilkan ketika klaster DB meningkatkan, menurunkan, menjeda, atau melanjutkan. Pilih tab Log & peristiwa untuk melihat peristiwa terbaru. Gambar berikut menunjukkan contoh peristiwa tersebut.




## Memantau kapasitas dan peristiwa penskalaan Aurora Serverless v1 klaster DB

Anda dapat melihat Aurora Serverless v1 Cluster DB CloudWatch untuk memantau kapasitas yang dialokasikan ke cluster DB dengan `ServerlessDatabaseCapacity` metrik. Anda

juga dapat memantau semua metrik Aurora CloudWatch standar, seperti `CPUUtilization`, `DatabaseConnections`, `Queries`, dan sebagainya.

Anda dapat meminta Aurora mempublikasikan beberapa atau semua log database. CloudWatch Anda memilih log yang akan diterbitkan dengan mengaktifkan parameter konfigurasi seperti `general_log` dan `slow_query_log` dalam grup parameter cluster DB yang terkait denganAurora Serverless v1 klaster. Tidak seperti klaster yang disediakan, Aurora Serverless v1 klaster tidak perlu Anda tentukan dalam pengaturan klaster DB yang jenis lognya akan diunggah ke CloudWatch. Aurora Serverless v1 klaster secara otomatis mengunggah semua log yang tersedia. Saat Anda menonaktifkan parameter konfigurasi log, penerbitan log akan CloudWatch berhenti. Anda juga dapat menghapus log CloudWatch jika tidak lagi diperlukan.

Untuk memulai dengan Amazon CloudWatch untuk Anda Aurora Serverless v1 Cluster DB, lihatMelihat Aurora Serverless v1 log dengan Amazon CloudWatch. Untuk mempelajari lebih lanjut tentang cara memantau klaster Aurora DB, lihat CloudWatch. Memantau peristiwa log di Amazon CloudWatch

Untuk terhubung ke Aurora Serverless v1 klaster DB, gunakan titik akhir basis data. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Amazon Aurora.

> **Note**
>
> Anda tidak dapat terhubung langsung ke instans DB tertentu di Aurora Serverless v1 klaster DB.

## Menghapus sebuah Aurora Serverless v1 Klaster DB

> **Important**
>
> AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Tergantung pada bagaimana Anda membuat Aurora Serverless v1 Cluster DB, perlindungan penghapusan mungkin diaktifkan secara default. Anda tidak dapat segera menghapus Aurora Serverless v1 Cluster DB yang memiliki perlindungan Penghapusan diaktifkan. Untuk menghapus Aurora Serverless v1 Cluster DB yang memiliki perlindungan penghapusan dengan menggunakan AWS Management Console, Anda terlebih dahulu memodifikasi cluster untuk menghapus perlindungan ini. Untuk informasi tentang penggunaan AWS CLI untuk tugas ini, lihatAWS CLI.

Untuk menonaktifkan perlindungan penghapusan menggunakan AWS Management Console

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Klaster DB.
3. Pilih Anda Aurora Serverless v1 Cluster DB dari daftar.
4. Pilih Modifikasi untuk membuka konfigurasi klaster DB Anda. Halaman cluster Modify DB membuka Pengaturan, pengaturan Kapasitas, dan detail konfigurasi lainnya untuk Anda Aurora Serverless v1 klaster DB. Perlindungan penghapusan ada di bagian Konfigurasi tambahan.
5. Kosongkan kotak centang Aktifkan perlindungan penghapusan di kartu properti Konfigurasi tambahan.
6. Pilih Lanjutkan. Ringkasan modifikasi akan muncul.
7. Pilih Ubah klaster untuk menerima ringkasan modifikasi tersebut. Anda juga dapat memilih Kembali untuk memodifikasi perubahan atau Batalkan untuk menghapus perubahan Anda.

Setelah perlindungan penghapusan tidak lagi aktif, Anda dapat menghapus Aurora Serverless v1 Cluster DB dengan menggunakan AWS Management Console.

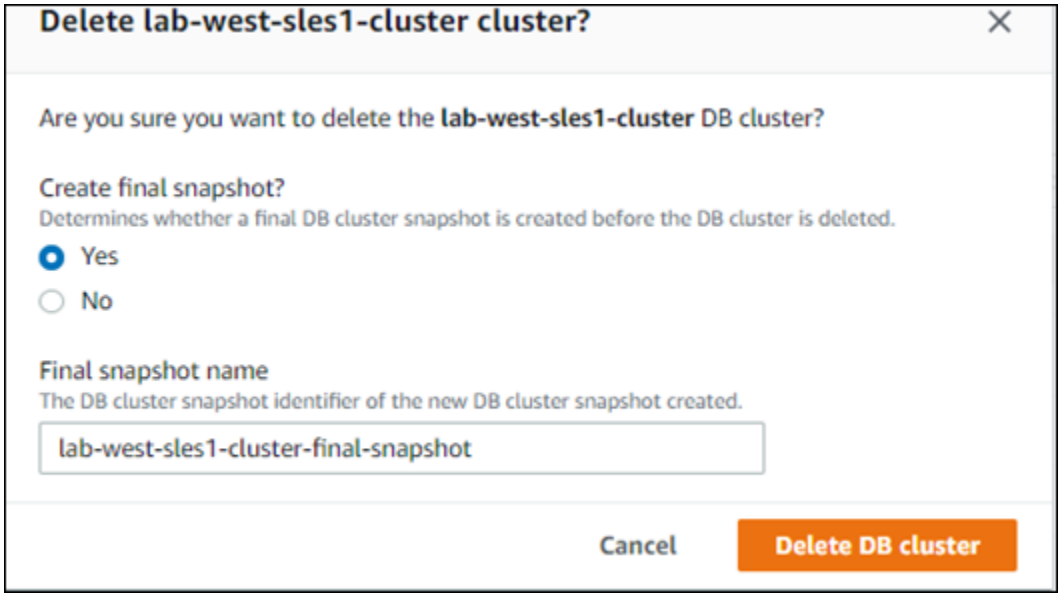
## Konsol

Untuk menghapus sebuah Aurora Serverless v1 Klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di bagian Sumber Daya, pilih Klaster DB.
3. Pilih Aurora Serverless v1 Cluster DB yang ingin Anda hapus.
4. Untuk Tindakan, pilih Hapus. Anda diminta untuk mengonfirmasi bahwa Anda ingin menghapus Aurora Serverless v1 klaster DB.

5. Kami menyarankan agar Anda tetap mempertahankan opsi yang telah dipilih:

   - Ya untuk Buat snapshot akhir?
   - Klaster Aurora Serverless v1 Nama cluster DB plus `-final-snapshot` untuk nama snapshot Akhir. Namun, Anda dapat mengubah nama untuk snapshot akhir Anda di bidang ini.




   Jika Anda memilih Tidak untuk Buat snapshot akhir? Anda tidak dapat memulihkan cluster DB Anda menggunakan snapshot atau point-in-time pemulihan.

6. Pilih Hapus klaster DB.

Aurora Serverless v1 menghapus cluster DB Anda. Jika Anda memilih untuk memiliki snapshot akhir, Anda melihat Aurora Serverless v1 Status cluster DB berubah menjadi “Pencadangan” sebelum dihapus dan tidak lagi muncul dalam daftar.

## AWS CLI

Sebelum memulai, konfigurasikan AWS CLI dengan ID Kunci AWS Akses, Kunci Akses AWS Rahasia, dan Wilayah AWS tempat Anda Aurora Serverless v1 Cluster DB adalah. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi di Panduan AWS Command Line Interface Pengguna.

Anda tidak dapat menghapus Aurora Serverless v1 Cluster DB hingga setelah Anda pertama kali menonaktifkan perlindungan penghapusan untuk cluster yang dikonfigurasi dengan opsi ini. Jika Anda mencoba menghapus klaster yang opsi perlindungannya diaktifkan, Anda akan melihat pesan kesalahan berikut ini.

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

Anda dapat mengubah Aurora Serverless v1 Pengaturan perlindungan penghapusan cluster DB dengan menggunakan modify-db-cluster AWS CLI perintah seperti yang ditunjukkan dalam berikut ini:

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

Perintah ini menampilkan properti yang direvisi untuk klaster DB yang ditentukan. Anda sekarang dapat menghapus Aurora Serverless v1 klaster DB.

Kami menyarankan agar Anda selalu membuat snapshot akhir setiap kali Anda menghapus Aurora Serverless v1 klaster DB. Contoh penggunaan berikut AWS CLI delete-db-clustermenunjukkan caranya. Anda memberi nama klaster DB Anda dan nama untuk snapshot.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

Untuk Windows:

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

# Aurora Serverless v1 dan versi mesin basis data Aurora

**Important**

AWS telah mengumumkan end-of-life tanggal untuk Aurora Serverless v1: 31 Maret 2025. Semua Aurora Serverless v1 cluster yang tidak dimigrasikan pada 31 Maret 2025 akan dimigrasikan ke Aurora Serverless v2 selama jendela pemeliharaan. Jika pemutakhiran gagal, Amazon Aurora mengonversi kluster v1 Tanpa Server menjadi klaster yang disediakan dengan versi engine yang setara selama jendela pemeliharaan. Jika berlaku, Amazon Aurora

akan mendaftarkan klaster yang telah dikonversi di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Dukungan yang Diperluas RDS.

Aurora Serverless v1 tersedia dalam versi tertentu Wilayah AWS dan untuk Aurora MySQL dan Aurora PostgreSQL tertentu saja. Untuk daftar dukungan Wilayah AWS itu saat ini Aurora Serverless v1 dan versi Aurora MySQL dan Aurora PostgreSQL spesifik yang tersedia di setiap Wilayah, lihat. Aurora Serverless v1

Aurora Serverless v1 menggunakan mesin basis data Aurora terkait untuk mengidentifikasi rilis spesifik yang didukung untuk setiap mesin basis data yang didukung, sebagai berikut:

- Aurora MySQL Nirserver
- Aurora PostgreSQL Nirserver

Saat rilis minor engine basis data tersedia untuk Aurora Serverless v1, mereka diterapkan secara otomatis di berbagai Wilayah AWS tempat Aurora Serverless v1 tersedia. Dengan kata lain, Anda tidak perlu meningkatkan Aurora Serverless v1 klaster DB untuk mendapatkan rilis minor baru untuk engine DB klaster Anda jika tersedia untuk Aurora Serverless v1.

## Aurora MySQL Nirserver

Aurora Serverless v1 tersedia dalam versi MySQL Aurora tertentu Wilayah AWS dan khusus saja. Untuk daftar dukungan Wilayah AWS itu saat ini Aurora Serverless v1 dan versi MySQL Aurora spesifik yang tersedia di setiap Wilayah, lihat. Aurora Serverless v1 dengan Aurora MySQL

Untuk mempelajari penyempurnaan dan perbaikan bug untuk Aurora MySQL versi 2, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2 dalam Catatan Rilis untuk Aurora MySQL.

Untuk menggunakan versi terbaru dari Aurora MySQL, Anda dapat menggunakan Aurora Serverless v2. Untuk versi MySQL Wilayah AWS dan Aurora yang dapat Anda gunakan Aurora Serverless v2, lihat Aurora Serverless v2 dengan Aurora MySQL. Untuk informasi penggunaan tentang Aurora Serverless v2, lihat Menggunakan Aurora Serverless v2.

## Aurora PostgreSQL Nirserver

Aurora Serverless v1 tersedia dalam versi Aurora PostgreSQL tertentu Wilayah AWS dan khusus saja. Untuk daftar dukungan Wilayah AWS itu saat ini Aurora Serverless v1 dan versi PostgreSQL

Aurora spesifik yang tersedia di setiap Wilayah, lihat. Aurora Serverless v1 dengan Aurora PostgreSQL

Jika Anda ingin menggunakan Aurora PostgreSQL untuk Aurora Serverless v1 Cluster DB, Anda dapat menggunakan versi Aurora PostgreSQL 13 yang kompatibel. Rilis minor untuk Aurora Edisi Kompatibel PostgreSQL hanya mencakup perubahan yang memiliki kompatibilitas mundur. Klaster Aurora Serverless v1 Cluster DB ditingkatkan secara transparan ketika rilis minor Aurora PostgreSQL tersedia untuk Aurora Serverless v1 di dalam Anda Wilayah AWS.

Untuk menggunakan versi Aurora PostgreSQL yang lebih baru, Anda dapat menggunakan Aurora Serverless v2. Untuk versi PostgreSQL Wilayah AWS dan Aurora yang dapat Anda gunakan Aurora Serverless v2, lihat Aurora Serverless v2 dengan Aurora PostgreSQL. Untuk informasi penggunaan tentang Aurora Serverless v2, lihat Menggunakan Aurora Serverless v2.

## Upgrade versi minor otomatis untuk Aurora Serverless v1

Saat rilis minor engine basis data tersedia untuk Aurora Serverless v1, mereka diterapkan secara otomatis di berbagai Wilayah AWS tempat Aurora Serverless v1 tersedia. Dengan kata lain, Anda tidak perlu meningkatkan Aurora Serverless v1 klaster DB untuk mendapatkan rilis minor baru untuk engine DB klaster Anda jika tersedia untuk Aurora Serverless v1.

# Menggunakan Amazon RDS Data API

Dengan menggunakan RDS Data API (Data API), Anda dapat bekerja dengan antarmuka layanan web ke cluster Aurora DB Anda. Data API tidak memerlukan koneksi persisten ke cluster DB. Sebaliknya, ia menyediakan titik akhir HTTP yang aman dan integrasi dengan AWS SDKs. Anda dapat menggunakan titik akhir untuk menjalankan pernyataan SQL tanpa mengelola koneksi.

Pengguna tidak perlu meneruskan kredensional dengan panggilan ke Data API, karena Data API menggunakan kredenal database yang disimpan di dalamnya. AWS Secrets Manager Untuk menyimpan kredensi di Secrets Manager, pengguna harus diberikan izin yang sesuai untuk menggunakan Secrets Manager, dan juga Data API. Lihat informasi selengkapnya tentang mengizinkan pengguna di Mengotorisasi akses ke Amazon RDS Data API.

Anda juga dapat menggunakan Data API untuk mengintegrasikan Amazon Aurora dengan AWS aplikasi lain seperti AWS Lambda, AWS AppSync, dan. AWS Cloud9 Data API menyediakan cara yang lebih aman untuk digunakan AWS Lambda. API ini memungkinkan Anda mengakses klaster basis data tanpa perlu mengonfigurasi fungsi Lambda untuk mengakses sumber daya di cloud privat virtual (VPC). Lihat informasi selengkapnya di AWS Lambda, AWS AppSync, dan AWS Cloud9.

Anda dapat mengaktifkan Data API saat membuat cluster Aurora DB. Anda juga dapat mengubah konfigurasinya di lain waktu. Untuk informasi selengkapnya, lihat Mengaktifkan API Data Amazon RDS.

Setelah mengaktifkan Data API, Anda juga dapat menggunakan editor kueri untuk menjalankan kueri ad hoc tanpa mengonfigurasi alat kueri untuk mengakses Aurora di VPC. Lihat informasi yang lebih lengkap di Menggunakan editor kueri Aurora.

Topik


- Ketersediaan wilayah dan versi untuk Amazon RDS Data API
- Batasan untuk Amazon RDS Data API
- Membandingkan perilaku Amazon RDS Data API untuk Aurora Serverless v2 dan klaster yang disediakan dengan cluster Aurora Serverless v1
- Mengotorisasi akses ke Amazon RDS Data API
- Mengaktifkan API Data Amazon RDS
- Membuat titik akhir VPC Amazon untuk Amazon RDS Data API ()AWS PrivateLink
- Memanggil Amazon RDS Data API

- Menggunakan pustaka klien Java untuk RDS Data API
- Memproses hasil kueri Amazon RDS Data API dalam format JSON
- Memecahkan Masalah Amazon RDS Data API
- Mencatat panggilan Amazon RDS Data API dengan AWS CloudTrail
- Memantau kueri API Data RDS dengan Performance Insights

## Ketersediaan wilayah dan versi untuk Amazon RDS Data API

Untuk informasi tentang Wilayah dan versi engine yang tersedia untuk Data API, lihat bagian berikut.

| Jenis cluster | Wilayah dan ketersediaan versi |
| --- | --- |
| Aurora PostgreSQL disediakan dan Serverless v2 | Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan |
| Aurora MySQL disediakan dan Serverless v2 | API data dengan Aurora MySQL Serverless v2 dan disediakan |
| Aurora PostgreSQL Tanpa Server v1 | API Data dengan Aurora PostgreSQL Serverless v1 |
| Aurora MySQL Tanpa Server v1 | API Data dengan Aurora MySQL Serverless v1 |

Jika Anda memerlukan modul kriptografi yang divalidasi oleh FIPS 140-2 saat mengakses Data API melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-2.

## Batasan untuk Amazon RDS Data API

RDS Data API memiliki batasan sebagai berikut:

- Anda hanya dapat menjalankan kueri Data API pada instance penulis di cluster DB. Namun, instance penulis dapat menerima kueri tulis dan baca.
- Dengan database global Aurora, Anda dapat mengaktifkan Data API pada cluster DB primer dan sekunder. Namun, cluster sekunder tidak memiliki instance penulis sampai dipromosikan menjadi

yang utama. Data API memerlukan akses ke instance penulis untuk pemrosesan kueri, bahkan untuk kueri baca. Akibatnya, kueri baca dan tulis yang dikirim ke cluster sekunder gagal sementara tidak memiliki instance penulis. Setelah cluster sekunder dipromosikan dan memiliki instance penulis yang tersedia, kueri Data API pada instans DB tersebut berhasil.

- Data API tidak didukung pada kelas instans T DB.
- Untuk Aurora Serverless v2 dan menyediakan kluster DB, RDS Data API tidak mendukung beberapa tipe data. Untuk daftar tipe yang didukung, lihatthe section called “Data API Aurora Serverless v2 dibandingkan dengan Aurora Serverless v1”.
- Untuk Aurora PostgreSQL versi 14 dan database yang lebih tinggi, Data API hanya mendukung enkripsi kata sandi. `scram-sha-256`
- Batas ukuran respons adalah 1 MiB. Jika panggilan menghasilkan data respons dengan ukuran lebih dari 1 MiB, panggilan tersebut akan diakhiri.
- UntukAurora Serverless v1, jumlah maksimum permintaan per detik adalah 1.000. Untuk semua database lain yang didukung, tidak ada batasan.
- Batas ukuran API Data adalah 64 KB per baris dalam set hasil yang ditampilkan oleh basis data. Pastikan bahwa setiap baris dalam set hasil adalah 64 KB atau kurang.

# Membandingkan perilaku Amazon RDS Data API untuk Aurora Serverless v2 dan klaster yang disediakan dengan cluster Aurora Serverless v1

Peningkatan terbaru untuk Amazon RDS Data APIs membuat Data APIs tersedia untuk cluster yang menggunakan versi terbaru dari PostgreSQL atau mesin MySQL. Cluster ini dapat dikonfigurasi untuk menggunakan Aurora Serverless v2 atau menyediakan kelas instance seperti atau. `db.r6g` `db.r6i`

Bagian berikut menjelaskan perbedaan Amazon RDS Data API antara Aurora Serverless v2 dan kluster DB yang disediakan, dan kluster DB. Aurora Serverless v1 Aurora Serverless v1 Cluster DB menggunakan mode `serverless` mesin. Cluster DB yang disediakan menggunakan mode mesin. `provisioned` Cluster Aurora Serverless v2 DB juga menggunakan mode `provisioned` engine, dan berisi satu atau lebih instance Aurora Serverless v2 DB dengan kelas `db.serverless` instance.

## Jumlah maksimum permintaan per detik

Aurora Serverless v1

Data APIs dapat membuat hingga 1.000 permintaan per detik.

Aurora Serverless v2

Data APIs dapat membuat jumlah permintaan per detik yang tidak terbatas.

## Mengaktifkan atau menonaktifkan Amazon RDS Data API pada database yang ada

Aurora Serverless v1

- Dengan Amazon RDS API — Gunakan `ModifyCluster` operasi dan tentukan `True` atau`False`, sebagaimana berlaku, untuk `EnableHttpEndpoint` parameter.
- Dengan AWS CLI — Gunakan `modify-db-cluster` operasi dengan `--no-enable-http-endpoint` opsi `--enable-http-endpoint` atau, sebagaimana berlaku.

Aurora Serverless v2

- Dengan Amazon RDS API — Gunakan `EnableHttpEndpoint` dan `DisableHttpEndpoint` operasi.
- Dengan:Gunakan dan AWS CLI`enable-http-endpoint` operasi. `disable-http-endpoint`

## CloudTrail acara

Aurora Serverless v1

Peristiwa dari panggilan API Data adalah peristiwa manajemen. Peristiwa ini secara otomatis disertakan dalam jejak secara default. Untuk informasi selengkapnya, lihat the section called “Mengecualikan peristiwa Data API dari CloudTrail jejak (Aurora Serverless v1hanya)”.

Aurora Serverless v2

Peristiwa dari panggilan API Data adalah peristiwa data. Peristiwa ini secara otomatis dikecualikan dalam jejak secara default. Untuk informasi selengkapnya, lihat the section called “Menyertakan peristiwa API Data dalam suatu CloudTrail jejak”.

# Dukungan multistatement

Aurora Serverless v1

- Untuk Aurora MySQL, multistatement tidak didukung.
- Untuk Aurora PostgreSQL, multistatement hanya mengembalikan respons kueri pertama.

Aurora Serverless v2

Multistatement tidak didukung. Mencoba mengeksekusi beberapa pernyataan dalam satu panggilan API kembali`“An error occurred (ValidationException) when calling the ExecuteStatement operation: Multistatements aren't supported.”`. Untuk mengeksekusi beberapa pernyataan, buat panggilan `ExecuteStatement` API terpisah atau gunakan `BatchExecuteStatement` untuk pemrosesan batch.

Contoh berikut menunjukkan pesan kesalahan yang dihasilkan dari panggilan API yang mencoba mengeksekusi multistatement.

```
 aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table; Select * FROM next_table;

                                "An error occurred (ValidationException) when calling
 the ExecuteStatement operation: Multistatements aren't supported.
```

Contoh berikut mengeksekusi beberapa pernyataan dengan panggilan `ExecuteStatement` API terpisah.

```
aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table;"

aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
```

```
    --sql "SELECT * FROM next_table;"
```

## Permintaan bersamaan untuk ID transaksi yang sama

Aurora Serverless v1

Permintaan selanjutnya menunggu hingga permintaan saat ini selesai. Aplikasi Anda perlu menangani kesalahan batas waktu jika masa tunggu terlalu lama.

Aurora Serverless v2

Ketika Data API menerima beberapa permintaan dengan ID transaksi yang sama, ia segera mengembalikan kesalahan ini:

`DatabaseErrorException: Transaction is still running a query`

Kesalahan ini terjadi dalam dua situasi:

- Aplikasi Anda membuat permintaan asinkron (seperti JavaScript janji) menggunakan ID transaksi yang sama.
- Permintaan sebelumnya dengan ID transaksi tersebut masih diproses.

Contoh berikut menunjukkan semua permintaan dieksekusi secara paralel dengan`promise.all()`.

```
const api_calls = [];
for (let i = 0; i < 10; i++) {
api_calls.push(
    client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    )
);
}
await Promise.all(api_calls);
```

Untuk mengatasi kesalahan ini, tunggu permintaan saat ini selesai sebelum mengirim permintaan lain dengan ID transaksi yang sama atau hapus ID transaksi untuk mengizinkan permintaan paralel.

Contoh berikut menunjukkan panggilan API yang menggunakan eksekusi sekuensial dengan ID transaksi yang sama.

```
 for (let i = 0; i < 10; i++) {
    await client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    ).promise()
);
}
```

## BatchExecuteStatement perilaku

Untuk informasi selengkapnya tentang `BatchExecuteStatement`, lihat BatchExecuteStatement.

Aurora Serverless v1

Objek bidang yang dihasilkan dalam hasil pembaruan mencakup nilai yang disisipkan.

Aurora Serverless v2

- Untuk Aurora MySQL, objek bidang yang dihasilkan dalam hasil pembaruan mencakup nilai yang disisipkan.
- Untuk Aurora PostgreSQL, objek bidang yang dihasilkan kosong.

## Eksekusi perilaku QL

Untuk informasi selengkapnya`ExecuteSQL`, lihat ExecuteSQL.

Aurora Serverless v1

`ExecuteSQL`Operasi ini tidak digunakan lagi.

Aurora Serverless v2

`ExecuteSQL`Operasi tidak didukung.

## ExecuteStatement perilaku

Untuk informasi selengkapnya tentang `ExecuteStatement`, lihat ExecuteStatement.

Aurora Serverless v1

`ExecuteStatement`Parameter mendukung pengambilan kolom array multidimensi dan semua tipe data lanjutan.

Aurora Serverless v2

`ExecuteStatement`Parameter tidak mendukung kolom array multidimensi. Ini juga tidak mendukung tipe data PostgreSQL tertentu, termasuk tipe geometris dan moneter. Ketika API Data menemukan tipe data yang tidak didukung, ia mengembalikan kesalahan ini—. `UnsupportedResultException: The result contains the unsupported data type data_type`

Untuk mengatasi masalah ini, kirimkan tipe data yang tidak didukung ke`TEXT`. Contoh berikut melemparkan tipe data yang tidak didukung ke. `TEXT`

```
SELECT custom_type::TEXT FROM my_table;--
ORSELECT CAST(custom_type AS TEXT) FROM my_table;
```

Untuk daftar tipe data yang didukung untuk setiap mesin database Aurora, lihat Referensi operasi API Data.

## Perilaku parameter skema

Aurora Serverless v1

`Schema`Paramater tidak didukung. Saat Anda menyertakan `Schema` parameter dalam panggilan API, API Data akan mengabaikan parameter tersebut.

Aurora Serverless v2

`Schema`Parameter tidak digunakan lagi. Saat Anda menyertakan `Schema` parameter dalam panggilan API, API Data mengembalikan kesalahan ini—`ValidationException: The schema parameter isn't supported`. Contoh berikut menunjukkan panggilan Data API yang mengembalikan `ValidationException` kesalahan.

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
```

```
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--schema "your_schema" \
--sql "SELECT * FROM your_table LIMIT 10"
```

Untuk mengatasi masalah ini, hapus `Schema` parameter dari panggilan API Anda.

Contoh berikut menunjukkan panggilan Data API dengan `Schema` parameter dihapus.

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--sql "SELECT * FROM your_table LIMIT 10"
```

# Mengotorisasi akses ke Amazon RDS Data API

Pengguna dapat menjalankan operasi Amazon RDS Data API (Data API) hanya jika mereka berwenang untuk melakukannya. Anda dapat memberikan izin kepada pengguna untuk menggunakan Data API dengan melampirkan kebijakan AWS Identity and Access Management (IAM) yang menentukan hak istimewa mereka. Anda juga dapat melampirkan kebijakan ini pada peran jika Anda menggunakan peran IAM. Kebijakan AWS terkelola`AmazonRDSDataFullAccess`, termasuk izin untuk API Data.

`AmazonRDSDataFullAccess`Kebijakan ini juga mencakup izin bagi pengguna untuk mendapatkan nilai rahasia. AWS Secrets Manager Pengguna perlu menggunakan Secrets Manager untuk menyimpan rahasia yang dapat mereka gunakan dalam panggilan mereka ke Data API. Menggunakan rahasia berarti bahwa pengguna tidak perlu menyertakan kredensi database untuk sumber daya yang mereka targetkan dalam panggilan mereka ke Data API. Data API secara transparan memanggil Secrets Manager, yang memungkinkan (atau menyangkal) permintaan pengguna untuk rahasia tersebut. Untuk informasi tentang menyiapkan rahasia yang akan digunakan dengan Data API, lihat[Menyimpan kredensi basis data di AWS Secrets Manager](#).

`AmazonRDSDataFullAccess`Kebijakan ini menyediakan akses lengkap (melalui Data API) ke sumber daya. Anda dapat mempersempit cakupan dengan menentukan kebijakan Anda sendiri yang menentukan Amazon Resource Name (ARN) sumber daya.

Misalnya, kebijakan berikut menunjukkan contoh izin minimum yang diperlukan bagi pengguna untuk mengakses Data API untuk klaster DB yang diidentifikasi oleh ARN-nya. Kebijakan tersebut

mencakup izin yang diperlukan untuk mengakses Secrets Manager dan mendapatkan otorisasi ke instans basis data untuk pengguna.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

Kami menyarankan agar Anda menggunakan ARN khusus untuk elemen "Sumber Daya" dalam pernyataan kebijakan Anda (seperti yang ditunjukkan dalam contoh), bukan karakter wildcard (*).

## Bekerja dengan otorisasi berbasis tag

RDS Data API (Data API) dan Secrets Manager keduanya mendukung otorisasi berbasis tag. Tag adalah pasangan nilai kunci yang melabeli sumber daya, seperti klaster RDS, dengan nilai string tambahan, misalnya:

- `environment:production`

- `environment:development`

Anda dapat menerapkan tag ke sumber daya Anda untuk alokasi biaya, dukungan operasi, kontrol akses, dan banyak alasan lainnya. (Jika Anda belum memiliki tag pada sumber daya dan ingin menerapkannya, Anda dapat mempelajari lebih lanjut di Memberi tag pada sumber daya Amazon RDS.) Anda dapat menggunakan tag dalam pernyataan kebijakan Anda untuk membatasi akses ke klaster RDS yang dilabeli tag ini. Sebagai contoh, klaster basis data Aurora mungkin memiliki tag yang mengidentifikasi lingkungannya sebagai produksi atau pengembangan.

Contoh-contoh berikut menunjukkan bagaimana Anda dapat menggunakan tag dalam pernyataan kebijakan Anda. Pernyataan ini mengharuskan klaster dan rahasia yang diteruskan dalam permintaan API Data memiliki tag `environment:production`.

Begini cara kebijakan diterapkan: Saat pengguna melakukan panggilan menggunakan Data API, permintaan akan dikirim ke layanan. Data API pertama-tama memverifikasi bahwa ARN cluster yang diteruskan dalam permintaan ditandai dengan. `environment:production` API ini kemudian memanggil Secrets Manager untuk mengambil nilai rahasia pengguna dalam permintaan tersebut. Secrets Manager juga memverifikasi bahwa rahasia pengguna ditandai dengan `environment:production`. Jika demikian, API Data kemudian menggunakan nilai yang diambil untuk kata sandi basis data pengguna. Terakhir, jika itu juga benar, permintaan API Data berhasil diinvokasi untuk pengguna.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
               ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                    ]
```

```
                    }
                }
            },
            {
                "Sid": "RDSDataServiceAccess",
                "Effect": "Allow",
                "Action": [
                      "rds-data:*"
                   ],
                "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
                "Condition": {
                       "StringEquals": {
                           "aws:ResourceTag/environment": [
                                           "production"
                                          ]
                    }
                }
            }
        ]
}
```

Contoh menunjukkan tindakan terpisah untuk `rds-data` dan `secretsmanager` untuk Data API dan Secrets Manager. Namun, Anda dapat menggabungkan tindakan dan menentukan kondisi tag dengan berbagai cara untuk mendukung kasus penggunaan spesifik Anda. Lihat informasi selengkapnya di Menggunakan kebijakan berbasis identitas (kebijakan IAM) untuk Secrets Manager.

Di elemen "Kondisi" kebijakan, Anda dapat memilih kunci tag dari beberapa pilihan berikut:

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

Untuk mempelajari lebih lanjut tentang tag sumber daya dan cara menggunakannya`aws:TagKeys`, lihat Mengontrol akses ke AWS sumber daya menggunakan tag sumber daya.

> **Note**
>
> Baik Data API dan AWS Secrets Manager otorisasi pengguna. Jika Anda tidak memiliki izin untuk semua tindakan yang ditentukan dalam kebijakan, Anda akan mendapatkan kesalahan `AccessDeniedException`.

## Menyimpan kredensi basis data di AWS Secrets Manager

Saat Anda memanggil Amazon RDS Data API (Data API), Anda meneruskan kredensi untuk klaster Aurora DB dengan menggunakan rahasia di Secrets Manager. Untuk memberikan kredensial dengan cara ini, Anda menentukan nama rahasia atau Amazon Resource Name (ARN) rahasia tersebut.

Menyimpan kredensial klaster basis data dalam rahasia

1. Gunakan Secrets Manager untuk membuat rahasia yang berisi kredensial untuk klaster basis data Aurora.

   Lihat petunjuknya di Membuat rahasia basis data dalam Panduan Pengguna AWS Secrets Manager .
2. Gunakan konsol Secrets Manager untuk melihat detail rahasia yang Anda buat, atau jalankan `aws secretsmanager describe-secret` AWS CLI perintah.

   Catat nama dan ARN rahasia. Anda dapat menggunakannya dalam panggilan ke Data API.

Lihat informasi selengkapnya tentang menggunakan Secrets Manager di Panduan Pengguna AWS Secrets Manager.

Untuk memahami bagaimana Amazon Aurora mengelola Identity and Access Management, lihat Bagaimana Amazon Aurora bekerja dengan IAM.

Lihat informasi selengkapnya tentang pembuatan kebijakan IAM di Membuat Kebijakan IAM dalam Panduan Pengguna IAM. Lihat informasi tentang cara menambahkan kebijakan IAM ke pengguna di Menambahkan dan Menghapus Izin Identitas IAM dalam Panduan Pengguna IAM.

# Mengaktifkan API Data Amazon RDS

Untuk menggunakan Amazon RDS Data API (Data API), aktifkan untuk klaster Aurora DB Anda. Anda dapat mengaktifkan Data API saat membuat atau memodifikasi cluster DB.

> **Note**
>
> Apakah Data API tersedia untuk klaster Anda tergantung pada versi Aurora, mesin database, dan AWS Wilayah Anda. Untuk versi Aurora yang lebih lama, Data API hanya berfungsi dengan Aurora Serverless v1 cluster. Untuk versi Aurora yang lebih baru, Data API bekerja

dengan cluster yang menggunakan provisioned dan instance. Aurora Serverless v2 Untuk memeriksa apakah klaster Anda dapat menggunakan Data API, lihatDaerah yang Didukung dan engine Aurora DB untuk RDS Data API.

Topik

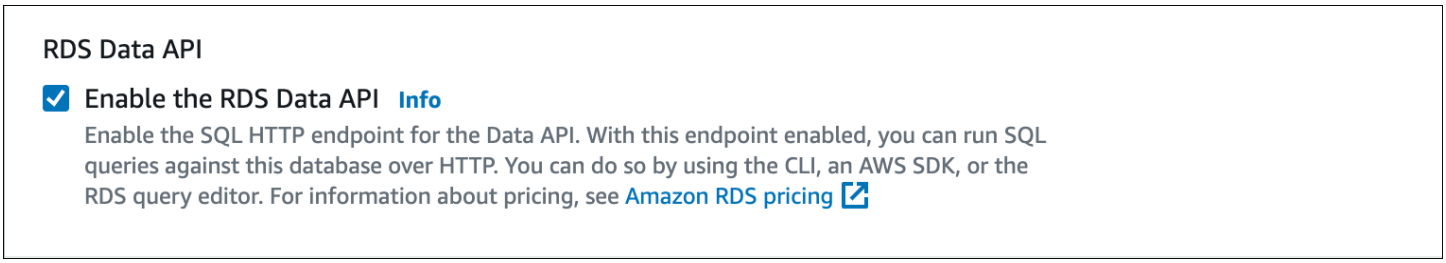
- Mengaktifkan RDS Data API saat Anda membuat database
- Mengaktifkan API Data RDS pada database yang ada

## Mengaktifkan RDS Data API saat Anda membuat database

Saat Anda membuat database yang mendukung RDS Data API (Data API), Anda dapat mengaktifkan fitur ini. Prosedur berikut menjelaskan cara melakukannya ketika Anda menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Untuk mengaktifkan Data API saat Anda membuat cluster DB, pilih kotak centang Aktifkan API Data RDS di bagian Konektivitas pada halaman Buat database, seperti pada tangkapan layar berikut.




Untuk petunjuk tentang cara membuat klaster Aurora DB yang dapat menggunakan API Data RDS, lihat berikut ini:

- Untuk Aurora Serverless v2 dan klaster yang disediakan - Membuat klaster DB Amazon Aurora
- Untuk Aurora Serverless v1 - Membuat Aurora Serverless v1 klaster DB.

AWS CLI

Untuk mengaktifkan Data API saat Anda membuat cluster Aurora DB, jalankan create-db-cluster AWS CLI perintah dengan opsi. `--enable-http-endpoint`

Contoh berikut membuat cluster Aurora PostgreSQL DB dengan Data API diaktifkan.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

API RDS

Untuk mengaktifkan Data API saat Anda membuat cluster Aurora DB, gunakan DBCluster operasi Create dengan nilai `EnableHttpEndpoint` parameter yang disetel ke. `true`

## Mengaktifkan API Data RDS pada database yang ada

Anda dapat memodifikasi cluster DB yang mendukung RDS Data API (Data API) untuk mengaktifkan atau menonaktifkan fitur ini.

Topik

- Mengaktifkan atau menonaktifkan Data API (Aurora Serverless v2dan disediakan)
- Mengaktifkan atau menonaktifkan Data API (hanya) Aurora Serverless v1

### Mengaktifkan atau menonaktifkan Data API (Aurora Serverless v2dan disediakan)

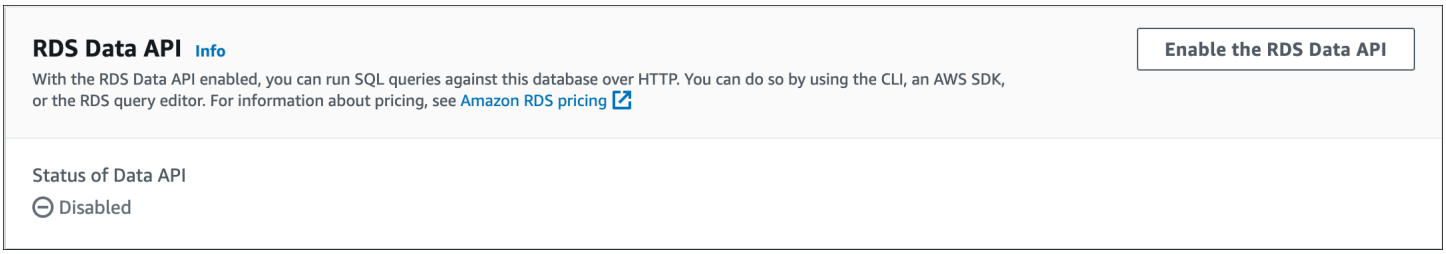
Gunakan prosedur berikut untuk mengaktifkan atau menonaktifkan Data API pada Aurora Serverless v2 dan database yang disediakan. Untuk mengaktifkan atau menonaktifkan Data API pada Aurora Serverless v1 database, gunakan prosedur dithe section called “Mengaktifkan atau menonaktifkan Data API (hanya) Aurora Serverless v1”.

Konsol

Anda dapat mengaktifkan atau menonaktifkan Data API dengan menggunakan konsol RDS untuk cluster DB yang mendukung fitur ini. Untuk melakukannya, buka halaman detail cluster database

tempat Anda ingin mengaktifkan atau menonaktifkan Data API, dan pada tab Konektivitas & keamanan, buka bagian API Data RDS. Bagian ini menampilkan status Data API, dan memungkinkan Anda untuk mengaktifkan atau menonaktifkannya.

Screenshot berikut menunjukkan bahwa RDS Data API tidak diaktifkan.




AWS CLI

Untuk mengaktifkan atau menonaktifkan Data API pada database yang ada, jalankan disable-http-endpoint AWS CLI perintah enable-http-endpointor, dan tentukan ARN cluster DB Anda.

Contoh berikut memungkinkan Data API.

Untuk Linux, macOS, atau Unix:

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

Untuk Windows:

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

API RDS

Untuk mengaktifkan atau menonaktifkan Data API pada database yang ada, gunakan EnableHttpEndpointdan DisableHttpEndpointoperasi.

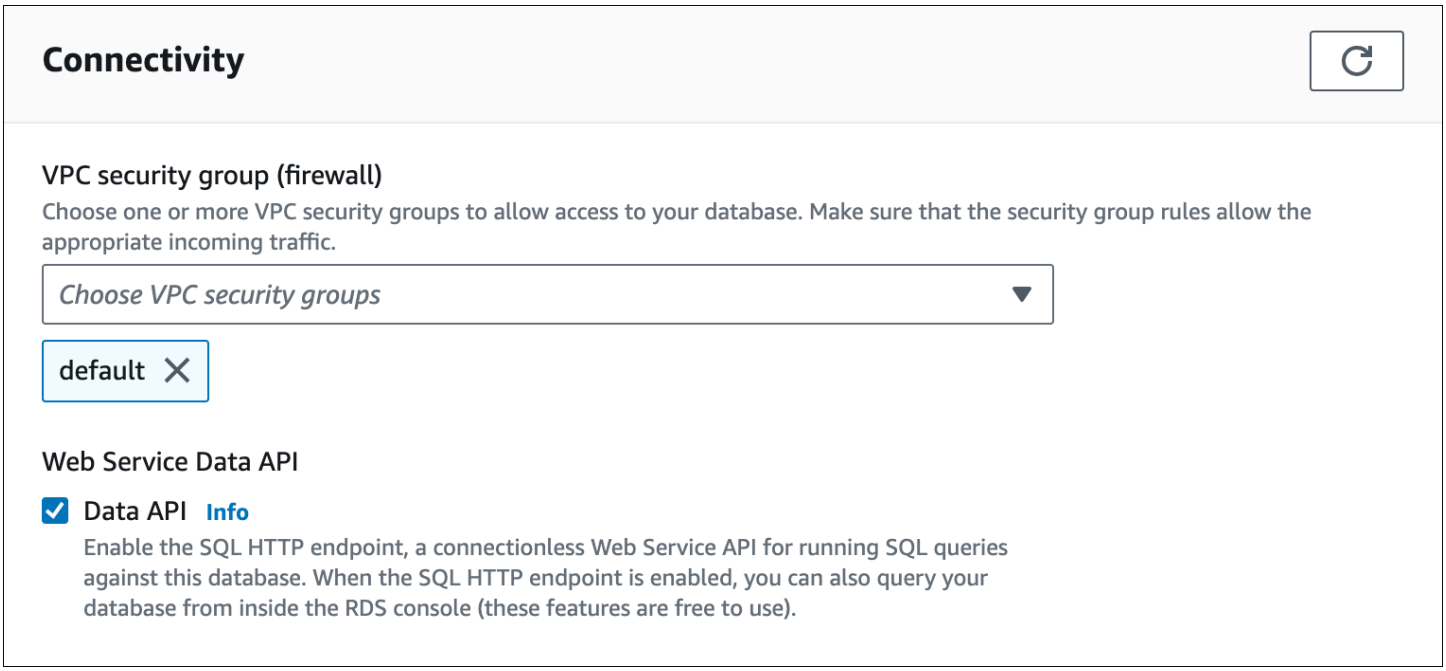
## Mengaktifkan atau menonaktifkan Data API (hanya) Aurora Serverless v1

Gunakan prosedur berikut untuk mengaktifkan atau menonaktifkan Data API pada Aurora Serverless v1 database yang ada. Untuk mengaktifkan atau menonaktifkan Data API pada Aurora Serverless v2 dan database yang disediakan, gunakan prosedur di. the section called “Mengaktifkan atau menonaktifkan Data API”

Konsol

Saat memodifikasi cluster Aurora Serverless v1 DB, Anda mengaktifkan Data API di bagian Konektivitas konsol RDS.

Tangkapan layar berikut menunjukkan API Data yang diaktifkan saat memodifikasi cluster Aurora DB.




Untuk petunjuk tentang cara memodifikasi cluster Aurora Serverless v1 DB, lihatMemodifikasi Aurora Serverless v1 Klaster DB.

AWS CLI

Untuk mengaktifkan atau menonaktifkan Data API, jalankan modify-db-cluster AWS CLI perintah, dengan `--enable-http-endpoint` atau`--no-enable-http-endpoint`, sebagaimana berlaku.

Contoh berikut memungkinkan Data API aktif`sample-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --enable-http-endpoint
```

Untuk Windows:

```
aws rds modify-db-cluster ^
```

```
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

API RDS

Untuk mengaktifkan Data API, gunakan DBCluster operasi Modify, dan tetapkan nilai `EnableHttpEndpoint` ke `true` atau`false`, sebagaimana berlaku.

# Membuat titik akhir VPC Amazon untuk Amazon RDS Data API ()AWS PrivateLink

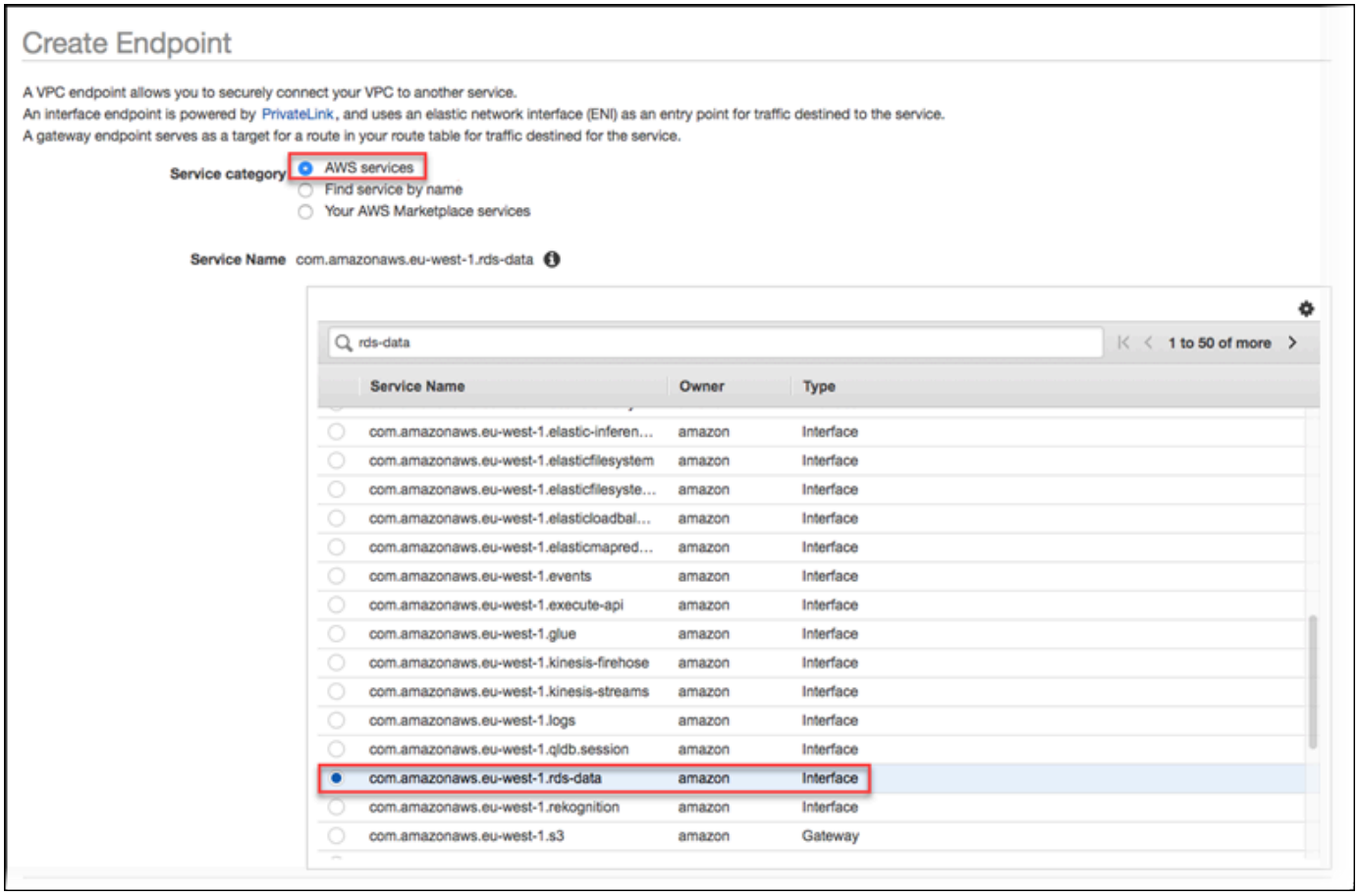
Amazon VPC memungkinkan Anda meluncurkan AWS sumber daya, seperti cluster dan aplikasi Aurora DB, ke cloud pribadi virtual (VPC). AWS PrivateLink menyediakan konektivitas pribadi antara VPCs dan AWS layanan dengan keamanan tinggi di jaringan Amazon. Dengan menggunakan AWS PrivateLink, Anda dapat membuat titik akhir VPC Amazon, yang memungkinkan Anda terhubung ke layanan di berbagai akun dan VPCs berdasarkan Amazon VPC. Lihat informasi selengkapnya tentang AWS PrivateLink di Layanan Titik Akhir VPC (AWS PrivateLink) dalam Panduan Pengguna Amazon Virtual Private Cloud.

Anda dapat memanggil RDS Data API (Data API) dengan titik akhir Amazon VPC. Menggunakan endpoint Amazon VPC menjaga lalu lintas antar aplikasi di Amazon VPC dan Data API di AWS jaringan, tanpa menggunakan alamat IP publik. Titik akhir Amazon VPC dapat membantu Anda memenuhi persyaratan kepatuhan dan peraturan yang berkaitan dengan membatasi konektivitas internet publik. Misalnya, jika Anda menggunakan titik akhir VPC Amazon, Anda dapat menjaga lalu lintas antara aplikasi yang berjalan pada EC2 instans Amazon dan API Data di VPCs dalamnya.

Setelah membuat titik akhir Amazon VPC, Anda dapat mulai menggunakannya tanpa membuat kode atau perubahan konfigurasi apa pun di aplikasi Anda.

Untuk membuat titik akhir VPC Amazon untuk Data API

1. Masuk ke AWS Management Console dan buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
2. Pilih Titik Akhir, lalu pilih Buat Titik Akhir.
3. Di halaman Buat Titik Akhir, untuk Kategori layanan, pilih Layanan AWS . Untuk Nama Layanan, pilih rds-data.




4. Untuk VPC, pilih VPC tempat membuat titik akhir.

   Pilih VPC yang berisi aplikasi yang membuat panggilan API Data.

5. Untuk Subnet, pilih subnet untuk setiap Availability Zone (AZ) yang digunakan oleh AWS layanan yang menjalankan aplikasi Anda.

   Untuk membuat titik akhir Amazon VPC, tentukan rentang alamat IP pribadi yang dapat mengakses titik akhir. Untuk melakukannya, pilih subnet untuk setiap Zona Ketersediaan. Hal tersebut membatasi titik akhir VPC ke rentang alamat IP pribadi khusus untuk setiap Zona Ketersediaan dan membuat titik akhir Amazon VPC di setiap Zona Ketersediaan.

6. Untuk Aktifkan nama DNS, pilih Aktifkan untuk titik akhir ini.

   DNS Pribadi menyelesaikan nama host DNS API Data standar (`https://rds-data.`*`region`*`.amazonaws.com`) ke alamat IP pribadi yang dikaitkan dengan nama host DNS khusus untuk titik akhir Amazon VPC Anda. Akibatnya, Anda dapat mengakses titik akhir VPC API Data menggunakan AWS CLI atau AWS SDKs tanpa membuat perubahan kode atau konfigurasi apa pun untuk memperbarui URL titik akhir Data API.

7. Untuk Grup keamanan, pilih satu grup keamanan untuk dikaitkan dengan titik akhir Amazon VPC.

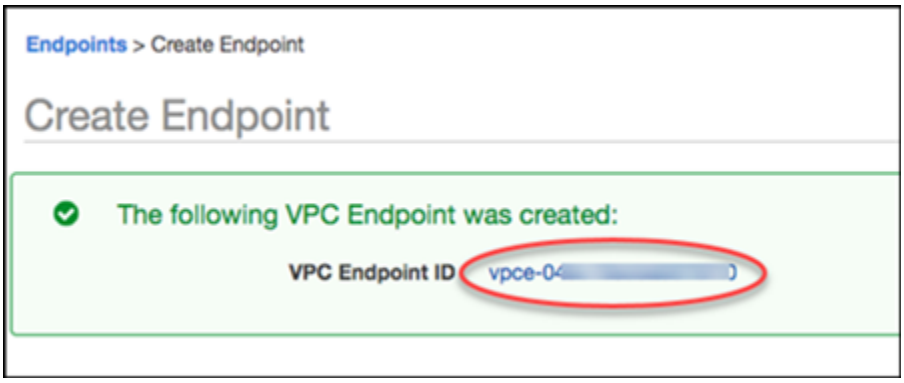
   Pilih grup keamanan yang memungkinkan akses ke AWS layanan yang menjalankan aplikasi Anda. Misalnya, jika EC2 instans Amazon menjalankan aplikasi Anda, pilih grup keamanan yang memungkinkan akses ke EC2 instans Amazon. Grup keamanan memungkinkan Anda mengontrol lalu lintas ke titik akhir Amazon VPC dari sumber daya di VPC Anda.

8. Untuk Kebijakan, pilih Akses Penuh untuk mengizinkan siapa pun di dalam Amazon VPC mengakses API Data melalui titik akhir ini. Atau, pilih Kustom untuk menentukan kebijakan yang membatasi akses.

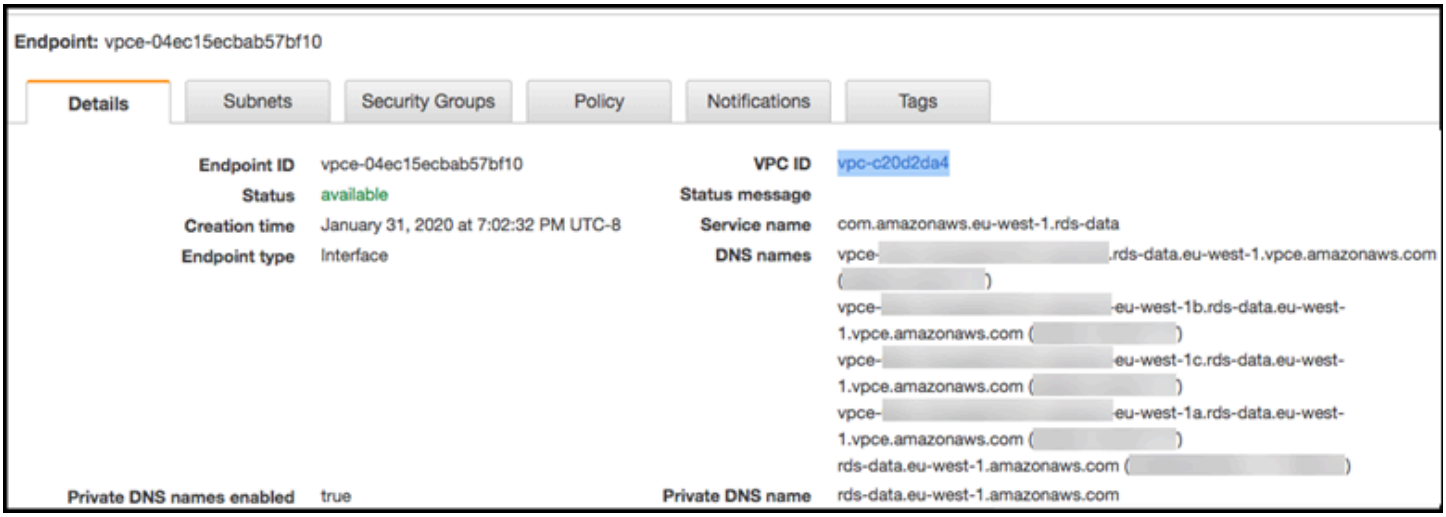
   Jika Anda memilih Kustom, masukkan kebijakan di alat pembuatan kebijakan.

9. Pilih Create endpoint.

Setelah titik akhir dibuat, pilih tautan di AWS Management Console untuk melihat detail titik akhir.




Tab Detail titik akhir menunjukkan nama host DNS yang dibuat saat membuat titik akhir Amazon VPC.




Anda dapat menggunakan titik akhir standar (`rds-data.`*`region`*`.amazonaws.com`) atau salah satu titik akhir khusus VPC untuk memanggil API Data dalam Amazon VPC. Titik akhir API Data standar secara otomatis merutekan ke titik akhir Amazon VPC. Perutean ini terjadi karena nama host DNS Pribadi diaktifkan saat titik akhir Amazon VPC dibuat.

Saat Anda menggunakan titik akhir VPC Amazon dalam panggilan API Data, semua lalu lintas antara aplikasi Anda dan Data API tetap berada di Amazon VPCs yang mengandungnya. Anda dapat menggunakan titik akhir Amazon VPC untuk semua jenis panggilan API Data. Untuk informasi tentang memanggil Data API, lihatMemanggil Amazon RDS Data API.

# Memanggil Amazon RDS Data API

Dengan Amazon RDS Data API (Data API) diaktifkan pada cluster Aurora DB Anda, Anda dapat menjalankan pernyataan SQL pada cluster Aurora DB dengan menggunakan Data API atau file. AWS CLI Data API mendukung bahasa pemrograman yang didukung oleh AWS SDKs. Untuk informasi selengkapnya, lihat Alat untuk dibangun AWS.

Topik

- Referensi operasi Amazon RDS Data API
- Memanggil API Data Amazon RDS dengan AWS CLI
- Memanggil Amazon RDS Data API dari aplikasi Python
- Memanggil Amazon RDS Data API dari aplikasi Java
- Mengontrol perilaku batas waktu API Data

# Referensi operasi Amazon RDS Data API

Amazon RDS Data API menyediakan operasi berikut untuk melakukan pernyataan SQL.

| Operasi API Data | AWS CLI perintah | Deskripsi |
| --- | --- | --- |
| ExecuteStatement | aws rds-data execute-statement | Menjalankan pernyataan SQL pada basis data. |
| BatchExecuteStatement | aws rds-data batch-execute-statement | Menjalankan pernyataan SQL batch pada array data untuk pembaruan massal dan operasi penyisipan. Anda dapat menjalankan pernyataan bahasa manipulasi data (DML) dengan array set parameter. Pernyataan SQL batch dapat memberikan peningkatan performa yang signifikan atas pernyataan penyisipan dan pembaruan individu. |

Anda dapat menggunakan operasi mana pun untuk menjalankan pernyataan SQL individual atau untuk menjalankan transaksi. Untuk transaksi, Data API menyediakan operasi berikut.

| Operasi API Data | AWS CLI perintah | Deskripsi |
| --- | --- | --- |
| BeginTransaction | aws rds-data begin-transaction | Memulai transaksi SQL. |
| CommitTransaction | aws rds-data commit-transaction | Mengakhiri transaksi SQL dan menerapkan perubahan. |
| RollbackTransaction | aws rds-data rollback-transaction | Melakukan pembatalan transaksi. |

Operasi untuk melakukan pernyataan SQL dan transaksi pendukung memiliki parameter dan AWS CLI opsi API Data umum berikut. Beberapa operasi mendukung parameter atau opsi lain.

| Parameter operasi API Data | AWS CLI opsi perintah | Wajib | Deskripsi |
|---|---|---|---|
| `resourceArn` | `--resource-arn` | Ya | Nama Sumber Daya Amazon (ARN) dari cluster Aurora DB. Cluster harus Akun AWS sama dengan peran IAM atau pengguna yang memanggil API Data. Untuk mengakses klaster di akun yang berbeda, ambil peran dalam akun itu. |
| `secretArn` | `--secret-arn` | Ya | Nama atau ARN rahasia yang memungkinkan akses ke klaster basis data. |

RDS Data API mendukung tipe data berikut untuk Aurora MySQL:

- `TINYINT(1)`, `BOOLEAN`, `BOOL`
- `TINYINT`
- `SMALLINT` [`SIGNED` | `UNSIGNED`]
- `MEDIUMINT` [`SIGNED` | `UNSIGNED`]
- `INT` [`SIGNED` | `UNSIGNED`]
- `BIGINT` [`SIGNED` | `UNSIGNED`]
- `FLOAT`
- `DOUBLE`
- `VARCHAR`, `CHAR`, `TEXT`, `ENUM`
- `VARBINARY`, `BINARY`, `BLOB`
- `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`
- `DECIMAL`
- `JSON`

- BIT, BIT(N)

RDS Data API mendukung jenis skalar Aurora PostgreSQL berikut:

- BOOL
- BYTEA
- DATE
- CIDR
- DECIMAL, NUMERIC
- ENUM
- FLOAT8, DOUBLE PRECISION
- INET
- INT, INT4, SERIAL
- INT2, SMALLINT, SMALLSERIAL
- INT8, BIGINT, BIGSERIAL
- JSONB, JSON
- REAL, FLOAT
- TEXT, CHAR(N), VARCHAR, NAME
- TIME
- TIMESTAMP
- UUID
- VECTOR

RDS Data API mendukung tipe array Aurora PostgreSQL berikut:

- BOOL[], BIT[]
- DATE[]
- DECIMAL[], NUMERIC[]
- FLOAT8[], DOUBLE PRECISION[]
- INT[], INT4[]
- INT2[]
- INT8[], BIGINT[]

- `JSON[]`
- `REAL[]`, `FLOAT[]`
- `TEXT[]`, `CHAR(N)[]`, `VARCHAR[]`, `NAME[]`
- `TIME[]`
- `TIMESTAMP[]`
- `UUID[]`

Anda dapat menggunakan parameter dalam panggilan Data API ke `ExecuteStatement` dan`BatchExecuteStatement`, dan ketika Anda menjalankan AWS CLI perintah `execute-statement` dan`batch-execute-statement`. Untuk menggunakan parameter, tentukan pasangan nama-nilai di tipe data `SqlParameter`. Tentukan nilai dengan tipe data `Field`. Tabel berikut memetakan tipe data Java Database Connectivity (JDBC) ke tipe data yang Anda tentukan dalam panggilan API Data.

| Tipe data JDBC | Tipe data API Data |
|---|---|
| `INTEGER, TINYINT, SMALLINT, BIGINT` | `LONG` (atau `STRING`) |
| `FLOAT, REAL, DOUBLE` | `DOUBLE` |
| `DECIMAL` | `STRING` |
| `BOOLEAN, BIT` | `BOOLEAN` |
| `BLOB, BINARY, LONGVARBINARY, VARBINARY` | `BLOB` |
| `CLOB` | `STRING` |
| Tipe lainnya (termasuk tipe terkait tanggal dan waktu) | `STRING` |

> **Note**
>
> Anda dapat menentukan tipe data `LONG` atau `STRING` dalam panggilan API Data Anda untuk nilai `LONG` yang dihasilkan oleh basis data. Kami menyarankan Anda melakukannya untuk

menghindari kehilangan presisi untuk jumlah yang sangat besar, yang dapat terjadi ketika Anda bekerja dengan JavaScript.

Tipe tertentu, seperti `DECIMAL` dan`TIME`, memerlukan petunjuk agar Data API meneruskan `String` nilai ke database sebagai tipe yang benar. Untuk menggunakan petunjuk, sertakan nilai untuk `typeHint` di tipe data `SqlParameter`. Berikut adalah nilai-nilai yang mungkin untuk `typeHint`:

- `DATE` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `DATE` ke basis data. Format yang diterima adalah `YYYY-MM-DD`.
- `DECIMAL` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `DECIMAL` ke basis data.
- `JSON` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `JSON` ke basis data.
- `TIME` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `TIME` ke basis data. Format yang diterima adalah `HH:MM:SS[.FFF]`.
- `TIMESTAMP` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `TIMESTAMP` ke basis data. Format yang diterima adalah `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `UUID` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `UUID` ke basis data.

> **Note**
>
> Saat ini, Data API tidak mendukung array Universal Unique Identifiers ()UUIDs.

> **Note**
>
> Untuk Amazon Aurora PostgreSQL, Data API selalu menampilkan tipe data Aurora PostgreSQL di zona waktu UTC. `TIMESTAMPTZ`

## Memanggil API Data Amazon RDS dengan AWS CLI

Anda dapat memanggil RDS Data API (Data API) menggunakan file. AWS CLI

Contoh berikut menggunakan AWS CLI for Data API. Lihat informasi selengkapnya di Referensi AWS CLI untuk API Data.

Dalam setiap contoh, ganti Amazon Resource Name (ARN) untuk cluster DB dengan ARN untuk cluster Aurora DB Anda. Selain itu, ganti ARN rahasia dengan ARN rahasia di Secrets Manager yang mengizinkan akses ke klaster basis data.

> **Note**
>
> Kaleng AWS CLI memformat tanggapan di JSON.

Topik

- Memulai transaksi SQL
- Menjalankan pernyataan SQL
- Menjalankan pernyataan SQL batch pada array data
- Menerapkan transaksi SQL
- Membatalkan transaksi SQL

## Memulai transaksi SQL

Anda dapat memulai transaksi SQL menggunakan perintah CLI `aws rds-data begin-transaction`. Panggilan ini menghasilkan pengidentifikasi transaksi.

> **Important**
>
> Dalam Data API, transaksi akan habis jika tidak ada panggilan yang menggunakan ID transaksinya dalam tiga menit. Jika waktu transaksi habis sebelum dilakukan, Data API akan mengembalikannya secara otomatis.
> Pernyataan bahasa definisi data MySQL (DDL) di dalam transaksi menyebabkan komit implisit. Kami menyarankan Anda menjalankan setiap pernyataan MySQL DDL dalam perintah `execute-statement` terpisah dengan opsi. `--continue-after-timeout`

Selain opsi umum, tentukan opsi `--database` yang menyediakan nama basis data.

Misalnya, perintah CLI berikut memulai transaksi SQL.

Untuk Linux, macOS, atau Unix:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Untuk Windows:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Berikut adalah contoh respons tersebut.

```
{
    "transactionId": "ABC1234567890xyz"
}
```

## Menjalankan pernyataan SQL

Anda dapat menjalankan pernyataan SQL menggunakan perintah CLI `aws rds-data execute-statement`.

Anda dapat menjalankan pernyataan SQL dalam transaksi dengan menentukan pengidentifikasi transaksi dengan opsi `--transaction-id`. Anda dapat memulai transaksi menggunakan perintah CLI `aws rds-data begin-transaction`. Anda dapat mengakhiri dan menerapkan transaksi menggunakan perintah CLI `aws rds-data commit-transaction`.

> **Important**
>
> Jika Anda tidak menentukan opsi `--transaction-id`, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--sql` (wajib) – Pernyataan SQL untuk dijalankan pada klaster basis data.
- `--transaction-id` (opsional) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda sertakan pernyataan SQL-nya.

- `--parameters` (opsional) – Parameter untuk pernyataan SQL.
- `--include-result-metadata | --no-include-result-metadata` (opsional) – Nilai yang menunjukkan apakah metadata disertakan dalam hasil. Nilai default-nya `--no-include-result-metadata`.
- `--database` (opsional) – Nama basis data.

  Opsi `--database` mungkin tidak berfungsi ketika Anda menjalankan pernyataan SQL setelah menjalankan `--sql "use database_name;"` di permintaan sebelumnya. Kami merekomendasikan agar Anda menggunakan `--database` opsi, bukan menjalankan pernyataan `--sql "use database_name;"`.
- `--continue-after-timeout | --no-continue-after-timeout`(opsional) — Nilai yang menunjukkan apakah akan terus menjalankan pernyataan setelah panggilan melebihi interval batas waktu API Data 45 detik. Nilai default-nya `--no-continue-after-timeout`.

  Untuk pernyataan bahasa definisi data (DDL), kami merekomendasikan agar Anda terus menjalankan pernyataan setelah waktu panggilan habis untuk menghindari kesalahan dan kemungkinan struktur data rusak.
- `--format-records-as "JSON"|"NONE"` – Nilai opsional yang menentukan apakah hasil yang ditetapkan akan diformat sebagai string JSON. Nilai default-nya `"NONE"`. Lihat informasi penggunaan tentang pemrosesan set hasil JSON di Memproses hasil kueri Amazon RDS Data API dalam format JSON.

Klaster basis data menghasilkan respons untuk panggilan.

> **Note**
>
> Batas ukuran respons adalah 1 MiB. Jika panggilan menghasilkan data respons dengan ukuran lebih dari 1 MiB, panggilan tersebut akan diakhiri.
> UntukAurora Serverless v1, jumlah maksimum permintaan per detik adalah 1.000. Untuk semua database lain yang didukung, tidak ada batasan.

Misalnya, perintah CLI berikut menjalankan satu pernyataan SQL dan menghilangkan metadata dalam hasil (default).

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

Berikut adalah contoh respons tersebut.

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
        ],
        [
            {
                "longValue": 3
            },
            {
                "stringValue": "ValueThree"
            }
```

```
        ]
    ]
}
```

Perintah CLI berikut menjalankan satu pernyataan SQL dalam transaksi dengan menentukan opsi `--transaction-id`.

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Berikut adalah contoh respons tersebut.

```
{
    "numberOfRecordsUpdated": 1
}
```

Perintah CLI berikut menjalankan satu pernyataan SQL dengan parameter.

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Berikut adalah contoh respons tersebut.

```
{
    "numberOfRecordsUpdated": 1
}
```

Perintah CLI berikut menjalankan pernyataan SQL bahasa definisi data (DDL). Pernyataan DDL mengganti nama kolom `job` menjadi kolom `role`.

> **Important**
>
> Untuk pernyataan DDL, kami merekomendasikan agar Anda terus menjalankan pernyataan tersebut setelah waktu panggilan habis. Ketika pernyataan DDL berakhir sebelum selesai dijalankan, itu dapat mengakibatkan kesalahan dan kemungkinan struktur data rusak. Untuk terus menjalankan pernyataan setelah panggilan melebihi interval batas waktu API Data RDS 45 detik, tentukan opsi. `--continue-after-timeout`

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Berikut adalah contoh respons tersebut.

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

> **Note**
>
> Data `generatedFields` tidak didukung oleh Aurora PostgreSQL. Untuk mendapatkan nilai-nilai bidang yang dihasilkan, gunakan klausa `RETURNING`. Lihat informasi selengkapnya di Returning data from modified rows dalam dokumentasi PostgreSQL.

## Menjalankan pernyataan SQL batch pada array data

Anda dapat menjalankan pernyataan SQL batch pada array data dengan menggunakan perintah CLI `aws rds-data batch-execute-statement`. Anda dapat menggunakan perintah ini untuk melakukan impor massal atau operasi pembaruan.

Anda dapat menjalankan pernyataan SQL dalam transaksi dengan menentukan pengidentifikasi transaksi dengan opsi `--transaction-id`. Anda dapat memulai transaksi menggunakan perintah CLI `aws rds-data begin-transaction`. Anda dapat mengakhiri dan menerapkan transaksi menggunakan perintah CLI `aws rds-data commit-transaction`.

> **Important**
>
> Jika Anda tidak menentukan opsi `--transaction-id`, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--sql` (wajib) – Pernyataan SQL untuk dijalankan pada klaster basis data.

> **Tip**
>
> Untuk pernyataan yang kompatibel dengan MySQL, jangan sertakan titik koma di akhir parameter `--sql`. Titik koma di belakang dapat menyebabkan kesalahan sintaksis.

- `--transaction-id` (opsional) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda sertakan pernyataan SQL-nya.
- `--parameter-set` (opsional) – Set parameter untuk operasi batch.
- `--database` (opsional) – Nama basis data.

Klaster basis data menghasilkan respons untuk panggilan.

> **Note**
>
> Tidak ada batas atas tetap pada jumlah set parameter. Namun, ukuran maksimum permintaan HTTP yang dikirimkan melalui Data API adalah 4 MiB. Jika permintaan melebihi batas ini, Data API mengembalikan kesalahan dan tidak memproses permintaan. Batas 4 MiB ini mencakup ukuran header HTTP dan notasi JSON dalam permintaan. Dengan demikian, jumlah set parameter yang dapat Anda sertakan tergantung pada kombinasi faktor, seperti ukuran pernyataan SQL dan ukuran setiap set parameter.
>
> Batas ukuran respons adalah 1 MiB. Jika panggilan menghasilkan data respons dengan ukuran lebih dari 1 MiB, panggilan tersebut akan diakhiri.
>
> UntukAurora Serverless v1, jumlah maksimum permintaan per detik adalah 1.000. Untuk semua database lain yang didukung, tidak ada batasan.

Misalnya, perintah CLI berikut menjalankan pernyataan SQL batch pada array data dengan satu set parameter.

Untuk Linux, macOS, atau Unix:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
```

```
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

Untuk Windows:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> Jangan sertakan jeda baris di opsi `--parameter-sets`.

## Menerapkan transaksi SQL

Dengan menggunakan perintah CLI `aws rds-data commit-transaction`, Anda dapat mengakhiri transaksi SQL yang Anda mulai dengan `aws rds-data begin-transaction` dan menerapkan perubahan.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--transaction-id` (wajib) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda akhiri dan terapkan.

Misalnya, perintah CLI berikut mengakhiri transaksi SQL dan menerapkan perubahan.

Untuk Linux, macOS, atau Unix:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Untuk Windows:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

Berikut adalah contoh respons tersebut.

```
{
    "transactionStatus": "Transaction Committed"
}
```

## Membatalkan transaksi SQL

Dengan menggunakan perintah CLI `aws rds-data rollback-transaction`, Anda dapat membatalkan transaksi SQL yang Anda mulai dengan `aws rds-data begin-transaction`. Membatalkan transaksi akan membatalkan perubahannya.

> **Important**
>
> Jika ID transaksi telah kedaluwarsa, transaksi dibatalkan secara otomatis. Dalam kasus ini, perintah `aws rds-data rollback-transaction` yang menentukan ID transaksi yang kedaluwarsa akan menghasilkan kesalahan.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--transaction-id` (wajib) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda batalkan.

Misalnya, AWS CLI perintah berikut memutar kembali transaksi SQL.

Untuk Linux, macOS, atau Unix:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Untuk Windows:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

Berikut adalah contoh respons tersebut.

```
{
    "transactionStatus": "Rollback Complete"
    }
```

## Memanggil Amazon RDS Data API dari aplikasi Python

Anda dapat memanggil Amazon RDS Data API (Data API) dari aplikasi Python.

Contoh berikut menggunakan AWS SDK untuk Python (Boto). Lihat informasi selengkapnya tentang Boto, lihat Dokumentasi AWS SDK for Python (Boto 3).

Dalam setiap contoh, ganti Amazon Resource Name (ARN) cluster DB dengan ARN untuk cluster Aurora DB Anda. Selain itu, ganti ARN rahasia dengan ARN rahasia di Secrets Manager yang mengizinkan akses ke klaster basis data.

Topik

- Menjalankan kueri SQL
- Menjalankan pernyataan SQL DML
- Menjalankan transaksi SQL

### Menjalankan kueri SQL

Anda dapat menjalankan pernyataan `SELECT` dan mengambil hasilnya dengan aplikasi Python.

Contoh berikut menjalankan kueri SQL.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
            resourceArn = cluster_arn,
            secretArn = secret_arn,
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
```

```
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

## Menjalankan pernyataan SQL DML

Anda dapat menjalankan pernyataan bahasa manipulasi data (DML) untuk memasukkan, memperbarui, atau menghapus data dalam basis data Anda. Anda juga dapat menggunakan parameter dalam pernyataan DML.

> **Important**
>
> Jika panggilan bukan bagian dari transaksi karena tidak menyertakan parameter `transactionID`, perubahan yang dihasilkan dari panggilan tersebut diterapkan secara otomatis.

Contoh berikut menjalankan pernyataan penyisipan SQL dan menggunakan parameter.

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
```

```
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

## Menjalankan transaksi SQL

Anda dapat memulai transaksi SQL, menjalankan satu atau beberapa pernyataan SQL, lalu menerapkan perubahan dengan aplikasi Python.

> **Important**
>
> Waktu transaksi habis jika tidak ada panggilan yang menggunakan ID transaksinya dalam tiga menit. Jika waktu transaksi habis sebelum diterapkan, transaksi akan dibatalkan secara otomatis.
>
> Jika Anda tidak menentukan ID transaksi, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Contoh berikut menjalankan transaksi SQL yang menyisipkan baris dalam tabel.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
     resourceArn = cluster_arn,
```

```
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

> **Note**
>
> Jika Anda menjalankan pernyataan bahasa definisi data (DDL), kami merekomendasikan agar Anda terus menjalankan pernyataan setelah waktu panggilan habis. Ketika pernyataan DDL dihentikan sebelum selesai dijalankan, hal ini dapat mengakibatkan kesalahan dan kemungkinan struktur data rusak. Untuk terus menjalankan pernyataan setelah panggilan melebihi interval batas waktu API Data RDS 45 detik, setel `continueAfterTimeout` parameter ke. `true`

## Memanggil Amazon RDS Data API dari aplikasi Java

Anda dapat memanggil Amazon RDS Data API (Data API) dari aplikasi Java.

Contoh berikut menggunakan AWS SDK for Java. Lihat informasi selengkapnya di Panduan Developer AWS SDK untuk Java.

Dalam setiap contoh, ganti Amazon Resource Name (ARN) cluster DB dengan ARN untuk cluster Aurora DB Anda. Selain itu, ganti ARN rahasia dengan ARN rahasia di Secrets Manager yang mengizinkan akses ke klaster basis data.

Topik

- Menjalankan kueri SQL
- Menjalankan transaksi SQL

- Menjalankan operasi SQL batch

## Menjalankan kueri SQL

Anda dapat menjalankan pernyataan SELECT dan mengambil hasilnya dengan aplikasi Java.

Contoh berikut menjalankan kueri SQL.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
```

```
}
```

## Menjalankan transaksi SQL

Anda dapat memulai transaksi SQL, menjalankan satu atau beberapa pernyataan SQL, lalu menerapkan perubahan dengan aplikasi Java.

> **Important**
>
> Waktu transaksi habis jika tidak ada panggilan yang menggunakan ID transaksinya dalam tiga menit. Jika waktu transaksi habis sebelum diterapkan, transaksi akan dibatalkan secara otomatis.
>
> Jika Anda tidak menentukan ID transaksi, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Contoh berikut menjalankan transaksi SQL.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
```

```
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);

    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> Jika Anda menjalankan pernyataan bahasa definisi data (DDL), kami merekomendasikan agar Anda terus menjalankan pernyataan setelah waktu panggilan habis. Ketika pernyataan DDL dihentikan sebelum selesai dijalankan, hal ini dapat mengakibatkan kesalahan dan kemungkinan struktur data rusak. Untuk terus menjalankan pernyataan setelah panggilan melebihi interval batas waktu API Data RDS 45 detik, setel `continueAfterTimeout` parameter ke. `true`

## Menjalankan operasi SQL batch

Anda dapat menjalankan operasi penyisipan dan pembaruan massal pada array data dengan aplikasi Java. Anda dapat menjalankan pernyataan DML dengan array set parameter.

> **Important**
>
> Jika Anda tidak menentukan ID transaksi, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Contoh berikut menjalankan operasi penyisipan batch.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

# Mengontrol perilaku batas waktu API Data

Semua panggilan ke API Data bersifat sinkron. Misalkan Anda melakukan operasi API Data yang menjalankan pernyataan SQL seperti `INSERT` atau`CREATE TABLE`. Jika panggilan Data API berhasil kembali, pemrosesan SQL selesai saat panggilan kembali.

Secara default, Data API membatalkan operasi dan mengembalikan kesalahan batas waktu jika operasi tidak selesai diproses dalam waktu 45 detik. Dalam hal ini, data tidak dimasukkan, tabel tidak dibuat, dan sebagainya.

Anda dapat menggunakan Data API untuk melakukan operasi jangka panjang yang tidak dapat diselesaikan dalam waktu 45 detik. Jika Anda berharap bahwa operasi seperti operasi massal `INSERT` atau DDL pada tabel besar membutuhkan waktu lebih dari 45 detik, Anda dapat menentukan `continueAfterTimeout` parameter untuk `ExecuteStatement` operasi tersebut. Aplikasi Anda masih menerima kesalahan batas waktu. Namun, operasi terus berjalan dan tidak dibatalkan. Sebagai contoh, lihat Menjalankan transaksi SQL.

Jika AWS SDK untuk bahasa pemrograman Anda memiliki periode batas waktu sendiri untuk panggilan API atau koneksi soket HTTP, pastikan bahwa semua periode batas waktu tersebut lebih dari 45 detik. Bagi sebagian orang SDKs, periode batas waktu kurang dari 45 detik secara default. Sebaiknya setel periode batas waktu khusus SDK atau khusus klien menjadi setidaknya satu menit. Melakukannya menghindari kemungkinan bahwa aplikasi Anda menerima kesalahan batas waktu sementara operasi Data API masih berhasil diselesaikan. Dengan begitu, Anda dapat yakin apakah akan mencoba kembali operasi atau tidak.

Misalnya, SDK mengembalikan kesalahan batas waktu ke aplikasi Anda, tetapi operasi Data API masih selesai dalam interval batas waktu API Data. Dalam hal ini, mencoba kembali operasi mungkin menyisipkan data duplikat atau menghasilkan hasil yang salah. SDK mungkin mencoba ulang operasi secara otomatis, menyebabkan data yang salah tanpa tindakan apa pun dari aplikasi Anda.

Interval batas waktu sangat penting untuk SDK Java 2. Dalam SDK tersebut, batas waktu panggilan API dan batas waktu soket HTTP keduanya 30 detik secara default. Berikut adalah contoh pengaturan batas waktu tersebut ke nilai yang lebih tinggi:

```
public RdsDataClient createRdsDataClient() {
    return RdsDataClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .httpClientBuilder(createHttpClientBuilder())
```

```
        .credentialsProvider(defaultCredentialsProvider()) // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return ApacheHttpClient.builder() // Change this to your desired HttpClient
        .socketTimeout(Duration.ofSeconds(60));
}
```

Berikut adalah contoh yang setara menggunakan klien data asinkron:

```
public static RdsDataAsyncClient createRdsDataAsyncClient() {
    return RdsDataAsyncClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .credentialsProvider(defaultCredentialsProvider())  // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallAttemptTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return NettyNioAsyncHttpClient.builder() // Change this to your desired
 AsyncHttpClient
        .readTimeout(Duration.ofSeconds(60));
}
```

# Menggunakan pustaka klien Java untuk RDS Data API

Anda dapat mengunduh dan menggunakan pustaka klien Java untuk RDS Data API (Data API). Pustaka klien Java ini menyediakan cara alternatif untuk menggunakan Data API. Dengan menggunakan pustaka ini, Anda dapat memetakan kelas sisi klien Anda ke permintaan dan respons Data API. Dukungan pemetaan ini dapat memudahkan integrasi dengan beberapa jenis Java tertentu, seperti `Date`, `Time`, dan `BigDecimal`.

## Mengunduh pustaka klien Java untuk API Data

Pustaka klien Java Data API adalah open source GitHub di lokasi berikut:

https://github.com/awslabs/rds-data-api-client-perpustakaan-java

Anda dapat membuat pustaka secara manual dari file sumber, tetapi praktik terbaiknya adalah memanfaatkan pustaka tersebut menggunakan manajemen dependensi Apache Maven. Tambahkan dependensi berikut ke file Maven POM Anda:

Untuk versi 2.x, yang kompatibel dengan AWS SDK 2.x, gunakan yang berikut ini:

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
   <version>2.0.0</version>
</dependency>
```

Untuk versi 1.x, yang kompatibel dengan AWS SDK 1.x, gunakan yang berikut ini:

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

## Contoh pustaka klien Java

Di bawah ini Anda dapat menemukan beberapa contoh umum penggunaan pustaka klien Java API Data. Contoh berikut mengasumsikan bahwa Anda memiliki tabel `accounts` dengan dua kolom: `accountId` dan `name`. Anda juga memiliki data transfer object (DTO) berikut.

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

Pustaka klien memungkinkan Anda untuk lulus DTOs sebagai parameter input. Contoh berikut menunjukkan bagaimana pelanggan DTOs dipetakan ke set parameter input.

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParamSets(account1, account2)
         .execute();
```

Dalam beberapa kasus, bekerja dengan nilai sederhana sebagai parameter input akan terasa lebih mudah. Anda dapat melakukannya dengan sintaksis berikut.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParameter("accountId", 3)
         .withParameter("name", "Zhang")
         .execute();
```

Berikut adalah contoh lain yang bekerja dengan nilai sederhana sebagai parameter input.

```

client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
         .execute();

```

Pustaka klien menyediakan pemetaan otomatis DTOs saat hasil dikembalikan. Contoh berikut menunjukkan bagaimana hasilnya dipetakan ke Anda DTOs.

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

Dalam banyak kasus, kumpulan hasil basis data hanya berisi satu nilai. Untuk menyederhanakan pengambilan hasil tersebut, pustaka klien menawarkan API berikut:

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
          .singleValue(Integer.class);
```

> **Note**
>
> Fungsi `mapToList` mengonversi set hasil SQL ke dalam daftar objek yang ditentukan pengguna. Kami tidak mendukung penggunaan pernyataan `.withFormatRecordsAs(RecordsFormatType.JSON)` dalam panggilan `ExecuteStatement` untuk pustaka klien Java karena fungsinya sama. Lihat informasi yang lebih lengkap di Memproses hasil kueri Amazon RDS Data API dalam format JSON.

# Memproses hasil kueri Amazon RDS Data API dalam format JSON

Ketika Anda memanggil operasi `ExecuteStatement`, Anda dapat memilih untuk menampilkan hasil kueri sebagai string dalam format JSON. Dengan demikian, Anda dapat menggunakan kemampuan parsing JSON bahasa pemrograman Anda untuk menafsirkan dan memformat ulang set hasil. Dengan melakukan ini, Anda dapat membantu menghindari penulisan kode tambahan untuk mengulang set hasil dan menafsirkan setiap nilai kolom.

Untuk meminta set hasil dalam format JSON, teruskan parameter `formatRecordsAs` opsional dengan nilai `JSON`. Set hasil yang diformat JSON ditampilkan di bidang `formattedRecords` struktur `ExecuteStatementResponse`.

Tindakan `BatchExecuteStatement` tidak menampilkan set hasil. Dengan demikian, opsi JSON tidak berlaku untuk tindakan itu.

Untuk menyesuaikan kunci dalam struktur hash JSON, tentukan alias kolom dalam set hasil. Anda dapat melakukannya dengan menggunakan klausa `AS` dalam daftar kolom kueri SQL Anda.

Anda dapat menggunakan kemampuan JSON untuk membuat set hasil lebih mudah dibaca dan memetakan kontennya ke kerangka kerja khusus bahasa. Karena volume set hasil berenkode ASCII lebih besar dari representasi default, Anda dapat memilih representasi default untuk kueri yang menampilkan sejumlah besar baris atau nilai kolom besar yang menghabiskan lebih banyak memori daripada yang tersedia untuk aplikasi Anda.

Topik


- Mengambil hasil kueri dalam format JSON
- Pemetaan Tipe Data
- Pemecahan Masalah
- Contoh


## Mengambil hasil kueri dalam format JSON

Untuk menerima hasil yang ditetapkan sebagai string JSON, sertakan `.withFormatRecordsAs(RecordsFormatType.JSON)` dalam `ExecuteStatement` panggilan. Nilai pengembalian kembali sebagai string JSON di bidang `formattedRecords`. Dalam kasus ini, `columnMetadata` bernilai `null`. Label kolom adalah kunci dari objek yang mewakili setiap baris. Nama kolom ini diulang untuk setiap baris dalam set hasil. Nilai kolom adalah string berkutip, nilai numerik, atau nilai khusus yang merepresentasikan `true`, `false`, atau `null`. Metadata kolom seperti batasan panjang dan tipe yang tepat untuk angka dan string tidak dipertahankan dalam respons JSON.

Jika Anda menghilangkan panggilan `.withFormatRecordsAs()` atau menentukan parameter `NONE`, set hasil ditampilkan dalam format biner menggunakan bidang `Records` dan `columnMetadata`.

## Pemetaan Tipe Data

Nilai SQL dalam set hasil dipetakan ke set tipe JSON yang lebih kecil. Nilai direpresentasikan dalam JSON sebagai string, angka, dan beberapa konstanta khusus seperti `true`, `false`, dan `null`. Anda dapat mengonversi nilai-nilai ini menjadi berbagai variabel dalam aplikasi Anda, dengan menggunakan pengetikan kuat atau lemah yang sesuai untuk bahasa pemrograman Anda.

| Tipe data JDBC | Tipe data JSON |
| --- | --- |
| `INTEGER`, `TINYINT`, `SMALLINT`, `BIGINT` | Angka secara default. String jika opsi `LongReturnType` diatur ke `STRING`. |
| `FLOAT`, `REAL`, `DOUBLE` | Angka |

| Tipe data JDBC | Tipe data JSON |
| --- | --- |
| DECIMAL | String secara default. Angka jika opsi DecimalReturnType diatur ke DOUBLE_OR_LONG . |
| STRING | String |
| BOOLEAN, BIT | Boolean |
| BLOB, BINARY, VARBINARY , LONGVARBINARY | String dalam enkode base64. |
| CLOB | String |
| ARRAY | Array |
| NULL | null |
| Tipe lainnya (termasuk tipe terkait tanggal dan waktu) | String |

## Pemecahan Masalah

Respons JSON dibatasi hingga 10 megabyte. Jika responsnya lebih besar dari batas ini, program Anda menerima kesalahan BadRequestException. Dalam kasus ini, Anda dapat mengatasi kesalahan menggunakan salah satu teknik berikut:

- Kurangi jumlah baris di set hasil. Untuk melakukannya, tambahkan klausa LIMIT. Anda dapat membagi set hasil besar menjadi beberapa set yang lebih kecil dengan mengirimkan beberapa kueri dengan klausa LIMIT dan OFFSET.

  Jika set hasil menyertakan baris yang difilter oleh logika aplikasi, Anda dapat menghapus baris tersebut dari set hasil dengan menambahkan kondisi lainnya dalam klausa WHERE.
- Kurangi jumlah kolom di set hasil. Untuk melakukannya, hapus item dari daftar pilih kueri.
- Persingkat label kolom dengan menggunakan alias kolom dalam kueri. Setiap nama kolom diulang dalam string JSON untuk setiap baris dalam set hasil. Oleh karena itu, hasil kueri dengan nama

kolom panjang dan banyak baris dapat melebihi batas ukuran. Khususnya, gunakan alias kolom untuk ekspresi rumit agar seluruh ekspresi tidak diulang dalam string JSON.

- Meskipun Anda dapat menggunakan alias kolom dengan SQL untuk menghasilkan set hasil yang memiliki beberapa kolom dengan nama yang sama, nama kunci duplikat tidak diperbolehkan di JSON. RDS API Data menampilkan kesalahan jika Anda meminta set hasil dalam format JSON dan beberapa kolom memiliki nama yang sama. Jadi, pastikan semua label kolom memiliki nama yang unik.

## Contoh

Contoh Java berikut menunjukkan cara memanggil `ExecuteStatement` dengan respons sebagai string berformat JSON, lalu menafsirkan set hasil. Gantikan nilai yang sesuai untuk*databaseName*,*secretStoreArn*, dan *clusterArn* parameter.

Contoh Java berikut menunjukkan kueri yang menampilkan nilai numerik desimal di set hasil. Panggilan `assertThat` menguji apakah bidang respons memiliki properti yang diharapkan berdasarkan aturan untuk set hasil JSON.

Contoh ini bekerja dengan skema dan data sampel berikut:

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

Nilai bidang `formattedRecords` dari program sebelumnya adalah:

```
[{"a":10.0}]
```

Bidang `Records` dan `ColumnMetadata` dalam respons keduanya bernilai null karena adanya set hasil JSON.

Contoh Java berikut menunjukkan kueri yang menampilkan nilai numerik integer di set hasil. Contoh panggilan `getFormattedRecords` untuk menampilkan hanya string berformat JSON dan mengabaikan bidang respons lain yang kosong atau bernilai null. Contoh ini melakukan deserialisasi hasil ke dalam struktur yang merepresentasikan daftar catatan. Setiap catatan memiliki bidang yang namanya sesuai dengan alias kolom dari set kumpulan. Teknik ini menyederhanakan kode yang mem-parsing set hasil. Aplikasi Anda tidak harus mengulang baris dan kolom dari set hasil dan mengonversi setiap nilai ke tipe yang sesuai.

Contoh ini bekerja dengan skema dan data sampel berikut:

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

Nilai bidang `formattedRecords` dari program sebelumnya adalah:

```
[{"a":17}]
```

Untuk mengambil kolom a baris hasil 0, aplikasi akan merujuk ke `recordsList.get(0).a`.

Sebaliknya, contoh Java berikut menunjukkan tipe kode yang diperlukan untuk membangun struktur data yang mempertahankan set hasil ketika Anda tidak menggunakan format JSON. Dalam kasus ini, setiap baris dari set hasil berisi bidang dengan informasi tentang satu pengguna. Membangun struktur data untuk merepresentasikan set hasil membutuhkan perulangan melalui baris. Untuk setiap baris, kode mengambil nilai setiap bidang, melakukan konversi tipe yang sesuai, dan menetapkan hasilnya ke bidang yang sesuai dalam objek yang mewakili baris. Kemudian, kode menambahkan objek yang merepresentasikan setiap pengguna ke struktur data yang mewakili seluruh set hasil. Jika kueri diubah untuk menyusun ulang, menambah, atau menghapus bidang dalam set hasil, kode aplikasi juga harus diubah.

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

Nilai sampel berikut menunjukkan nilai bidang `formattedRecords` untuk set hasil dengan jumlah kolom, alias kolom, dan tipe data kolom yang berbeda.

Jika set hasil mencakup beberapa baris, setiap barisnya direpresentasikan sebagai objek yang merupakan elemen array. Setiap kolom dalam set hasil menjadi kunci dalam objek. Nama kolom ini diulang untuk setiap baris dalam set hasil. Oleh karena itu, untuk set hasil yang terdiri dari banyak baris dan kolom, Anda mungkin perlu menentukan alias kolom pendek agar tidak melebihi batas panjang untuk seluruh respons.

Contoh ini bekerja dengan skema dan data sampel berikut:

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

Jika kolom dalam set hasil didefinisikan sebagai ekspresi, teks ekspresi menjadi kunci JSON. Oleh karena itu, praktik mudah umumnya adalah menentukan alias kolom deskriptif untuk setiap ekspresi dalam daftar pilih kueri. Misalnya, kueri berikut mencakup ekspresi seperti panggilan fungsi dan operasi aritmatika dalam daftar pilihannya.

```
select count(*), max(id), 4+7 from sample_names;
```

Ekspresi tersebut diteruskan ke set hasil JSON sebagai kunci.

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

Menambahkan kolom AS dengan label deskriptif membuat kunci lebih sederhana untuk ditafsirkan dalam set hasil JSON.

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

Dengan kueri SQL yang direvisi, label kolom yang ditentukan oleh klausa AS digunakan sebagai nama kunci.

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

Nilai untuk setiap pasangan kunci-nilai dalam string JSON dapat berupa string berkutip. String dapat berisi karakter unicode. Jika string berisi urutan escape maupun karakter '' atau \, karakter tersebut didahului oleh karakter escape backslash. Contoh string JSON berikut menunjukkan kemungkinan ini. Misalnya, hasil `string_with_escape_sequences` berisi karakter khusus backspace, newline, carriage return, tab, form feed, dan \.

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

Nilai untuk setiap pasangan kunci-nilai dalam string JSON juga dapat merepresentasikan angka. Angka tersebut dapat berupa bilangan bulat, nilai floating-point, nilai negatif, atau nilai yang direpresentasikan sebagai notasi eksponensial. Contoh string JSON berikut menunjukkan kemungkinan ini.

```
[{"integer_value":17}]
[{"float_value":10.0}]
```

```
[{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

Nilai Boolean dan null direpresentasikan dengan kata kunci khusus tanpa kutip `true`, `false`, dan `null`. Contoh string JSON berikut menunjukkan kemungkinan ini.

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

Jika Anda memilih nilai tipe BLOB, hasilnya direpresentasikan dalam string JSON sebagai nilai berenkode base64. Untuk mengonversi kembali nilai ke representasi aslinya, Anda dapat menggunakan fungsi dekode yang sesuai dalam bahasa aplikasi Anda. Misalnya, Anda memanggil fungsi `Base64.getDecoder().decode()` di Java. Output sampel berikut menunjukkan hasil pemilihan nilai BLOB `hello world` dan menampilkan set hasil sebagai string JSON.

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

Contoh Python berikut menunjukkan cara mengakses nilai-nilai dari hasil panggilan ke fungsi `execute_statement` Python. Set hasil adalah nilai string di bidang `response['formattedRecords']`. Kode ini mengubah string JSON menjadi struktur data dengan memanggil fungsi `json.loads`. Kemudian, setiap baris dari set hasil adalah elemen daftar dalam struktur data, dan Anda dapat merujuk ke setiap bidang set hasil berdasarkan nama dalam setiap baris.

```
import json

result = json.loads(response['formattedRecords'])
print (result[0]["id"])
```

JavaScript Contoh berikut menunjukkan cara mengakses nilai-nilai dari hasil panggilan ke JavaScript `executeStatement` fungsi. Set hasil adalah nilai string di bidang `response.formattedRecords`. Kode ini mengubah string JSON menjadi struktur data dengan memanggil fungsi `JSON.parse`. Kemudian, setiap baris dari set hasil adalah elemen array dalam struktur data, dan Anda dapat merujuk ke setiap bidang set hasil berdasarkan nama dalam setiap baris.

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

# Memecahkan Masalah Amazon RDS Data API

Gunakan bagian berikut, berjudul dengan pesan kesalahan umum, untuk membantu memecahkan masalah yang Anda miliki dengan Amazon RDS Data API (Data API).

Topik


- Transaksi <transaction_ID>tidak ditemukan
- Paket untuk kueri terlalu besar
- Respons basis data melebihi batas ukuran
- HttpEndpointtidak diaktifkan untuk cluster <cluster_ID>
- DatabaseErrorException: Transaksi masih menjalankan kueri
- Pengecualian hasil yang tidak didukung
- Multi-pernyataan tidak didukung
- Parameter skema tidak didukung


## Transaksi <transaction_ID>tidak ditemukan

Dalam kasus ini, ID transaksi yang ditentukan dalam panggilan API Data tidak ditemukan. Penyebab masalah ini ditambahkan ke pesan kesalahan, dan merupakan salah satu dari yang berikut:

- Transaksi mungkin kedaluwarsa.

  Pastikan setiap panggilan transaksional berjalan dalam tiga menit sejak panggilan sebelumnya berjalan.

  Mungkin juga ID transaksi yang ditentukan tidak dibuat oleh BeginTransactionpanggilan. Pastikan panggilan Anda memiliki ID transaksi yang valid.
- Satu panggilan sebelumnya mengakibatkan penghentian transaksi Anda.

  Transaksi telah diakhiri oleh panggilan `CommitTransaction` atau `RollbackTransaction` Anda.
- Transaksi telah dibatalkan karena kesalahan dari panggilan sebelumnya.

  Periksa apakah panggilan Anda sebelumnya telah memberikan pengecualian apa pun.

Lihat informasi tentang menjalankan transaksi di Memanggil Amazon RDS Data API.

## Paket untuk kueri terlalu besar

Dalam kasus ini, set hasil yang ditampilkan untuk satu baris terlalu besar. Batas ukuran API Data adalah 64 KB per baris dalam set hasil yang ditampilkan oleh basis data.

Untuk mengatasi masalah ini, pastikan setiap baris dalam set hasil berukuran 64 KB atau kurang.

## Respons basis data melebihi batas ukuran

Dalam kasus ini, ukuran set hasil yang ditampilkan oleh basis data terlalu besar. Batas API Data adalah 1 MiB per baris dalam set hasil yang ditampilkan oleh basis data.

Untuk mengatasi masalah ini, pastikan panggilan ke Data API mengembalikan 1 MiB data atau kurang. Jika Anda perlu menampilkan lebih dari 1 MiB, Anda dapat menggunakan beberapa panggilan `ExecuteStatement` dengan klausa `LIMIT` di kueri Anda.

Lihat informasi selengkapnya tentang klausa `LIMIT` di SELECT syntax dalam dokumentasi MySQL.

## HttpEndpointtidak diaktifkan untuk cluster <cluster_ID>

Periksa penyebab potensial berikut untuk masalah ini:

- Cluster Aurora DB tidak mendukung API Data. Untuk informasi tentang jenis cluster DB yang didukung RDS Data API, lihat. the section called “Ketersediaan wilayah dan versi untuk Amazon RDS Data API”
- Data API tidak diaktifkan untuk cluster Aurora DB. Untuk menggunakan Data API dengan cluster Aurora DB, Data API harus diaktifkan untuk cluster DB. Untuk informasi tentang mengaktifkan Data API, lihatMengaktifkan API Data Amazon RDS.
- Cluster DB diganti namanya setelah Data API diaktifkan untuknya. Dalam hal ini, matikan Data API untuk cluster itu dan kemudian aktifkan lagi.
- ARN yang Anda tentukan tidak sama persis dengan ARN klaster. Periksa apakah ARN yang ditampilkan dari sumber lain atau dibangun oleh logika program cocok dengan ARN klaster secara tepat. Misalnya, pastikan ARN yang Anda gunakan memiliki kapitalisasi huruf yang benar untuk semua karakter alfabet.

## DatabaseErrorException: Transaksi masih menjalankan kueri

Jika aplikasi Anda mengirimkan permintaan dengan ID transaksi dan transaksi tersebut saat ini sedang memproses permintaan lain, Data API mengembalikan kesalahan ini ke aplikasi Anda

segera. Kondisi ini mungkin muncul jika aplikasi Anda membuat permintaan asinkron, menggunakan mekanisme seperti “janji” di Javascript.

Untuk mengatasi masalah ini, tunggu hingga permintaan sebelumnya selesai dan kemudian coba lagi permintaan tersebut. Anda dapat terus mencoba lagi sampai kesalahan tidak lagi terjadi, atau aplikasi menerima beberapa jenis kesalahan yang berbeda.

Kondisi ini dapat terjadi dengan Data API untuk Aurora Serverless v2 dan instance yang disediakan. Di Data API forAurora Serverless v1, permintaan selanjutnya untuk ID transaksi yang sama secara otomatis menunggu permintaan sebelumnya selesai. Namun, perilaku lama itu berpotensi mengalami batas waktu karena permintaan sebelumnya terlalu lama. Jika Anda mem-porting aplikasi Data API lama yang membuat permintaan bersamaan, ubah logika penanganan pengecualian Anda untuk memperhitungkan jenis kesalahan baru ini.

## Pengecualian hasil yang tidak didukung

Data API tidak mendukung semua tipe data. Kesalahan ini terjadi ketika Anda menjalankan kueri yang mengembalikan tipe data yang tidak didukung.

Untuk mengatasi masalah ini, kirimkan tipe data yang tidak didukung ke`TEXT`. Misalnya:

```
SELECT custom_type::TEXT FROM my_table;
-- OR
SELECT CAST(custom_type AS TEXT) FROM my_table;
```

## Multi-pernyataan tidak didukung

Multi-pernyataan tidak didukung dalam API Data untuk Aurora Serverless v2 dan klaster yang disediakan. Mencoba mengeksekusi beberapa pernyataan dalam satu panggilan API menghasilkan kesalahan ini.

Untuk mengeksekusi beberapa pernyataan, gunakan panggilan `ExecuteStatement` API terpisah atau gunakan `BatchExecuteStatement` API untuk pemrosesan batch.

## Parameter skema tidak didukung

Aurora Serverless v1 diam-diam mengabaikan parameter skema. Namun, Aurora Serverless v2 dan klaster yang disediakan secara eksplisit menolak panggilan API yang menyertakan parameter skema.

Untuk mengatasi masalah ini, hapus parameter skema dari semua panggilan ke Data API saat Anda menggunakan Aurora Serverless v2 atau klaster yang disediakan.

# Mencatat panggilan Amazon RDS Data API dengan AWS CloudTrail

RDS Data API (Data API) terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Data API. CloudTrail menangkap semua panggilan API untuk Data API sebagai peristiwa, termasuk panggilan dari konsol Amazon RDS dan dari panggilan kode ke operasi API Data. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail peristiwa secara berkelanjutan ke bucket Amazon S3, termasuk peristiwa untuk Data API. Dengan menggunakan data yang dikumpulkan oleh CloudTrail, Anda dapat menentukan banyak informasi. Informasi ini mencakup permintaan yang dibuat untuk API Data, alamat IP asal permintaan, siapa yang membuat permintaan, kapan permintaan tersebut dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

## Bekerja dengan informasi API Data di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas yang didukung (peristiwa manajemen) terjadi di Data API, aktivitas tersebut direkam dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara manajemen terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk peristiwa untuk Data API, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua AWS Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna AWS CloudTrail :

- Ikhtisar untuk membuat jejak
- CloudTrail layanan dan integrasi yang didukung
- Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail
- Menerima file CloudTrail log dari beberapa Wilayah dan Menerima file CloudTrail log dari beberapa akun

Semua operasi API Data dicatat oleh CloudTrail dan didokumentasikan dalam referensi API layanan data Amazon RDS. Misalnya, panggilan ke`BatchExecuteStatement`, `BeginTransaction``CommitTransaction`, dan `ExecuteStatement` operasi menghasilkan entri dalam file CloudTrail log.

Setiap entri peristiwa atau log berisi informasi tentang entitas yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut:

- Baik permintaan tersebut dibuat dengan kredensial pengguna atau root.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi selengkapnya, lihat Elemen userIdentity CloudTrail .

## Menyertakan dan mengecualikan peristiwa API Data dari jejak AWS CloudTrail

Sebagian besar pengguna API Data mengandalkan peristiwa dalam AWS CloudTrail jejak untuk menyediakan catatan operasi API Data. Data peristiwa tidak mengungkapkan nama database, nama skema, atau pernyataan SQL dalam permintaan ke Data API. Namun, mengetahui pengguna mana yang membuat jenis panggilan terhadap cluster DB tertentu pada waktu tertentu dapat membantu mendeteksi pola akses anomali.

### Menyertakan peristiwa API Data dalam suatu AWS CloudTrail jejak

Untuk Aurora PostgreSQL Serverless v2 dan database yang disediakan, operasi API Data berikut dicatat sebagai peristiwa data. AWS CloudTrail Peristiwa data adalah operasi API data-plane volume tinggi yang CloudTrail tidak masuk secara default. Biaya tambahan berlaku untuk peristiwa data. Untuk informasi tentang CloudTrail harga, lihat AWS CloudTrail Harga.

- BatchExecuteStatement
- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

Anda dapat menggunakan CloudTrail konsol AWS CLI, atau operasi CloudTrail API untuk mencatat operasi API Data ini. Di CloudTrail konsol, pilih RDS Data API - DB Cluster untuk tipe peristiwa Data. Untuk informasi selengkapnya, lihat Mencatat peristiwa data dengan AWS Management Console di Panduan AWS CloudTrail Pengguna.

Dengan menggunakan AWS CLI, jalankan `aws cloudtrail put-event-selectors` perintah untuk mencatat operasi Data API ini untuk jejak Anda. Untuk mencatat semua peristiwa Data API pada kluster DB, `AWS::RDS::DBCluster` tentukan jenis sumber daya. Contoh berikut mencatat semua peristiwa Data API pada cluster DB. Untuk informasi selengkapnya, lihat Mencatat peristiwa data dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
         "Field": "resources.type",
         "Equals": [
            "AWS::RDS::DBCluster"
         ]
      }
   ]
}'
```

Anda dapat mengonfigurasi pemilih acara lanjutan untuk memfilter tambahan pada`readOnly`, `eventName`, dan `resources.`ARN bidang. Untuk informasi lebih lanjut tentang bidang ini, lihat AdvancedFieldSelector.

## Mengecualikan peristiwa Data API dari AWS CloudTrail jejak (Aurora Serverless v1hanya)

UntukAurora Serverless v1, peristiwa Data API adalah peristiwa manajemen. Secara default, semua peristiwa Data API disertakan dalam AWS CloudTrail jejak. Namun, karena Data API dapat menghasilkan sejumlah besar peristiwa, Anda mungkin ingin mengecualikan peristiwa ini dari

CloudTrail jejak Anda. Setelan peristiwa Exclude Amazon RDS Data API mengecualikan semua peristiwa Data API dari jejak. Anda tidak dapat mengecualikan peristiwa API Data tertentu.

Untuk mengecualikan peristiwa API Data dari jejak, lakukan hal berikut:

- Di CloudTrail konsol, pilih setelan Kecualikan peristiwa Amazon RDS Data API saat Anda membuat jejak atau memperbarui jejak.
- Di CloudTrail API, gunakan PutEventSelectorsoperasi. Jika Anda menggunakan pemilih peristiwa lanjutan, Anda dapat mengecualikan peristiwa API Data dengan menyetel `eventSource` bidang yang tidak sama dengan`rdsdata.amazonaws.com`. Jika Anda menggunakan pemilih acara dasar, Anda dapat mengecualikan peristiwa Data API dengan menyetel nilai `ExcludeManagementEventSources` atribut `kerdsdata.amazonaws.com`. Untuk informasi selengkapnya, lihat Mencatat peristiwa dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

> **Warning**
>
> Mengecualikan peristiwa Data API dari CloudTrail log dapat mengaburkan tindakan API Data. Berhati-hatilah saat memberikan `cloudtrail:PutEventSelectors` kepada pengguna utama yang diperlukan untuk melakukan operasi ini.

Anda dapat mematikan pengecualian ini kapan saja dengan mengubah pengaturan konsol atau pemilih peristiwa untuk jejak. Jejak kemudian akan mulai mencatat peristiwa API Data. Namun, jejak tidak dapat memulihkan peristiwa API Data yang terjadi sedangkan pengecualian tersebut sedang berlaku.

Saat Anda mengecualikan peristiwa Data API dengan menggunakan konsol atau API, operasi CloudTrail `PutEventSelectors` API yang dihasilkan juga dicatat di CloudTrail log Anda. Jika peristiwa Data API tidak muncul di CloudTrail log Anda, cari `PutEventSelectors` peristiwa dengan `ExcludeManagementEventSources` atribut yang disetel `kerdsdata.amazonaws.com`.

Untuk informasi lebih lanjut, lihat Mencatat log peristiwa manajemen untuk jejak di Panduan Pengguna AWS CloudTrail .

# Memahami entri file log API Data

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, jadi file tersebut tidak muncul dalam urutan tertentu.

Aurora PostgreSQL Serverless v2 dan disediakan

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `ExecuteStatement` operasi untuk Aurora PostgreSQL Serverless v2 dan database yang disediakan. Untuk database ini, semua peristiwa Data API adalah peristiwa data di mana sumber peristiwa adalah rdsdataapi.amazonaws.com dan jenis acara adalah Rds Data Service.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
```

```
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```

Aurora Serverless v1

Contoh berikut menunjukkan bagaimana entri CloudTrail log contoh sebelumnya muncul untuk. Aurora Serverless v1 UntukAurora Serverless v1, semua acara adalah acara manajemen di mana sumber acara adalah rdsdata.amazonaws.com dan jenis acaranya adalah. AwsApiCall

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
```

```
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

# Memantau kueri API Data RDS dengan Performance Insights

Jika klaster Aurora menjalankan Aurora Serverless v2 atau menyediakan instance, Anda dapat menggunakan Performance Insights dengan RDS Data API.

Untuk informasi selengkapnya tentang cara menggunakan Performance Insights dengan Aurora, lihat. Memantau muatan DB dengan Wawasan Performa di Amazon Aurora

## Bagaimana kueri API Data RDS direpresentasikan dalam Performance Insights

Dengan Data API, klaster Aurora Anda memproses kueri berdasarkan panggilan API Data yang Anda kirimkan dari aplikasi Anda. Data API juga melakukan beberapa pernyataan SQL sebagai bagian dari cara kerja internalnya sendiri, seperti membatalkan kueri yang melebihi ambang batas waktu. Kedua jenis operasi SQL ditampilkan dalam statistik dan grafik Performance Insights.

- Untuk kueri Data API yang Anda kirimkan ke cluster Aurora, bidang Host di dasbor PI ditandai sebagai RDS Data API. Untuk Aurora PostgreSQL, bidang application_name memiliki nilai. `rds-data-api` Cari label ini saat Anda menganalisis pemuatan basis data menggunakan host Top atau Aplikasi Teratas sebagai dimensi.
- Semua kueri internal yang dijalankan Data API untuk mengelola aspek database seperti kumpulan koneksi dan batas waktu kueri dianotasi dengan awalan RDS Data API. Contoh: `/* RDS Data API */ select * from my_table;` Mencari awalan ini ketika Anda menganalisis beban database oleh Top SQL sebagai dimensi. pernyataan dianotasi dengan komentar SQL dari. `/* RDS Data API */`

# Menggunakan editor kueri Aurora

Editor kueri Aurora memungkinkan Anda menjalankan pernyataan SQL pada cluster Aurora DB Anda melalui file. AWS Management Console Anda dapat menjalankan query SQL, pernyataan manipulasi data (DHTML), dan pernyataan definisi data (DDL). Menggunakan antarmuka konsol memungkinkan Anda melakukan pemeliharaan database, menghasilkan laporan, dan melakukan eksperimen SQL. Anda dapat menghindari pengaturan konfigurasi jaringan untuk terhubung ke cluster DB Anda dari sistem klien terpisah seperti EC2 instance atau komputer laptop.

Editor kueri memerlukan cluster Aurora DB dengan RDS Data API (Data API) diaktifkan. Untuk informasi tentang cluster DB yang mendukung Data API dan cara mengaktifkannya, lihatMenggunakan Amazon RDS Data API. SQL yang dapat Anda jalankan tunduk pada batasan Data API. Untuk informasi selengkapnya, lihat the section called “Batasan”.

## Ketersediaan editor kueri

Editor kueri tersedia untuk cluster Aurora DB menggunakan versi mesin Aurora MySQL dan Aurora PostgreSQL yang mendukung API Data, dan di mana Data API tersedia. Wilayah AWS Untuk informasi selengkapnya, lihat Daerah yang Didukung dan engine Aurora DB untuk RDS Data API.

## Mengotorisasi akses ke editor kueri

Pengguna harus diotorisasi untuk menjalankan kueri di editor kueri. Anda dapat mengotorisasi pengguna untuk menjalankan kueri di editor kueri dengan menambahkan `AmazonRDSDataFullAccess` kebijakan, kebijakan yang telah ditentukan AWS Identity and Access Management (IAM), ke pengguna tersebut.

> **Note**
>
> Pastikan untuk menggunakan nama pengguna dan kata sandi yang sama saat Anda membuat pengguna IAM seperti yang Anda lakukan untuk pengguna database, seperti nama pengguna administratif dan kata sandi. Untuk informasi selengkapnya, lihat Membuat Pengguna IAM di Akun AWS Anda dalam Panduan Pengguna AWS Identity and Access Management .

Anda juga dapat membuat kebijakan IAM yang memberikan akses ke editor kueri. Setelah Anda membuat kebijakan, tambahkan ke setiap pengguna yang membutuhkan akses ke editor kueri.

Kebijakan berikut memberikan izin minimum yang diperlukan bagi pengguna untuk mengakses editor kueri.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
```

```
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
        }
    ]
}
```

Untuk informasi tentang pembuatan kebijakan IAM, lihat Membuat kebijakan IAM di Panduan Pengguna AWS Identity and Access Management .

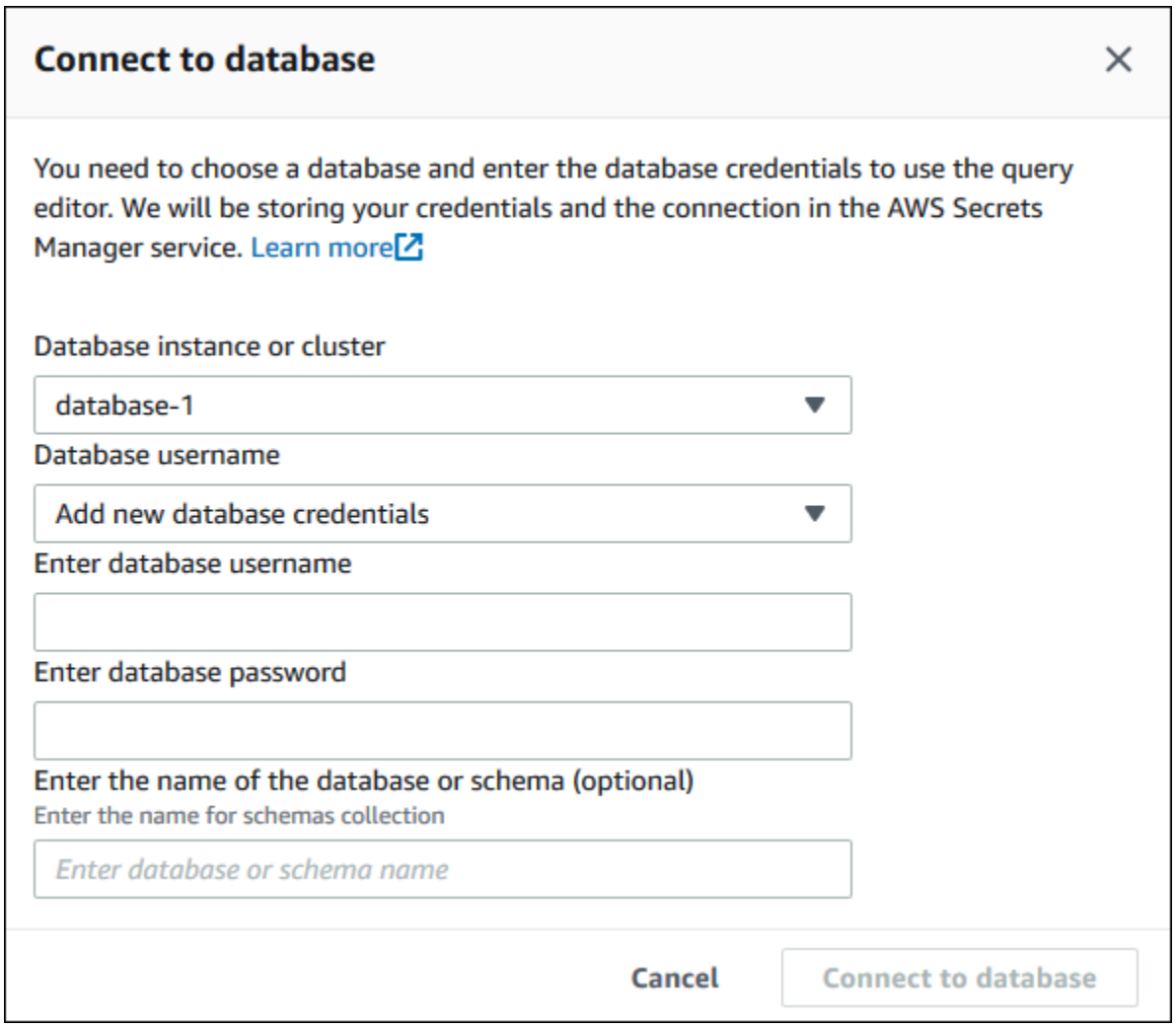
Untuk informasi tentang cara menambahkan kebijakan IAM ke pengguna, lihat Menambahkan dan menghapus izin identitas IAM di Panduan Pengguna AWS Identity and Access Management .

## Menjalankan kueri di editor kueri

Anda dapat menjalankan pernyataan SQL pada cluster Aurora DB di editor kueri. SQL yang dapat Anda jalankan tunduk pada batasan Data API. Untuk informasi selengkapnya, lihat the section called “Batasan”.

Untuk menjalankan kueri di editor kueri

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda membuat cluster Aurora DB yang ingin Anda kueri.
3. Di panel navigasi, pilih Basis data.
4. Pilih cluster Aurora DB tempat Anda ingin menjalankan kueri SQL.
5. Untuk Tindakan, pilih Kueri. Jika Anda belum tersambung ke basis data sebelumnya, halaman Hubungkan ke basis data akan terbuka.




6. Masukkan informasi berikut:

   a. Untuk instance atau cluster Database, pilih cluster Aurora DB tempat Anda ingin menjalankan kueri SQL.

   b. Untuk Nama pengguna basis data, pilih nama pengguna dari pengguna basis data yang akan dihubungkan, atau pilih Tambahkan kredensial basis data baru. Jika Anda memilih Tambahkan kredensial basis data baru, masukkan nama pengguna untuk kredensial basis data baru di Masukkan nama pengguna basis data.

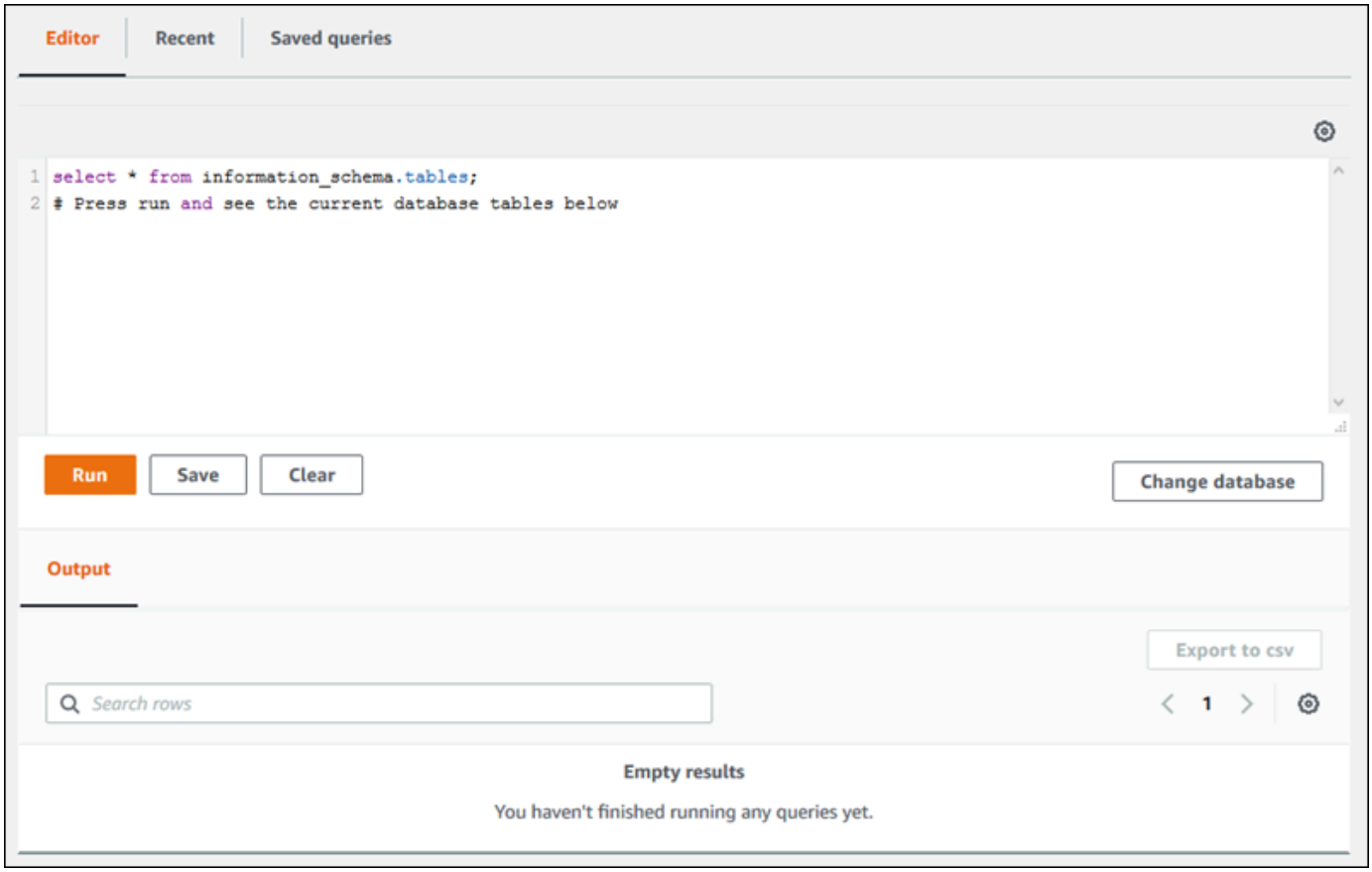
   c. Untuk Masukkan kata sandi basis data, masukkan kata sandi untuk pengguna basis data yang Anda pilih.

   d. Di kotak terakhir, masukkan nama basis data atau skema yang ingin digunakan untuk klaster DB Aurora.

   e. Pilih Hubungkan ke basis data.

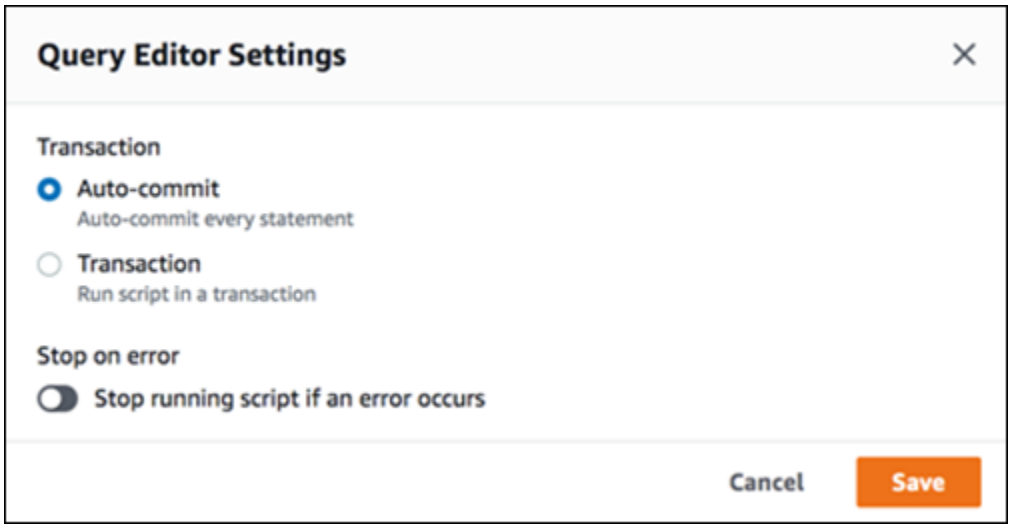
> **Note**
>
> Jika koneksi berhasil, informasi koneksi dan autentikasi Anda akan disimpan di AWS Secrets Manager. Anda tidak perlu memasukkan informasi koneksi lagi.

7. Di editor kueri, masukkan kueri SQL yang ingin dijalankan di basis data.




Setiap pernyataan SQL dapat dilakukan secara otomatis, atau Anda dapat menjalankan pernyataan SQL dalam skrip sebagai bagian dari transaksi. Untuk mengontrol perilaku ini, pilih ikon roda gigi di atas jendela kueri.

Jendela Pengaturan Editor Kueri muncul.




Jika Anda memilih Lakukan otomatis, setiap pernyataan SQL akan dilakukan secara otomatis. Jika Anda memilih Transaksi, Anda dapat menjalankan grup pernyataan dalam skrip. Pernyataan secara otomatis dilakukan di akhir skrip kecuali secara eksplisit dilakukan atau dibatalkan sebelum itu. Selain itu, Anda dapat memilih untuk menghentikan skrip yang sedang berjalan jika terjadi kesalahan dengan mengaktifkan Hentikan saat terjadi kesalahan.

> **Note**
>
> Dalam grup pernyataan, pernyataan bahasa definisi data (DDL) dapat menyebabkan pernyataan bahasa manipulasi data (DML) sebelumnya dilakukan. Anda juga dapat memasukkan pernyataan `COMMIT` dan `ROLLBACK` dalam grup pernyataan dalam skrip.

Setelah Anda menentukan pilihan di jendela Pengaturan Editor Kueri, pilih Simpan.

8. Pilih Jalankan atau tekan Ctrl+Enter, dan editor kueri akan menampilkan hasil kueri Anda.

   Setelah menjalankan kueri, simpan ke Kueri tersimpan dengan memilih Simpan.

   Ekspor hasil kueri ke format spreadsheet dengan memilih Ekspor ke csv.

Anda dapat menemukan, mengedit, dan menjalankan kembali kueri sebelumnya. Untuk melakukannya, pilih tab Baru atau tab Kueri tersimpan, pilih teks kueri, lalu pilih Jalankan.

Untuk mengubah basis data, pilih Ubah basis data.

# Referensi Layanan Metadata Kueri Database () DBQMS API

Layanan Metadata Kueri Basis Data (`dbqms`) adalah layanan khusus internal. Ini menyediakan kueri terbaru dan tersimpan Anda untuk editor kueri di AWS Management Console untuk beberapa AWS layanan, termasuk AmazonRDS.

DBQMSTindakan yang didukung

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries
- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries
- DescribeQueryHistory
- DescribeTabs

- GetQueryString
- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

## CreateFavoriteQuery

Menyimpan kueri favorit baru. Setiap pengguna dapat membuat hingga 1.000 kueri yang disimpan. Batas ini dapat berubah di masa mendatang.

## CreateQueryHistory

Simpan entri riwayat kueri baru.

## CreateTab

Menyimpan tab kueri baru. Setiap pengguna dapat membuat hingga 10 tab kueri.

## DeleteFavoriteQueries

Hapus satu atau beberapa kueri tersimpan.

## DeleteQueryHistory

Hapus entri riwayat kueri.

## DeleteTab

Hapus entri tab kueri.

## DescribeFavoriteQueries

Mencantumkan kueri disimpan yang dibuat oleh pengguna dalam akun tertentu.

## DescribeQueryHistory

Mencantumkan entri riwayat kueri.

## DescribeTabs

Mencantumkan tab kueri yang dibuat oleh pengguna dalam akun tertentu.

## GetQueryString

Mengambil teks kueri penuh dari ID kueri.

## UpdateFavoriteQuery

Memperbarui string, deskripsi, nama, atau tanggal kedaluwarsa kueri.

## UpdateQueryHistory

Memperbarui status riwayat kueri.

## UpdateTab

Memperbarui nama tab kueri dan string kueri.

# Menggunakan machine learning Amazon Aurora

Dengan menggunakan pembelajaran mesin Amazon Aurora, Anda dapat mengintegrasikan cluster Aurora DB Anda dengan salah satu layanan pembelajaran AWS mesin berikut, tergantung pada kebutuhan Anda. Mereka masing-masing mendukung kasus penggunaan pembelajaran mesin tertentu.

Amazon Bedrock

Amazon Bedrock adalah layanan yang dikelola sepenuhnya yang membuat model fondasi terkemuka dari perusahaan AI tersedia melaluiAPI, bersama dengan perkakas pengembang untuk membantu membangun dan menskalakan aplikasi AI generatif. Dengan Amazon Bedrock, Anda membayar untuk menjalankan inferensi pada salah satu model yayasan pihak ketiga. Harga didasarkan pada volume token input dan token output, dan apakah Anda telah membeli throughput yang disediakan untuk model tersebut. Untuk informasi lebih lanjut, lihat Apa itu Amazon Bedrock? di Panduan Pengguna Amazon Bedrock.

Amazon Comprehend

Amazon Comprehend adalah layanan NLP pemrosesan bahasa alami terkelola () yang digunakan untuk mengekstrak wawasan dari dokumen. Dengan Amazon Comprehend, Anda dapat menyimpulkan sentimen berdasarkan konten dokumen, dengan menganalisis entitas, frasa kunci, bahasa, dan fitur lainnya. Untuk mempelajari selengkapnya, lihat Apa itu Amazon Comprehend? di Panduan Developer Amazon Comprehend.

SageMaker AI

Amazon SageMaker AI adalah layanan pembelajaran mesin yang dikelola sepenuhnya. Ilmuwan dan pengembang data menggunakan Amazon SageMaker AI untuk membangun, melatih, dan menguji model pembelajaran mesin untuk berbagai tugas inferensi, seperti deteksi penipuan dan rekomendasi produk. Ketika model pembelajaran mesin siap digunakan dalam produksi, model ini dapat diterapkan ke lingkungan yang dihosting Amazon SageMaker AI. Untuk informasi lebih lanjut, lihat Apa itu Amazon SageMaker AI? di Panduan Pengembang SageMaker AI Amazon.

Menggunakan Amazon Comprehend dengan cluster Aurora DB Anda memiliki pengaturan awal yang lebih sedikit daripada menggunakan AI. SageMaker Jika Anda baru mengenal pembelajaran AWS mesin, kami sarankan Anda memulai dengan menjelajahi Amazon Comprehend.

Topik

- Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL
- Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora Postgre SQL


# Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora My SQL

Dengan menggunakan pembelajaran mesin Amazon Aurora dengan cluster Aurora My SQL DB Anda, Anda dapat menggunakan Amazon Bedrock, Amazon Comprehend, atau Amazon AI, tergantung pada kebutuhan Anda. SageMaker Mereka masing-masing mendukung kasus penggunaan pembelajaran mesin yang berbeda.

Daftar Isi


- Persyaratan untuk menggunakan pembelajaran mesin Aurora dengan Aurora My SQL
- Wilayah dan ketersediaan versi
- Fitur dan keterbatasan pembelajaran mesin Aurora yang didukung dengan Aurora My SQL
- Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan pembelajaran mesin Aurora
  - Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon Bedrock
  - Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon Comprehend
  - Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan AI SageMaker
    - Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon S3 SageMaker untuk AI (Opsional)
  - Memberikan akses pengguna basis data ke machine learning Aurora
    - Memberikan akses ke fungsi Amazon Bedrock
    - Memberikan akses ke fungsi Amazon Comprehend
    - Memberikan akses ke fungsi SageMaker AI
- Menggunakan Amazon Bedrock dengan cluster Aurora My DB SQL Anda
- Menggunakan Amazon Comprehend dengan cluster Aurora My DB Anda SQL
- Menggunakan SageMaker AI dengan cluster Aurora My SQL DB Anda
  - Persyaratan set karakter untuk fungsi SageMaker AI yang mengembalikan string
  - Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan)
- Pertimbangan kinerja untuk menggunakan pembelajaran mesin Aurora dengan Aurora My SQL
  - Model dan prompt

  - Cache kueri
  - Optimalisasi batch untuk panggilan fungsi machine learning Aurora
- Memantau machine learning Aurora

## Persyaratan untuk menggunakan pembelajaran mesin Aurora dengan Aurora My SQL

AWS Layanan pembelajaran mesin adalah layanan terkelola yang diatur dan dijalankan di lingkungan produksi mereka sendiri. Pembelajaran mesin Aurora mendukung integrasi dengan Amazon Bedrock, Amazon Comprehend, dan AI. SageMaker Sebelum mencoba mengatur cluster Aurora My SQL DB Anda untuk menggunakan pembelajaran mesin Aurora, pastikan Anda memahami persyaratan dan prasyarat berikut.

- Layanan pembelajaran mesin harus berjalan Wilayah AWS sama dengan cluster Aurora My SQL DB Anda. Anda tidak dapat menggunakan layanan pembelajaran mesin dari cluster Aurora My SQL DB di Wilayah yang berbeda.
- Jika klaster Aurora My SQL DB Anda berada di cloud publik virtual (VPC) yang berbeda dari Amazon Bedrock, Amazon Comprehend, atau layanan SageMaker AI Anda, grup Keamanan perlu mengizinkan koneksi keluar ke layanan pembelajaran mesin VPC Aurora target. Untuk informasi selengkapnya, lihat Mengontrol lalu lintas ke AWS sumber daya Anda menggunakan grup keamanan di Panduan VPC Pengguna Amazon.
- Anda dapat memutakhirkan cluster Aurora yang menjalankan versi Aurora My yang lebih rendah SQL ke versi yang lebih tinggi yang didukung jika Anda ingin menggunakan pembelajaran mesin Aurora dengan cluster itu. Untuk informasi selengkapnya, lihat Pembaruan mesin database untuk Amazon Aurora My SQL.
- Cluster Aurora My SQL DB Anda harus menggunakan grup parameter cluster DB kustom. Di akhir proses penyiapan untuk setiap layanan machine learning Aurora yang ingin Anda gunakan, Anda menambahkan Amazon Resource Name (ARN) dari IAM peran terkait yang dibuat untuk layanan tersebut. Kami menyarankan Anda membuat grup parameter cluster DB khusus untuk Aurora My Anda SQL terlebih dahulu dan mengkonfigurasi cluster Aurora My SQL DB Anda untuk menggunakannya sehingga siap untuk Anda modifikasi di akhir proses penyiapan.
- Untuk SageMaker AI:
  - Komponen pembelajaran mesin yang ingin Anda gunakan untuk kesimpulan harus diatur dan siap digunakan. Selama proses konfigurasi untuk cluster Aurora My SQL DB Anda, pastikan untuk memiliki titik ARN akhir SageMaker AI yang tersedia. Ilmuwan data di tim

Anda kemungkinan besar paling mampu menangani bekerja dengan SageMaker AI untuk mempersiapkan model dan menangani tugas-tugas semacam itu lainnya. Untuk memulai dengan Amazon SageMaker AI, lihat Memulai dengan Amazon SageMaker AI. Untuk informasi selengkapnya tentang inferensi dan titik akhir, lihat Inferensi real-time.

- Untuk menggunakan SageMaker AI dengan data pelatihan Anda sendiri, Anda harus menyiapkan bucket Amazon S3 sebagai bagian dari konfigurasi Aurora My SQL untuk pembelajaran mesin Aurora. Untuk melakukannya, Anda mengikuti proses umum yang sama seperti untuk menyiapkan integrasi SageMaker AI. Untuk ringkasan proses penyiapan opsional ini, lihat Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon S3 SageMaker untuk AI (Opsional).

- Untuk database global Aurora, Anda menyiapkan layanan pembelajaran mesin Aurora yang ingin Anda gunakan dalam semua yang Wilayah AWS membentuk basis data global Aurora Anda. Misalnya, jika Anda ingin menggunakan pembelajaran mesin Aurora dengan SageMaker AI untuk database global Aurora Anda, Anda melakukan hal berikut untuk setiap cluster Aurora My DB di setiap: SQL Wilayah AWS
  - Siapkan layanan Amazon SageMaker AI dengan model pelatihan SageMaker AI dan titik akhir yang sama. Ini juga harus menggunakan nama yang sama.
  - Buat IAM peran seperti yang dijelaskan dalamMenyiapkan cluster Aurora My SQL DB Anda untuk menggunakan pembelajaran mesin Aurora.
  - Tambahkan IAM peran ke grup parameter cluster DB khusus untuk setiap cluster Aurora My SQL DB di setiap cluster. ARN Wilayah AWS

  Tugas-tugas ini mengharuskan pembelajaran mesin Aurora tersedia untuk versi Aurora My Anda SQL dalam semua Wilayah AWS yang membentuk basis data global Aurora Anda.

## Wilayah dan ketersediaan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi khusus dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS.

- Untuk informasi tentang versi dan ketersediaan Wilayah untuk Amazon Comprehend dan SageMaker Amazon AI dengan SQL Aurora My, lihat. Pembelajaran mesin Aurora dengan Aurora My SQL
- Amazon Bedrock hanya didukung di Aurora SQL My versi 3.06 dan lebih tinggi.

Untuk informasi tentang ketersediaan Wilayah untuk Amazon Bedrock, lihat Dukungan model oleh Wilayah AWS di Panduan Pengguna Amazon Bedrock.

## Fitur dan keterbatasan pembelajaran mesin Aurora yang didukung dengan Aurora My SQL

Saat menggunakan SQL Aurora My dengan pembelajaran mesin Aurora, batasan berikut berlaku:

- Ekstensi pembelajaran mesin Aurora tidak mendukung antarmuka vektor.
- Integrasi pembelajaran mesin Aurora tidak didukung saat digunakan dalam pemicu.
- Fungsi pembelajaran mesin Aurora tidak kompatibel dengan replikasi biner logging (binlog).
  - Pengaturan `--binlog-format=STATEMENT` memunculkan pengecualian untuk panggilan ke fungsi machine learning Aurora.
  - Fungsi machine learning Aurora Function bersifat nondeterministik, dan fungsi tersimpan nondeterministik tidak kompatibel dengan format binlog.

  Untuk informasi selengkapnya, lihat Format Pencatatan Biner di SQL dokumentasi Saya.
- Fungsi tersimpan yang memanggil tabel dengan kolom generated-always tidak didukung. Ini berlaku untuk semua fungsi Aurora My storageSQL. Untuk mempelajari lebih lanjut tentang jenis kolom ini, lihat CREATETABLEdan Kolom yang Dihasilkan dalam SQL dokumentasi Saya.
- Fungsi Amazon Bedrock tidak mendukung`RETURNS JSON`. Anda dapat menggunakan `CONVERT` atau `CAST` mengonversi dari `TEXT` ke `JSON` jika diperlukan.
- Amazon Bedrock tidak mendukung permintaan batch.
- Aurora My SQL mendukung titik akhir SageMaker AI apa pun yang membaca dan menulis format value (CSV) yang dipisahkan koma, melalui file. `ContentType text/csv` Format ini diterima oleh algoritme SageMaker AI bawaan berikut:
  - Linear Learner
  - Random Cut Forest
  - XGBoost

  Untuk mempelajari lebih lanjut tentang algoritme ini, lihat Memilih Algoritma di Panduan Pengembang Amazon SageMaker AI.

# Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan pembelajaran mesin Aurora

Dalam topik berikut, Anda dapat menemukan prosedur pengaturan terpisah untuk masing-masing layanan machine learning Aurora ini.

Topik


- Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon Bedrock
- Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon Comprehend
- Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan AI SageMaker
  - Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon S3 SageMaker untuk AI (Opsional)
- Memberikan akses pengguna basis data ke machine learning Aurora
  - Memberikan akses ke fungsi Amazon Bedrock
  - Memberikan akses ke fungsi Amazon Comprehend
  - Memberikan akses ke fungsi SageMaker AI


## Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon Bedrock

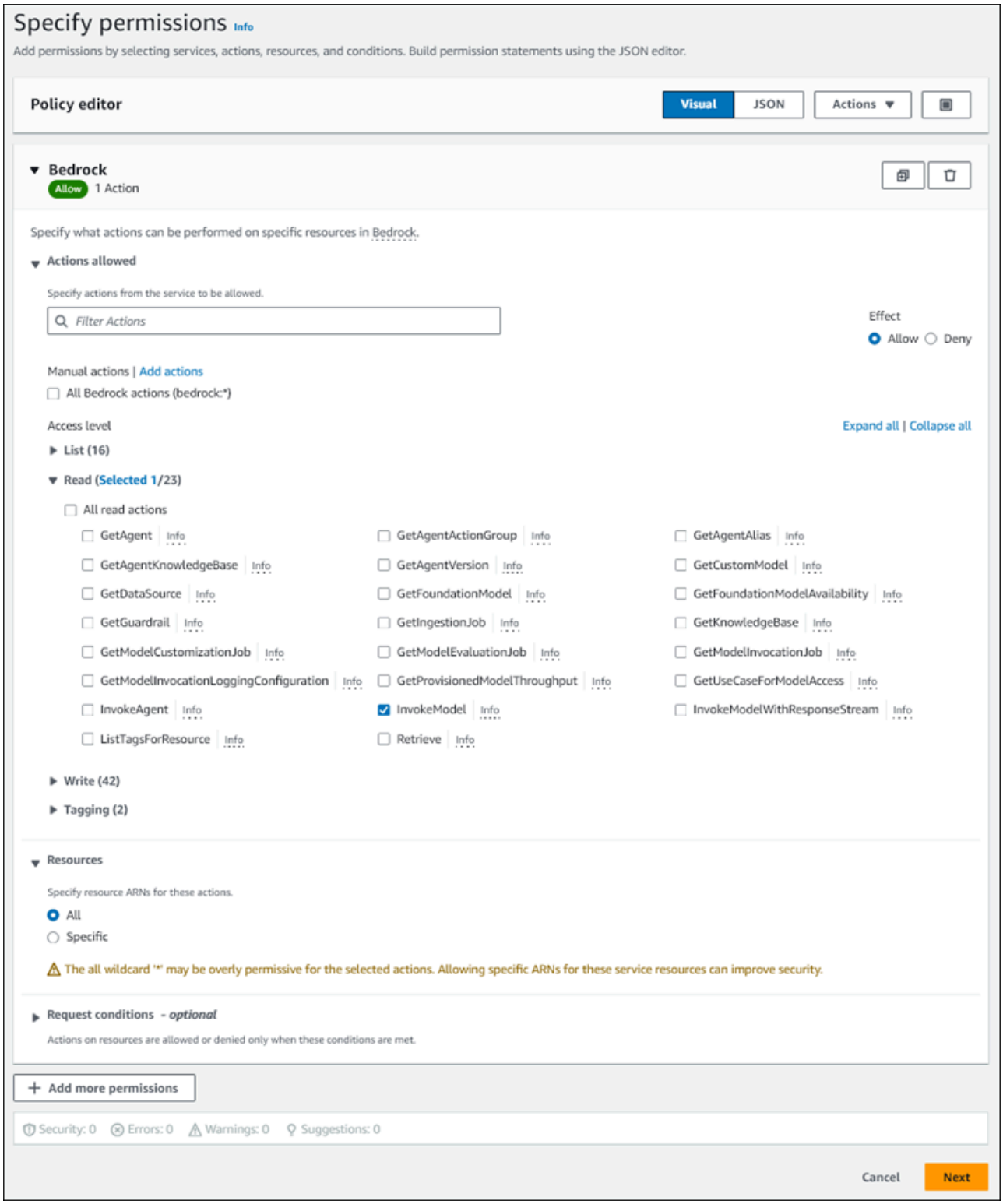
Pembelajaran mesin Aurora bergantung pada AWS Identity and Access Management (IAM) peran dan kebijakan untuk memungkinkan klaster Aurora My SQL DB Anda mengakses dan menggunakan layanan Amazon Bedrock. Prosedur berikut membuat kebijakan dan peran IAM izin sehingga klaster DB Anda dapat berintegrasi dengan Amazon Bedrock.

Untuk membuat IAM kebijakan

1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Pilih Kebijakan di panel navigasi.
3. Pilih Buat kebijakan.
4. Pada halaman Tentukan izin, untuk Pilih layanan, pilih Batuan Dasar.

   Tampilan izin Amazon Bedrock.
5. Perluas Baca, lalu pilih InvokeModel.
6. Untuk Sumber Daya, pilih Semua.

Halaman Tentukan izin harus menyerupai gambar berikut.




7. Pilih Berikutnya.

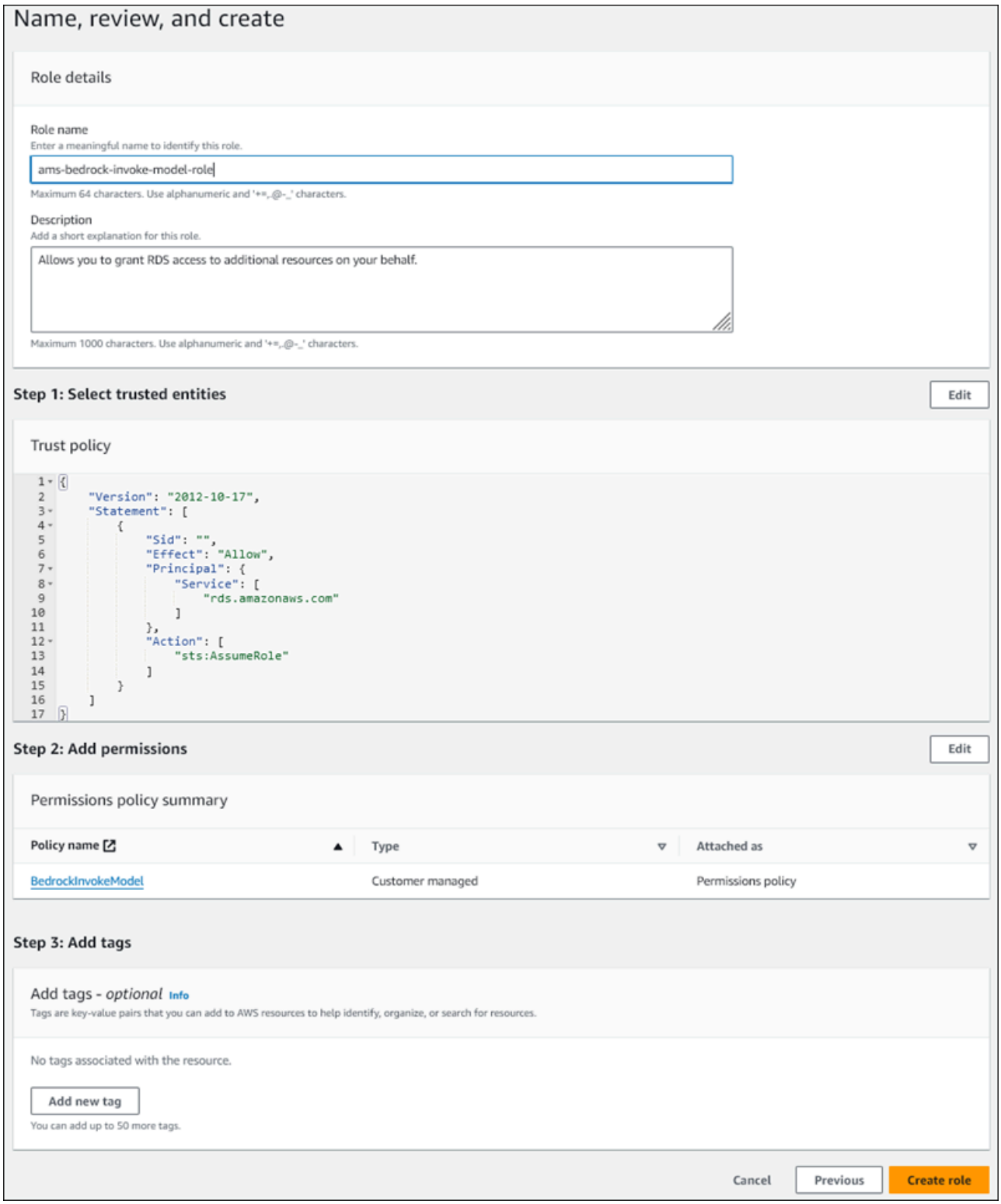
8. Pada halaman Tinjau dan buat, masukkan nama untuk kebijakan Anda, misalnya**`BedrockInvokeModel`**.
9. Tinjau kebijakan Anda, lalu pilih Buat kebijakan.

Selanjutnya Anda membuat IAM peran yang menggunakan kebijakan izin Amazon Bedrock.

Untuk membuat IAM peran

1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Pilih Peran di panel navigasi.
3. Pilih Buat peran.
4. Pada halaman Pilih entitas tepercaya, untuk kasus penggunaan, pilih RDS.
5. Pilih RDS- Tambahkan Peran ke Database, lalu pilih Berikutnya.
6. Pada halaman Tambahkan izin, untuk kebijakan Izin, pilih IAM kebijakan yang Anda buat, lalu pilih Berikutnya.
7. Pada halaman Nama, tinjau, dan buat, masukkan nama untuk peran Anda, misalnya**`ams-bedrock-invoke-model-role`**.

   Peran harus menyerupai gambar berikut.



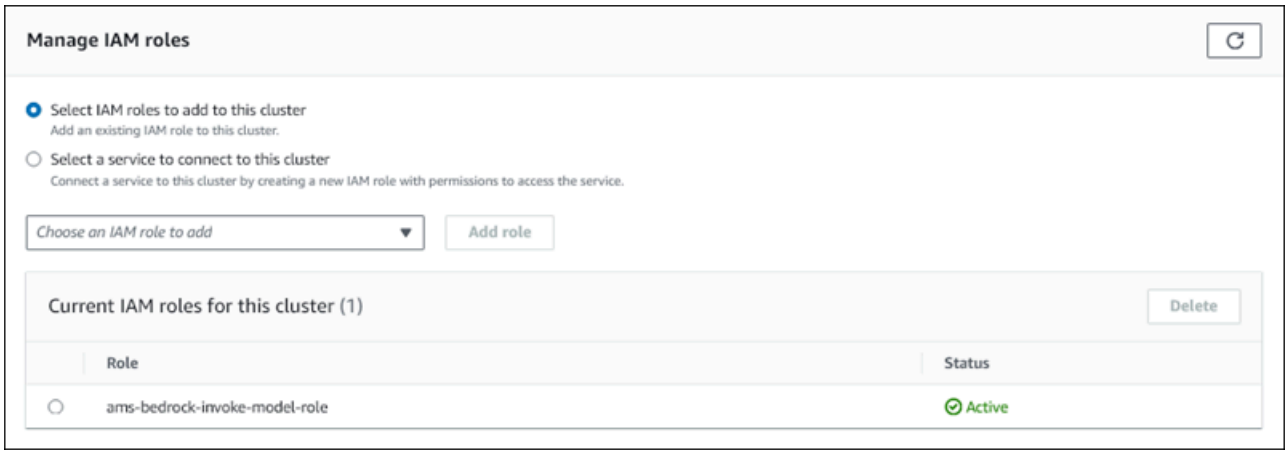

8. Tinjau peran Anda, lalu pilih Buat peran.

Selanjutnya Anda mengaitkan IAM peran Amazon Bedrock dengan cluster DB Anda.

Untuk mengaitkan IAM peran dengan cluster DB Anda

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Basis data dari panel navigasi.
3. Pilih klaster Aurora My SQL DB yang ingin Anda sambungkan ke layanan Amazon Bedrock.
4. Pilih tab Konektivitas & keamanan.
5. Untuk bagian Kelola IAM peran, pilih Pilih IAM untuk ditambahkan ke klaster ini.
6. Pilih IAM yang Anda buat, lalu pilih Tambah peran.

   IAMPeran dikaitkan dengan cluster DB Anda, pertama dengan status Tertunda, lalu Aktif. Saat proses selesai, Anda dapat menemukan peran dalam daftar Peran saat ini IAM untuk klaster ini.




Anda harus menambahkan IAM peran ini ke parameter grup `aws_default_bedrock_role` parameter cluster DB kustom yang terkait dengan cluster Aurora My SQL DB Anda. ARN Jika cluster Aurora My SQL DB Anda tidak menggunakan grup parameter cluster DB kustom, Anda perlu membuatnya untuk digunakan dengan cluster Aurora My SQL DB Anda untuk menyelesaikan integrasi. Untuk informasi selengkapnya, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

Untuk mengkonfigurasi parameter cluster DB

1. Di RDS Konsol Amazon, buka tab Konfigurasi klaster Aurora My SQL DB Anda.
2. Temukan grup parameter cluster DB yang dikonfigurasi untuk cluster Anda. Pilih tautan untuk membuka grup parameter cluster DB kustom Anda, lalu pilih Edit.

3. Temukan parameter `aws_default_bedrock_role` dalam grup parameter klaster DB kustom Anda.
4. Di bidang Nilai, masukkan ARN IAM peran.
5. Pilih Simpan perubahan untuk menyimpan pengaturan.
6. Reboot instance utama cluster Aurora My SQL DB Anda sehingga pengaturan parameter ini berlaku.

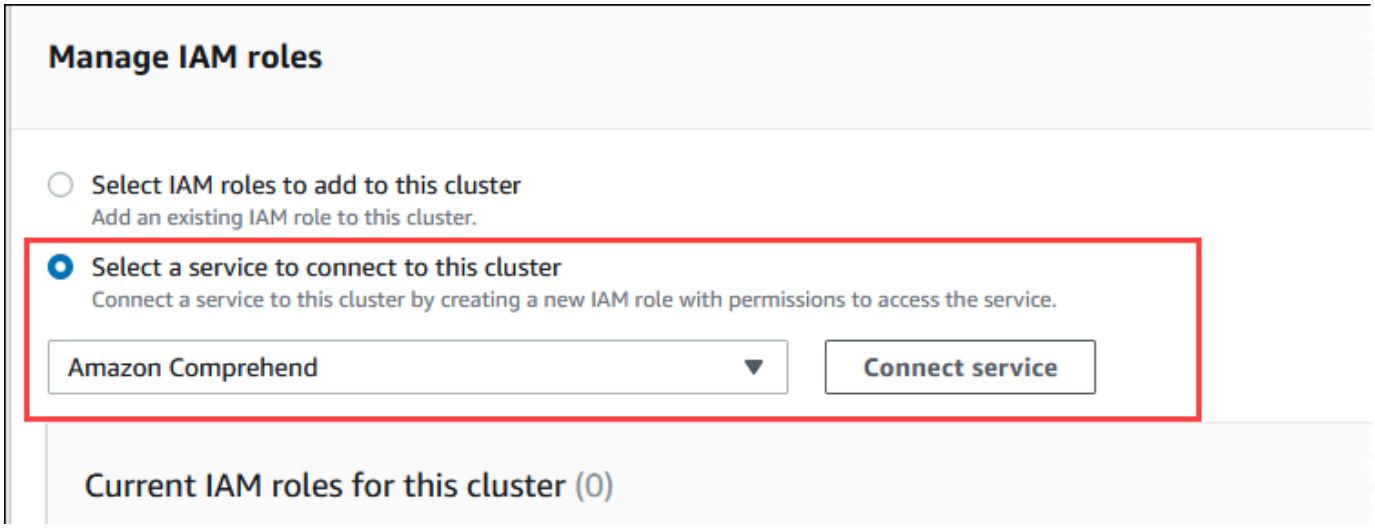
IAMIntegrasi untuk Amazon Bedrock selesai. Lanjutkan menyiapkan cluster Aurora My SQL DB Anda untuk bekerja dengan Amazon Bedrock oleh. Memberikan akses pengguna basis data ke machine learning Aurora

## Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon Comprehend

Pembelajaran mesin Aurora bergantung pada AWS Identity and Access Management peran dan kebijakan untuk memungkinkan klaster Aurora My SQL DB Anda mengakses dan menggunakan layanan Amazon Comprehend. Prosedur berikut secara otomatis membuat IAM peran dan kebijakan untuk klaster Anda sehingga dapat menggunakan Amazon Comprehend.

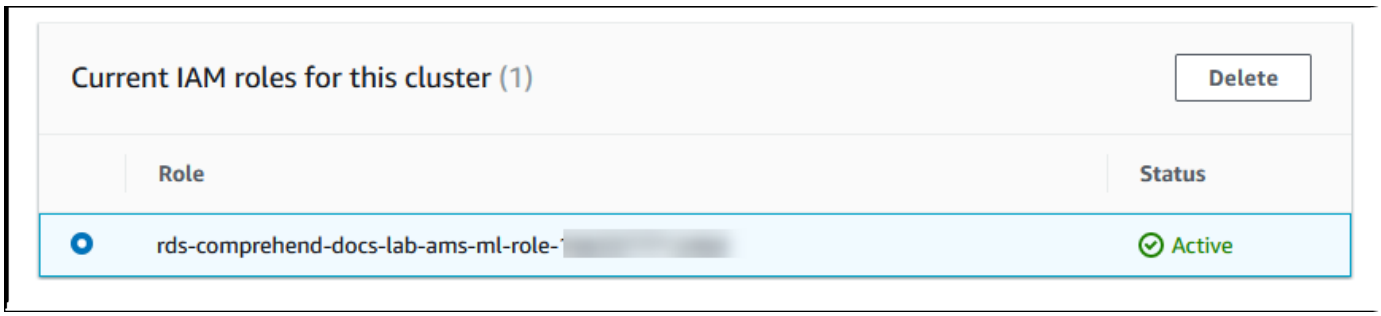
Untuk mengatur klaster Aurora My SQL DB Anda agar menggunakan Amazon Comprehend

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Basis data dari panel navigasi.
3. Pilih klaster Aurora My SQL DB yang ingin Anda sambungkan ke layanan Amazon Comprehend.
4. Pilih tab Konektivitas & keamanan.
5. Untuk bagian Kelola IAM peran, pilih Pilih layanan untuk terhubung ke klaster ini.
6. Pilih Amazon Comprehend dari menu, lalu pilih Connect service.




7. Dialog Hubungkan klaster ke Amazon Comprehend tidak memerlukan informasi tambahan apa pun. Namun, Anda mungkin melihat pesan yang memberi tahu Anda bahwa integrasi antara Aurora dan Amazon Comprehend saat ini sedang dalam pratinjau. Pastikan untuk membaca pesan tersebut sebelum melanjutkan. Anda dapat memilih Batal jika Anda memilih untuk tidak melanjutkan.
8. Pilih Hubungkan layanan untuk menyelesaikan proses integrasi.

   Aurora menciptakan peran. IAM Ini juga membuat kebijakan yang memungkinkan klaster Aurora My SQL DB menggunakan layanan Amazon Comprehend dan melampirkan kebijakan ke peran tersebut. Saat proses selesai, Anda dapat menemukan peran dalam daftar Peran saat ini IAM untuk klaster ini seperti yang ditunjukkan pada gambar berikut.

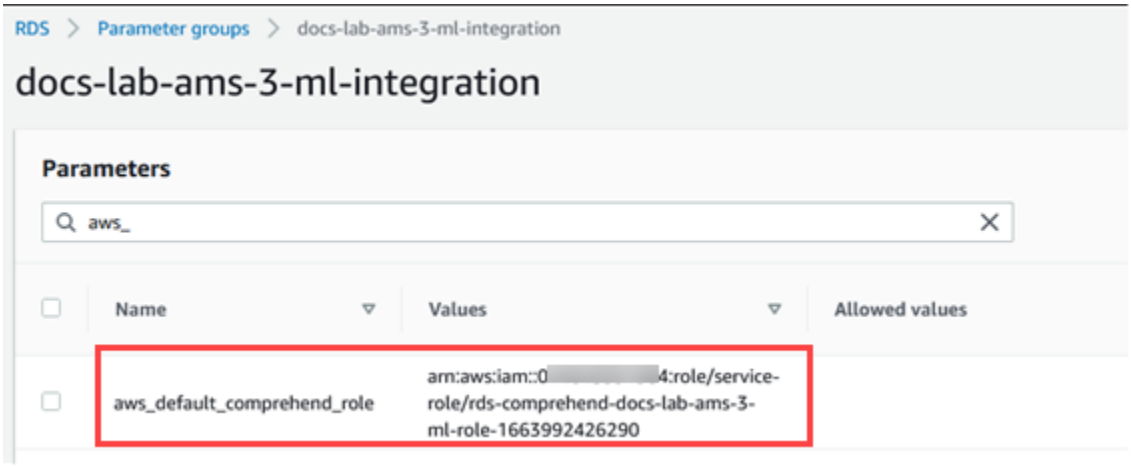



   Anda perlu menambahkan IAM peran ini ke parameter grup `aws_default_comprehend_role` parameter cluster DB kustom yang terkait dengan cluster Aurora My SQL DB Anda. ARN Jika cluster Aurora My SQL DB Anda tidak menggunakan grup parameter cluster DB kustom, Anda perlu membuatnya untuk digunakan dengan cluster Aurora My SQL DB Anda untuk menyelesaikan integrasi. Untuk informasi selengkapnya, lihat Grup parameter cluster DB untuk cluster Amazon Aurora DB.

Setelah membuat grup parameter cluster DB kustom Anda dan mengaitkannya dengan cluster Aurora SQL My DB Anda, Anda dapat melanjutkan mengikuti langkah-langkah ini.

Jika klaster Anda menggunakan grup parameter klaster DB kustom, lakukan hal berikut.

a. Di RDS Konsol Amazon, buka tab Konfigurasi klaster Aurora My SQL DB Anda.
b. Temukan grup parameter cluster DB yang dikonfigurasi untuk cluster Anda. Pilih tautan untuk membuka grup parameter cluster DB kustom Anda, lalu pilih Edit.
c. Temukan parameter `aws_default_comprehend_role` dalam grup parameter klaster DB kustom Anda.
d. Di bidang Nilai, masukkan ARN IAM peran.
e. Pilih Simpan perubahan untuk menyimpan pengaturan. Pada gambar berikut, Anda dapat melihat contohnya.



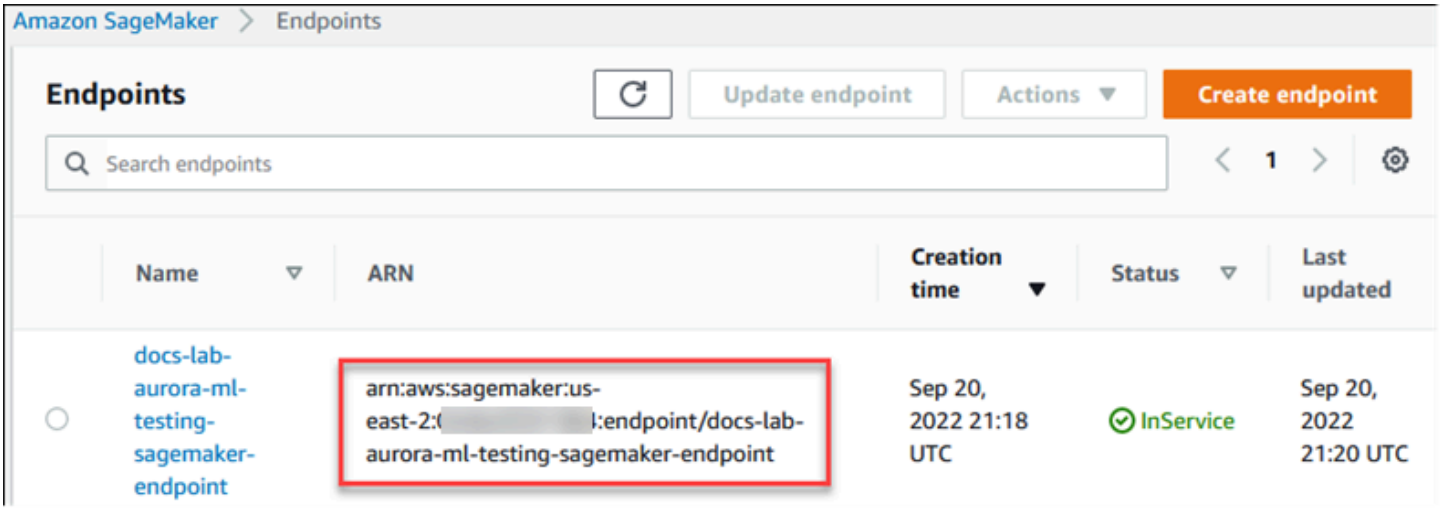

Reboot instance utama cluster Aurora My SQL DB Anda sehingga pengaturan parameter ini berlaku.

IAMIntegrasi untuk Amazon Comprehend selesai. Lanjutkan menyiapkan klaster Aurora My SQL DB Anda agar berfungsi dengan Amazon Comprehend dengan memberikan akses ke pengguna database yang sesuai.

## Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan AI SageMaker

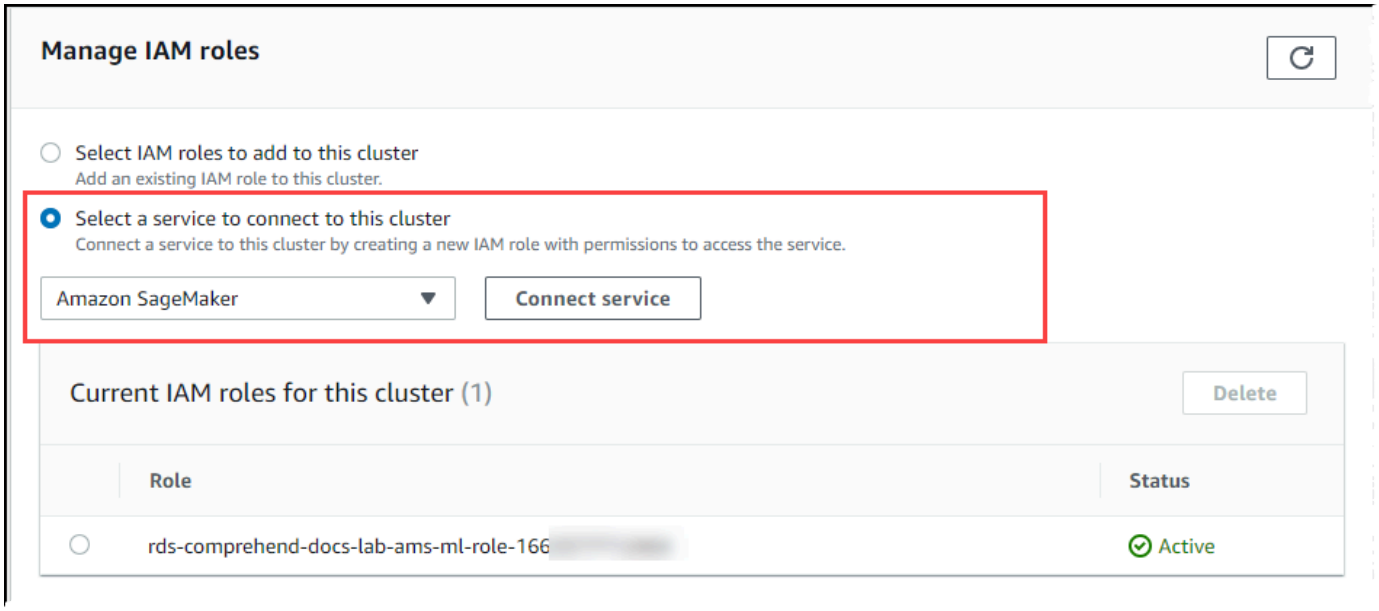
Prosedur berikut secara otomatis membuat IAM peran dan kebijakan untuk cluster Aurora My SQL DB Anda sehingga dapat menggunakan SageMaker AI. Sebelum mencoba mengikuti prosedur ini, pastikan Anda memiliki titik akhir SageMaker AI yang tersedia sehingga Anda dapat memasukkannya

saat diperlukan. Biasanya, ilmuwan data di tim Anda akan melakukan pekerjaan untuk menghasilkan titik akhir yang dapat Anda gunakan dari cluster Aurora SQL My DB Anda. Anda dapat menemukan titik akhir seperti itu di konsol SageMaker AI. Di panel navigasi, buka menu Inferensi dan pilih Titik akhir. Pada gambar berikut, Anda dapat melihat contohnya.

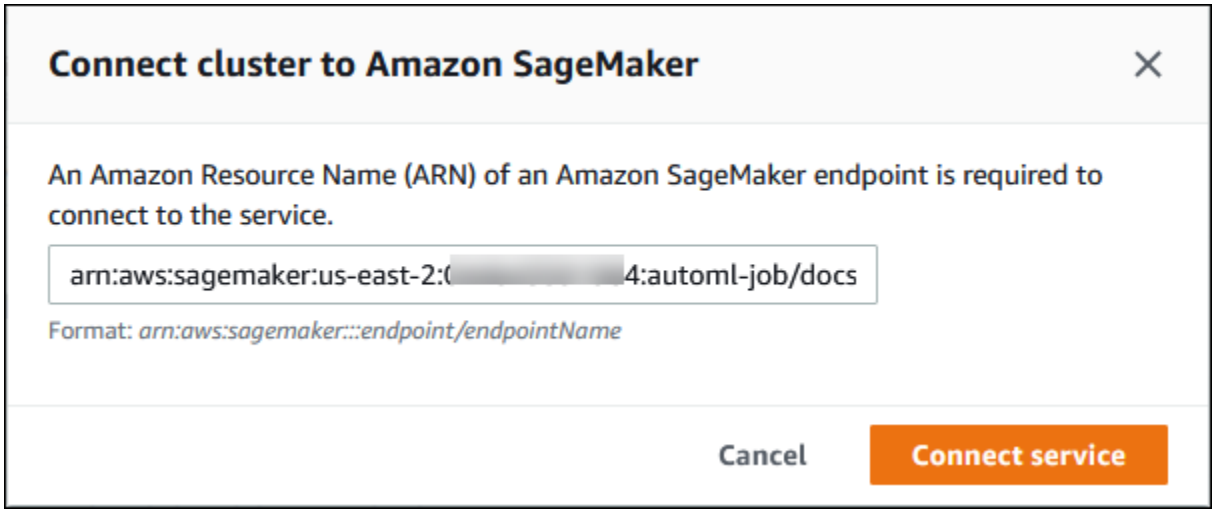



Untuk mengatur cluster Aurora My SQL DB Anda untuk menggunakan AI SageMaker

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Database dari menu RDS navigasi Amazon dan kemudian pilih cluster Aurora SQL My DB yang ingin Anda sambungkan SageMaker ke layanan AI.
3. Pilih tab Konektivitas & keamanan.
4. Gulir ke bagian Kelola IAM peran, lalu pilih Pilih layanan untuk terhubung ke klaster ini. Pilih SageMaker AI dari pemilih.




5. Pilih Hubungkan layanan.
6. Dalam dialog Connect cluster to SageMaker AI, masukkan ARN endpoint SageMaker AI.




7. Aurora menciptakan peran. IAM Ini juga menciptakan kebijakan yang memungkinkan cluster Aurora My SQL DB untuk menggunakan layanan SageMaker AI dan melampirkan kebijakan ke peran tersebut. Saat proses selesai, Anda dapat menemukan peran dalam daftar Peran saat ini IAM untuk klaster ini.
8. Buka konsol IAM di https://console.aws.amazon.com/iam/.
9. Pilih Peran dari bagian Manajemen akses pada menu navigasi AWS Identity and Access Management .
10. Temukan peran yang dicari di daftar peran. Namanya menggunakan pola berikut.

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. Buka halaman Ringkasan peran dan temukanARN. Catat ARN atau salin menggunakan widget salin.
12. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
13. Pilih klaster Aurora My SQL DB Anda, lalu pilih tab Konfigurasinya.
14. Temukan grup parameter klaster DB, dan pilih tautan untuk membuka grup parameter klaster DB kustom Anda. Temukan `aws_default_sagemaker_role` parameter dan masukkan ARN IAM peran di bidang Nilai dan Simpan pengaturan.
15. Reboot instance utama cluster Aurora My SQL DB Anda sehingga pengaturan parameter ini berlaku.

IAMPengaturan sekarang selesai. Lanjutkan menyiapkan cluster Aurora My SQL DB Anda untuk bekerja dengan SageMaker AI dengan memberikan akses ke pengguna database yang sesuai.

Jika Anda ingin menggunakan model SageMaker AI Anda untuk pelatihan daripada menggunakan komponen SageMaker AI pra-bangun, Anda juga perlu menambahkan bucket Amazon S3 ke cluster Aurora SQL My DB Anda, seperti yang diuraikan dalam berikut ini. Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon S3 SageMaker untuk AI (Opsional)

Menyiapkan cluster Aurora My SQL DB Anda untuk menggunakan Amazon S3 SageMaker untuk AI (Opsional)

Untuk menggunakan SageMaker AI dengan model Anda sendiri daripada menggunakan komponen pra-bangun yang disediakan oleh SageMaker AI, Anda perlu menyiapkan bucket Amazon S3 untuk digunakan cluster Aurora SQL My DB. Untuk informasi selengkapnya tentang cara membuat bucket Amazon S3, lihat Membuat bucket dalam Panduan Pengguna Amazon Simple Storage Service.

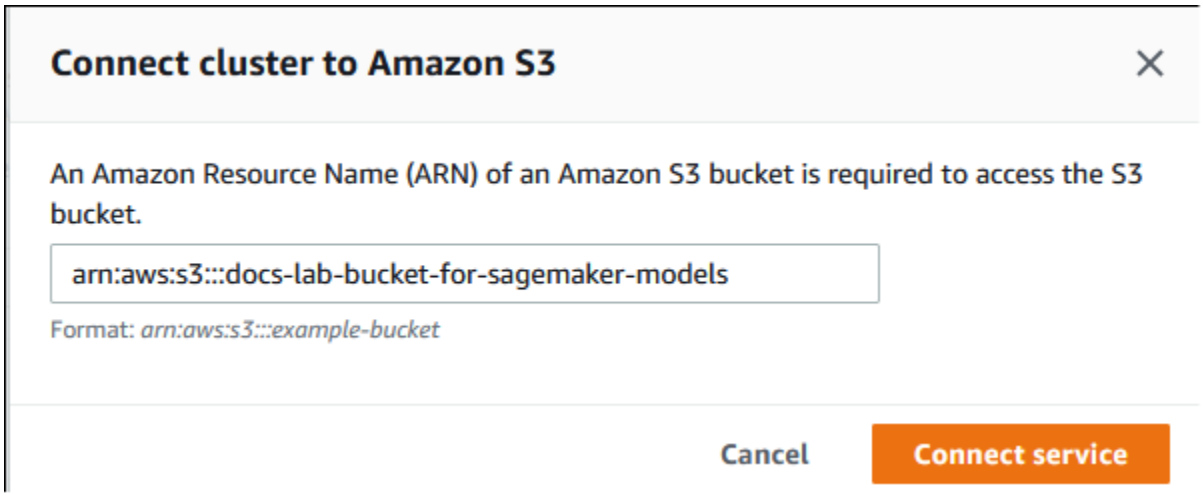
Untuk mengatur cluster Aurora My SQL DB Anda agar menggunakan bucket Amazon S3 untuk AI SageMaker

1. Masuk ke AWS Management Console dan buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
2. Pilih Database dari menu RDS navigasi Amazon dan kemudian pilih cluster Aurora SQL My DB yang ingin Anda sambungkan SageMaker ke layanan AI.
3. Pilih tab Konektivitas & keamanan.

4. Gulir ke bagian Kelola IAM peran, lalu pilih Pilih layanan untuk terhubung ke klaster ini. Pilih Amazon S3 dari pemilih.

5. Pilih Hubungkan layanan.

6. Dalam dialog Connect cluster to Amazon S3, masukkan bucket Amazon S3, seperti yang ditunjukkan pada gambar berikut. ARN




7. Pilih Hubungkan layanan untuk menyelesaikan proses ini.

Untuk informasi selengkapnya tentang menggunakan bucket Amazon S3 dengan SageMaker AI, lihat Menentukan Bucket Amazon S3 untuk Mengunggah Kumpulan Data Pelatihan dan Menyimpan Data Output di Panduan Pengembang Amazon AI. SageMaker  Untuk mempelajari selengkapnya tentang bekerja dengan SageMaker AI, lihat  SageMaker Memulai Instans Notebook Amazon AI di Panduan Pengembang Amazon SageMaker AI.

## Memberikan akses pengguna basis data ke machine learning Aurora

Pengguna database harus diberikan izin untuk menjalankan fungsi pembelajaran mesin Aurora. Cara Anda memberikan izin tergantung pada versi My SQL yang Anda gunakan untuk cluster Aurora My SQL DB Anda, sebagaimana diuraikan dalam berikut ini. Bagaimana Anda melakukannya tergantung pada versi My SQL yang digunakan cluster Aurora My SQL DB Anda.

- Untuk Aurora My SQL version 3 (My SQL 8.0 kompatibel), pengguna database harus diberikan peran database yang sesuai. Untuk informasi selengkapnya, lihat Menggunakan Peran di Manual Referensi SQL 8.0 Saya.

- Untuk Aurora My SQL version 2 (My SQL 5.7 kompatibel), pengguna database diberikan hak istimewa. Untuk informasi selengkapnya, lihat Kontrol Akses dan Manajemen Akun di Manual Referensi SQL 5.7 Saya.

Tabel berikut menunjukkan peran dan hak istimewa yang dibutuhkan pengguna database untuk bekerja dengan fungsi pembelajaran mesin.

| Aurora SQL Versi saya 3 (peran) | Aurora SQL Versi saya 2 (hak istimewa) |
|---|---|
| AWS_BEDROCK_ACCESS | – |
| AWS_COMPREHEND_ACCESS | INVOKE COMPREHEND |
| AWS_SAGEMAKER_ACCESS | INVOKE SAGEMAKER |

Memberikan akses ke fungsi Amazon Bedrock

Untuk memberi pengguna database akses ke fungsi Amazon Bedrock, gunakan SQL pernyataan berikut:

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

Pengguna database juga perlu diberikan EXECUTE izin untuk fungsi yang Anda buat untuk bekerja dengan Amazon Bedrock:

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

Terakhir, pengguna database harus mengatur peran mereka keAWS_BEDROCK_ACCESS:

```
SET ROLE AWS_BEDROCK_ACCESS;
```

Fungsi Amazon Bedrock sekarang tersedia untuk digunakan.

Memberikan akses ke fungsi Amazon Comprehend

Untuk memberi pengguna database akses ke fungsi Amazon Comprehend, gunakan pernyataan yang sesuai untuk versi Aurora Saya. SQL

- Aurora SQL Versi saya 3 (kompatibel dengan SQL 8.0 saya)

```
GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
```

- Aurora SQL Versi saya 2 (kompatibel dengan SQL 5.7 saya)

```
GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
```

Fungsi Amazon Comprehend sekarang tersedia untuk digunakan. Untuk contoh penggunaan, lihat Menggunakan Amazon Comprehend dengan cluster Aurora My DB Anda SQL.

Memberikan akses ke fungsi SageMaker AI

Untuk memberi pengguna database akses ke fungsi SageMaker AI, gunakan pernyataan yang sesuai untuk versi Aurora My SQL Anda.

- Aurora SQL Versi saya 3 (kompatibel dengan SQL 8.0 saya)

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
```

- Aurora SQL Versi saya 2 (kompatibel dengan SQL 5.7 saya)

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

Pengguna database juga perlu diberikan EXECUTE izin untuk fungsi yang Anda buat untuk bekerja dengan SageMaker AI. Misalkan Anda membuat dua fungsi, `db1.anomoly_score` dan`db2.company_forecasts`, untuk memanggil layanan titik akhir SageMaker AI Anda. Anda memberikan hak istimewa eksekusi seperti yang ditunjukkan pada contoh berikut.

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

Fungsi SageMaker AI sekarang tersedia untuk digunakan. Untuk contoh penggunaan, lihat Menggunakan SageMaker AI dengan cluster Aurora My SQL DB Anda.

# Menggunakan Amazon Bedrock dengan cluster Aurora My DB SQL Anda

Untuk menggunakan Amazon Bedrock, Anda membuat fungsi (UDF) yang ditentukan pengguna di SQL database Aurora My yang memanggil model. Untuk informasi selengkapnya, lihat Model yang didukung di Amazon Bedrock di Panduan Pengguna Amazon Bedrock.

A UDF menggunakan sintaks berikut:

```
CREATE FUNCTION function_name (argument type)
        [DEFINER = user]
    RETURNS mysql_data_type
    [SQL SECURITY {DEFINER | INVOKER}]
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'model_id'
    [CONTENT_TYPE 'content_type']
    [ACCEPT 'content_type']
    [TIMEOUT_MS timeout_in_milliseconds];
```

- Fungsi Amazon Bedrock tidak mendukungRETURNS JSON. Anda dapat menggunakan CONVERT atau CAST mengonversi dari TEXT ke JSON jika diperlukan.
- Jika Anda tidak menentukan CONTENT_TYPE atauACCEPT, defaultnya adalahapplication/json.
- Jika Anda tidak menentukanTIMEOUT_MS, nilai untuk aurora_ml_inference_timeout digunakan.

Misalnya, berikut ini UDF memanggil model Amazon Titan Text Express:

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

Untuk memungkinkan pengguna DB menggunakan fungsi ini, gunakan SQL perintah berikut:

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

Kemudian pengguna dapat memanggil `invoke_titan` seperti fungsi lainnya, seperti yang ditunjukkan pada contoh berikut. Pastikan untuk memformat badan permintaan sesuai dengan model teks Amazon Titan.

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
```
```

```
            "Price": "$15"
        },
        {
            "Row Number": "4",
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}

Untuk model lain yang Anda gunakan, pastikan untuk memformat badan permintaan dengan tepat untuk mereka. Untuk informasi selengkapnya, lihat Parameter inferensi untuk model foundation di Panduan Pengguna Amazon Bedrock.

## Menggunakan Amazon Comprehend dengan cluster Aurora My DB Anda SQL

Untuk Aurora My, pembelajaran mesin SQL Aurora menyediakan dua fungsi bawaan berikut untuk bekerja dengan Amazon Comprehend dan data teks Anda. Anda memberikan teks untuk menganalisis (`input_data`) dan menentukan bahasa (`language_code`).

aws_comprehend_detect_sentiment

Fungsi ini mengidentifikasi teks seolah-olah memiliki postur emosional positif, negatif, netral, atau campuran. Dokumentasi referensi fungsi ini adalah sebagai berikut.

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
  [,max_batch_size]
)
```

Untuk mempelajari selengkapnya, lihat Sentimen di Panduan Developer Amazon Comprehend.

aws_comprehend_detect_sentiment_confidence

Fungsi ini mengukur tingkat kepercayaan sentimen yang terdeteksi untuk teks tertentu. Ia menampilkan nilai (type, `double`) yang menunjukkan kepercayaan sentimen yang ditetapkan oleh fungsi aws_comprehend_detect_sentiment ke teks. Keyakinan adalah metrik statistik antara 0 dan 1. Makin tinggi tingkat kepercayaan, makin berat hasil yang bisa Anda berikan. Ringkasan dokumentasi fungsinya adalah sebagai berikut.

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
  [,max_batch_size]
)
```

Di kedua fungsi (aws_comprehend_detect_sentiment_confidence, aws_comprehend_detect_sentiment), `max_batch_size` menggunakan nilai default 25 jika tidak ada yang ditentukan. Ukuran Batch harus selalu lebih besar dari 0. Anda dapat menggunakan `max_batch_size` untuk menyetel performa dari panggilan fungsi Amazon Comprehend. Ukuran batch yang besar memperdagangkan kinerja yang lebih cepat untuk penggunaan memori yang lebih besar di cluster Aurora My SQL DB. Untuk informasi selengkapnya, lihat Pertimbangan kinerja untuk menggunakan pembelajaran mesin Aurora dengan Aurora My SQL.

Untuk informasi selengkapnya tentang parameter dan tipe pengembalian untuk fungsi deteksi sentimen di Amazon Comprehend, lihat DetectSentiment

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Comprehend

Berikut adalah contoh kueri sederhana yang menginvokasi dua fungsi ini untuk melihat seberapa senang pelanggan Anda dengan tim dukungan Anda. Misalkan Anda memiliki tabel basis data (`support`) yang menyimpan umpan balik pelanggan setelah setiap permintaan bantuan. Contoh kueri ini menerapkan kedua fungsi default ke teks di kolom `feedback` dari tabel dan menampilkan hasilnya. Nilai kepercayaan yang ditampilkan oleh fungsi adalah ganda, antara 0,0 dan 1,0. Untuk output yang lebih mudah dibaca, kueri ini membulatkan hasil menjadi 6 poin desimal. Untuk perbandingan yang lebih mudah, kueri ini juga mengurutkan hasil dalam urutan menurun, dari hasil yang memiliki tingkat kepercayaan tertinggi, pertama.

```
SELECT feedback AS 'Customer feedback',
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
```

```
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+-----------------------------------------------------------+-----------+------------+
| Customer feedback                                         | Sentiment | Confidence |
+-----------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!             | POSITIVE  |   0.999771 |
| The latest version of this product stinks!                | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.       | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.   | MIXED     |   0.957958 |
| Your support tech helped me in fifteen minutes.           | POSITIVE  |   0.949491 |
| My problem was never resolved!                            | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?    | NEUTRAL   |   0.902706 |
| I cannot stand that chatbot.                              | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                      | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.             | NEGATIVE  |   0.481805 |
+-----------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example Contoh: Menentukan sentimen rata-rata untuk teks di atas tingkat kepercayaan tertentu

Kueri Amazon Comprehend biasanya mencari baris di mana sentimennya adalah nilai tertentu, dengan tingkat kepercayaan yang lebih besar dari angka tertentu. Misalnya, kueri berikut ini memperlihatkan bagaimana Anda bisa menentukan sentimen rata-rata dokumen di basis data Anda. Kueri hanya mempertimbangkan dokumen dengan tingkat kepercayaan penilaian minimal 80%.

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

## Menggunakan SageMaker AI dengan cluster Aurora My SQL DB Anda

Untuk menggunakan fungsionalitas SageMaker AI dari cluster Aurora My SQL DB Anda, Anda perlu membuat fungsi tersimpan yang menyematkan panggilan Anda ke titik akhir SageMaker AI dan fitur inferensinya. Anda melakukannya dengan menggunakan My SQL secara umum dengan cara yang sama seperti yang Anda lakukan untuk tugas pemrosesan lainnya di cluster Aurora My SQL DB Anda. `CREATE FUNCTION`

Untuk menggunakan model yang diterapkan di SageMaker AI untuk inferensi, Anda membuat fungsi yang ditentukan pengguna menggunakan pernyataan My SQL data definition language (DDL) untuk fungsi yang disimpan. Setiap fungsi yang disimpan mewakili titik akhir SageMaker AI yang menghosting model. Saat Anda menentukan fungsi seperti itu, Anda menentukan parameter input ke model, titik akhir SageMaker AI spesifik yang akan dipanggil, dan jenis pengembalian. Fungsi mengembalikan inferensi yang dihitung oleh titik akhir SageMaker AI setelah menerapkan model ke parameter input.

Semua fungsi tersimpan machine learning Aurora menampilkan tipe numerik atau `VARCHAR`. Anda dapat menggunakan jenis numerik kecuali `BIT`. Jenis lainnya, seperti `JSON`, `BLOB`, `TEXT`, dan `DATE` tidak diizinkan.

Contoh berikut menunjukkan `CREATE FUNCTION` sintaks untuk bekerja dengan SageMaker AI.

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

Ini adalah perpanjangan dari `CREATE FUNCTION` DDL pernyataan reguler. Dalam `CREATE FUNCTION` pernyataan yang mendefinisikan fungsi SageMaker AI, Anda tidak menentukan badan fungsi. Alih-alih, Anda menentukan kata kunci `ALIAS` yang biasanya digunakan oleh badan fungsi. Saat ini, machine learning Aurora hanya mendukung `aws_sagemaker_invoke_endpoint` untuk sintaksis yang diperluas ini. Anda harus menentukan parameter `endpoint_name`. Titik akhir SageMaker AI dapat memiliki karakteristik yang berbeda untuk setiap model.

> **Note**
>
> Untuk informasi selengkapnya tentang`CREATE FUNCTION`, lihat CREATEPROCEDUREdan CREATE FUNCTION Pernyataan di Manual Referensi SQL 8.0 Saya.

Parameter `max_batch_size` bersifat opsional. Secara default, ukuran batch maksimum adalah 10.000. Anda dapat menggunakan parameter ini dalam fungsi Anda untuk membatasi

jumlah maksimum input yang diproses dalam permintaan batch ke AI. SageMaker `max_batch_size`Parameter dapat membantu menghindari kesalahan yang disebabkan oleh input yang terlalu besar, atau membuat SageMaker AI mengembalikan respons lebih cepat. Parameter ini memengaruhi ukuran buffer internal yang digunakan untuk pemrosesan permintaan SageMaker AI. Menentukan nilai yang terlalu besar untuk `max_batch_size` dapat menyebabkan overhead memori yang besar pada instans DB Anda.

Sebaiknya biarkan pengaturan `MANIFEST` pada nilai default `OFF`. Meskipun Anda dapat menggunakan `MANIFEST ON` opsi ini, beberapa fitur SageMaker AI tidak dapat langsung menggunakan yang CSV diekspor dengan opsi ini. Format manifes tidak kompatibel dengan format manifes yang diharapkan dari SageMaker AI.

Anda membuat fungsi tersimpan terpisah untuk setiap model SageMaker AI Anda. Pemetaan fungsi ke model ini diperlukan karena titik akhir dikaitkan dengan model tertentu, dan setiap model menerima parameter yang berbeda. Menggunakan SQL tipe untuk input model dan tipe keluaran model membantu menghindari kesalahan konversi tipe yang meneruskan data bolak-balik antara AWS layanan. Anda dapat mengontrol siapa yang dapat menerapkan model tersebut. Anda juga dapat mengontrol karakteristik runtime dengan menentukan parameter yang mewakili ukuran batch maksimum.

Saat ini, semua fungsi machine learning Aurora memiliki properti `NOT DETERMINISTIC`. Jika Anda tidak menentukan properti tersebut secara eksplisit, Aurora menyetel `NOT DETERMINISTIC` secara otomatis. Persyaratan ini karena model SageMaker AI dapat diubah tanpa pemberitahuan apa pun ke database. Jika itu terjadi, panggilan ke fungsi machine learning Aurora mungkin menampilkan hasil yang berbeda untuk input yang sama dalam satu transaksi.

Anda tidak dapat menggunakan karakteristik `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, atau `MODIFIES SQL DATA` dalam pernyataan `CREATE FUNCTION` Anda.

Berikut ini adalah contoh penggunaan titik akhir SageMaker AI untuk mendeteksi anomali. Ada titik akhir SageMaker `random-cut-forest-model` AI. Model yang sesuai sudah dilatih oleh algoritma `random-cut-forest`. Untuk setiap input, model menampilkan skor anomali. Contoh ini menunjukkan poin data yang nilainya lebih besar dari 3 deviasi standar (kira-kira persentil ke-99,9) dari skor rata-rata.

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);
```

```
select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

## Persyaratan set karakter untuk fungsi SageMaker AI yang mengembalikan string

Sebaiknya tentukan kumpulan karakter `utf8mb4` sebagai tipe pengembalian untuk fungsi SageMaker AI Anda yang mengembalikan nilai string. Jika itu tidak praktis, gunakan panjang string yang cukup besar agar jenis kembalian dapat menampung nilai yang direpresentasikan dalam kumpulan karakter `utf8mb4`. Contoh berikut menunjukkan cara mendeklarasikan set karakter `utf8mb4` untuk fungsi Anda.

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

Saat ini, setiap fungsi SageMaker AI yang mengembalikan string menggunakan set karakter `utf8mb4` untuk nilai yang dikembalikan. Nilai yang dikembalikan menggunakan set karakter ini meskipun fungsi SageMaker AI Anda mendeklarasikan set karakter yang berbeda untuk tipe pengembaliannya secara implisit atau eksplisit. Jika fungsi SageMaker AI Anda mendeklarasikan set karakter yang berbeda untuk nilai yang dikembalikan, data yang dikembalikan mungkin terpotong secara diam-diam jika Anda menyimpannya di kolom tabel yang tidak cukup panjang. Misalnya, kueri dengan klausa `DISTINCT` membuat tabel sementara. Dengan demikian, hasil fungsi SageMaker AI mungkin terpotong karena cara string ditangani secara internal selama kueri.

## Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan)

Kami menyarankan Anda memulai pembelajaran mesin Aurora dan SageMaker AI dengan menggunakan beberapa algoritme yang disediakan, dan bahwa ilmuwan data di tim Anda memberi Anda titik akhir SageMaker AI yang dapat Anda gunakan dengan kode Anda. SQL Berikut ini, Anda dapat menemukan informasi minimal tentang penggunaan bucket Amazon S3 Anda sendiri dengan model SageMaker AI Anda sendiri dan cluster Aurora My DB Anda. SQL

Machine learning terdiri dari dua langkah utama: pelatihan, dan inferensi. Untuk melatih model SageMaker AI, Anda mengekspor data ke bucket Amazon S3. Bucket Amazon S3 digunakan oleh instance notebook Jupyter SageMaker AI untuk melatih model Anda sebelum digunakan. Anda dapat menggunakan `SELECT INTO OUTFILE S3` pernyataan untuk menanyakan data dari cluster Aurora My SQL DB dan menyimpannya langsung ke file teks yang disimpan dalam bucket Amazon S3. Kemudian instans notebook menggunakan data dari bucket Amazon S3 untuk pelatihan.

Pembelajaran mesin Aurora memperluas `SELECT INTO OUTFILE` sintaks yang ada di Aurora SQL My untuk mengekspor data ke format. CSV CSVFile yang dihasilkan dapat langsung dikonsumsi oleh model yang membutuhkan format ini untuk tujuan pelatihan.

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

Ekstensi mendukung CSV format standar.

- `TEXT`Formatnya sama dengan format SQL ekspor Saya yang ada. Ini adalah format default.
- Format `CSV` adalah format yang baru diperkenalkan yang mengikuti spesifikasi di RFC-4180.
- Jika Anda menentukan kata kunci opsional `HEADER`, file output akan berisi satu baris header. Label di baris header sesuai dengan nama kolom dari pernyataan `SELECT`.
- Anda masih dapat menggunakan kata kunci `CSV` dan `HEADER` sebagai pengenal.

Sintaks dan tata bahasa `SELECT INTO` yang diperluas sekarang adalah sebagai berikut:

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

## Pertimbangan kinerja untuk menggunakan pembelajaran mesin Aurora dengan Aurora My SQL

Amazon Bedrock, Amazon Comprehend SageMaker , dan layanan AI melakukan sebagian besar pekerjaan ketika dipanggil oleh fungsi pembelajaran mesin Aurora. Itu berarti Anda dapat menskalakan sumber daya tersebut sesuai kebutuhan, secara mandiri. Untuk cluster Aurora My SQL DB Anda, Anda dapat membuat panggilan fungsi Anda seefisien mungkin. Berikut ini, Anda dapat menemukan beberapa pertimbangan performa yang perlu diperhatikan saat bekerja dengan machine learning Aurora.

## Model dan prompt

Performa saat menggunakan Amazon Bedrock sangat bergantung pada model dan prompt yang Anda gunakan. Pilih model dan prompt yang optimal untuk kasus penggunaan Anda.

## Cache kueri

Cache SQL kueri Aurora My tidak berfungsi untuk fungsi pembelajaran mesin Aurora. Aurora My SQL tidak menyimpan hasil kueri di cache kueri untuk SQL pernyataan apa pun yang memanggil fungsi pembelajaran mesin Aurora.

## Optimalisasi batch untuk panggilan fungsi machine learning Aurora

Aspek performa utama machine learning Aurora yang dapat Anda pengaruhi dari klaster Aurora adalah pengaturan mode batch untuk panggilan ke fungsi tersimpan machine learning Aurora. Fungsi machine learning biasanya membutuhkan overhead yang besar, sehingga tidak praktis untuk memanggil layanan eksternal secara terpisah untuk setiap baris. Machine learning Aurora dapat meminimalkan overhead ini dengan menggabungkan panggilan ke layanan machine learning Aurora eksternal untuk banyak baris ke dalam satu batch. Machine learning Aurora menerima respons untuk semua baris input, dan mengirimkan respons, satu baris dalam satu waktu, ke kueri saat dijalankan. Pengoptimalan ini meningkatkan throughput dan latensi kueri Aurora Anda tanpa mengubah hasil.

Saat Anda membuat fungsi tersimpan Aurora yang terhubung ke titik akhir SageMaker AI, Anda menentukan parameter ukuran batch. Parameter ini memengaruhi berapa banyak baris yang ditransfer untuk setiap panggilan yang mendasarinya ke SageMaker AI. Untuk kueri yang memproses sejumlah besar baris, overhead untuk membuat panggilan SageMaker AI terpisah untuk setiap baris bisa sangat besar. Semakin besar kumpulan data yang diproses oleh prosedur tersimpan, semakin besar Anda dapat membuat ukuran batch.

Jika pengoptimalan mode batch dapat diterapkan ke fungsi SageMaker AI, Anda dapat mengetahuinya dengan memeriksa paket kueri yang dihasilkan oleh `EXPLAIN PLAN` pernyataan tersebut. Dalam kasus ini, kolom `extra` dalam rencana eksekusi termasuk `Batched machine learning`. Contoh berikut menunjukkan panggilan ke fungsi SageMaker AI yang menggunakan mode batch.

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+----+-------------+----------+------------+------+--------------+------+---------
+------+------+----------+-------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                   |
+----+-------------+----------+------------+------+--------------+------+---------
+------+------+----------+-------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
+----+-------------+----------+------------+------+--------------+------+---------
+------+------+----------+-------------------------+
1 row in set, 1 warning (0.01 sec)
```

Saat Anda memanggil salah satu fungsi Amazon Comprehend default, Anda dapat mengontrol ukuran batch dengan menentukan parameter `max_batch_size` opsional. Parameter ini membatasi jumlah maksimum nilai `input_text` yang diproses di setiap batch. Dengan mengirim beberapa item sekaligus, ini mengurangi jumlah perjalanan bolak-balik antara Aurora dan Amazon Comprehend. Membatasi ukuran batch berguna dalam situasi seperti kueri dengan klausa `LIMIT`. Dengan menggunakan nilai kecil untuk `max_batch_size`, Anda dapat menghindari permintaan Amazon Comprehend lebih sering daripada Anda memiliki teks input.

Optimalisasi batch untuk mengevaluasi fungsi machine learning Aurora berlaku dalam kasus berikut:

- Fungsi panggilan dalam daftar pilih atau `WHERE` klausa pernyataan `SELECT`
- Panggilan fungsi dalam `VALUES` daftar `INSERT` dan `REPLACE` pernyataan
- SageMaker Fungsi AI dalam `SET` nilai dalam `UPDATE` pernyataan:

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

## Memantau machine learning Aurora

Anda dapat memantau operasi batch pembelajaran mesin Aurora dengan menanyakan beberapa variabel global, seperti yang ditunjukkan pada contoh berikut.

```
show status like 'Aurora_ml%';
```

Anda dapat mengatur ulang variabel status dengan menggunakan pernyataan `FLUSH STATUS`. Jadi, semua angka mewakili total, rata-rata, dan seterusnya, sejak terakhir kali variabel disetel ulang.

`Aurora_ml_logical_request_cnt`

Jumlah permintaan logis yang telah dievaluasi instans DB untuk dikirim ke layanan machine learning Aurora sejak status pengaturan ulang status terakhir. Bergantung pada apakah proses batching telah digunakan, nilai ini dapat lebih tinggi dari `Aurora_ml_actual_request_cnt`.

`Aurora_ml_logical_response_cnt`

Jumlah respons agregat yang diterima Aurora SQL My dari layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_actual_request_cnt`

Jumlah permintaan agregat yang dibuat Aurora SQL My ke layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_actual_response_cnt`

Jumlah respons agregat yang diterima Aurora SQL My dari layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_cache_hit_cnt`

Jumlah cache internal agregat yang diterima Aurora SQL My dari layanan pembelajaran mesin Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_retry_request_cnt`

Jumlah permintaan yang dicoba ulang yang dikirim instans DB ke layanan machine learning Aurora sejak pengaturan ulang status terakhir.

`Aurora_ml_single_request_cnt`

Jumlah gabungan fungsi machine learning Aurora yang dievaluasi oleh mode non-batch di semua kueri yang dijalankan oleh pengguna instans DB.

Untuk informasi tentang pemantauan kinerja operasi SageMaker AI yang disebut dari fungsi pembelajaran mesin Aurora, lihat Memantau Amazon SageMaker AI.

# Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora Postgre SQL

Dengan menggunakan pembelajaran mesin Amazon Aurora dengan cluster SQL DB Aurora Postgre Anda, Anda dapat menggunakan Amazon Comprehend atau Amazon AI atau Amazon Bedrock, tergantung pada kebutuhan Anda. SageMaker Layanan ini masing-masing mendukung kasus penggunaan pembelajaran mesin tertentu.

Pembelajaran mesin Aurora didukung dalam versi tertentu Wilayah AWS dan untuk Aurora Postgre tertentu saja. SQL Sebelum mencoba mengatur pembelajaran mesin Aurora, periksa ketersediaan untuk versi Aurora SQL Postgre dan Wilayah Anda. Untuk detailnya, lihat Pembelajaran mesin Aurora dengan Aurora Postgre SQL.

Topik


- Persyaratan untuk menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL
- Fitur yang didukung dan keterbatasan pembelajaran mesin Aurora dengan Aurora Postgre SQL
- Menyiapkan cluster Aurora Postgre SQL DB Anda untuk menggunakan pembelajaran mesin Aurora
- Menggunakan Amazon Bedrock dengan cluster DB Aurora Postgre SQL Anda
- Menggunakan Amazon Comprehend dengan cluster DB Aurora Postgre Anda SQL
- Menggunakan SageMaker AI dengan cluster Aurora SQL Postgre DB Anda
- Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan)
- Pertimbangan kinerja untuk menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL
- Memantau machine learning Aurora


## Persyaratan untuk menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL

AWS Layanan pembelajaran mesin adalah layanan terkelola yang diatur dan dijalankan di lingkungan produksi mereka sendiri. Pembelajaran mesin Aurora mendukung integrasi dengan Amazon Comprehend, AI, dan Amazon SageMaker Bedrock. Sebelum mencoba mengatur cluster Aurora Postgre SQL DB Anda untuk menggunakan pembelajaran mesin Aurora, pastikan Anda memahami persyaratan dan prasyarat berikut.

- Layanan Amazon SageMaker Comprehend, AI, dan Amazon Bedrock harus berjalan Wilayah AWS sama dengan cluster Aurora Postgre DB Anda. SQL Anda tidak dapat menggunakan layanan Amazon SageMaker Comprehend atau AI atau Amazon Bedrock dari klaster DB SQL Aurora Postgre di Wilayah lain.
- Jika cluster Aurora Postgre SQL DB Anda berada di cloud publik virtual (VPC) yang berbeda berdasarkan layanan Amazon daripada VPC layanan Amazon Comprehend dan SageMaker AI Anda, grup Keamanan perlu mengizinkan koneksi keluar ke layanan VPC pembelajaran mesin Aurora target. Untuk informasi selengkapnya, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora ke layanan lain AWS.
- Untuk SageMaker AI, komponen pembelajaran mesin yang ingin Anda gunakan untuk inferensi harus diatur dan siap digunakan. Selama proses konfigurasi untuk cluster Aurora Postgre SQL DB Anda, Anda harus memiliki Amazon Resource Name (ARN) dari titik akhir AI yang tersedia. SageMaker Ilmuwan data di tim Anda kemungkinan besar paling mampu menangani bekerja dengan SageMaker AI untuk mempersiapkan model dan menangani tugas-tugas semacam itu lainnya. Untuk memulai dengan Amazon SageMaker AI, lihat Memulai dengan Amazon SageMaker AI. Untuk informasi selengkapnya tentang inferensi dan titik akhir, lihat Inferensi real-time.
- Untuk Amazon Bedrock, Anda harus memiliki ID model model Bedrock yang ingin Anda gunakan untuk inferensi yang tersedia selama proses konfigurasi cluster DB Aurora Postgre Anda. SQL Ilmuwan data di tim Anda kemungkinan besar paling mampu bekerja dengan Bedrock untuk memutuskan model mana yang akan digunakan, menyempurnakannya jika diperlukan, dan menangani tugas-tugas semacam itu lainnya. Untuk memulai dengan Amazon Bedrock, lihat Cara mengatur Bedrock.
- Pengguna Amazon Bedrock perlu meminta akses ke model sebelum tersedia untuk digunakan. Jika Anda ingin menambahkan model tambahan untuk pembuatan teks, obrolan, dan gambar, Anda perlu meminta akses ke model di Amazon Bedrock. Untuk informasi selengkapnya, lihat Akses model.

## Fitur yang didukung dan keterbatasan pembelajaran mesin Aurora dengan Aurora Postgre SQL

Pembelajaran mesin Aurora mendukung titik akhir SageMaker AI apa pun yang dapat membaca dan menulis format nilai yang dipisahkan koma (CSV) melalui nilai. `ContentType` `text/csv` Algoritma SageMaker AI bawaan yang saat ini menerima format ini adalah sebagai berikut.

- Linear Learner

- Random Cut Forest
- XGBoost

  Untuk mempelajari lebih lanjut tentang algoritme ini, lihat Memilih Algoritma di Panduan Pengembang Amazon SageMaker AI.

Saat menggunakan Amazon Bedrock dengan pembelajaran mesin Aurora, batasan berikut berlaku:

- Fungsi yang ditentukan pengguna (UDFs) menyediakan cara asli untuk berinteraksi dengan Amazon Bedrock. UDFsTidak memiliki permintaan atau persyaratan respons khusus, sehingga mereka dapat menggunakan model apa pun.
- Anda dapat menggunakan UDFs untuk membangun alur kerja yang diinginkan. Misalnya, Anda dapat menggabungkan primitif dasar seperti `pg_cron` menjalankan kueri, mengambil data, menghasilkan kesimpulan, dan menulis ke tabel untuk menyajikan kueri secara langsung.
- UDFstidak mendukung panggilan batch atau parallel.
- Ekstensi Machine Learning Aurora tidak mendukung antarmuka vektor. Sebagai bagian dari ekstensi, fungsi tersedia untuk menampilkan penyematan respons model dalam `float8[]` format untuk menyimpan embeddings tersebut di Aurora. Untuk informasi lebih lanjut tentang penggunaan`float8[]`, lihatMenggunakan Amazon Bedrock dengan cluster DB Aurora Postgre SQL Anda.

## Menyiapkan cluster Aurora Postgre SQL DB Anda untuk menggunakan pembelajaran mesin Aurora

Agar pembelajaran mesin Aurora dapat bekerja dengan cluster SQL DB Aurora Postgre Anda, Anda perlu membuat peran AWS Identity and Access Management (IAM) untuk setiap layanan yang ingin Anda gunakan. IAMPeran ini memungkinkan cluster Aurora Postgre SQL DB Anda untuk menggunakan layanan pembelajaran mesin Aurora atas nama cluster. Anda juga perlu menginstal ekstensi machine learning Aurora. Dalam topik berikut, Anda dapat menemukan prosedur penyiapan untuk masing-masing layanan machine learning Aurora ini.

Topik

- Menyiapkan Aurora Postgre untuk SQL menggunakan Amazon Bedrock
- Menyiapkan Aurora Postgre SQL untuk menggunakan Amazon Comprehend
- Menyiapkan Aurora Postgre untuk menggunakan Amazon AI SQL SageMaker

- Menyiapkan Aurora Postgre untuk menggunakan SQL Amazon S3 untuk AI (Lanjutan) SageMaker
- Menginstal ekstensi machine learning Aurora

## Menyiapkan Aurora Postgre untuk SQL menggunakan Amazon Bedrock

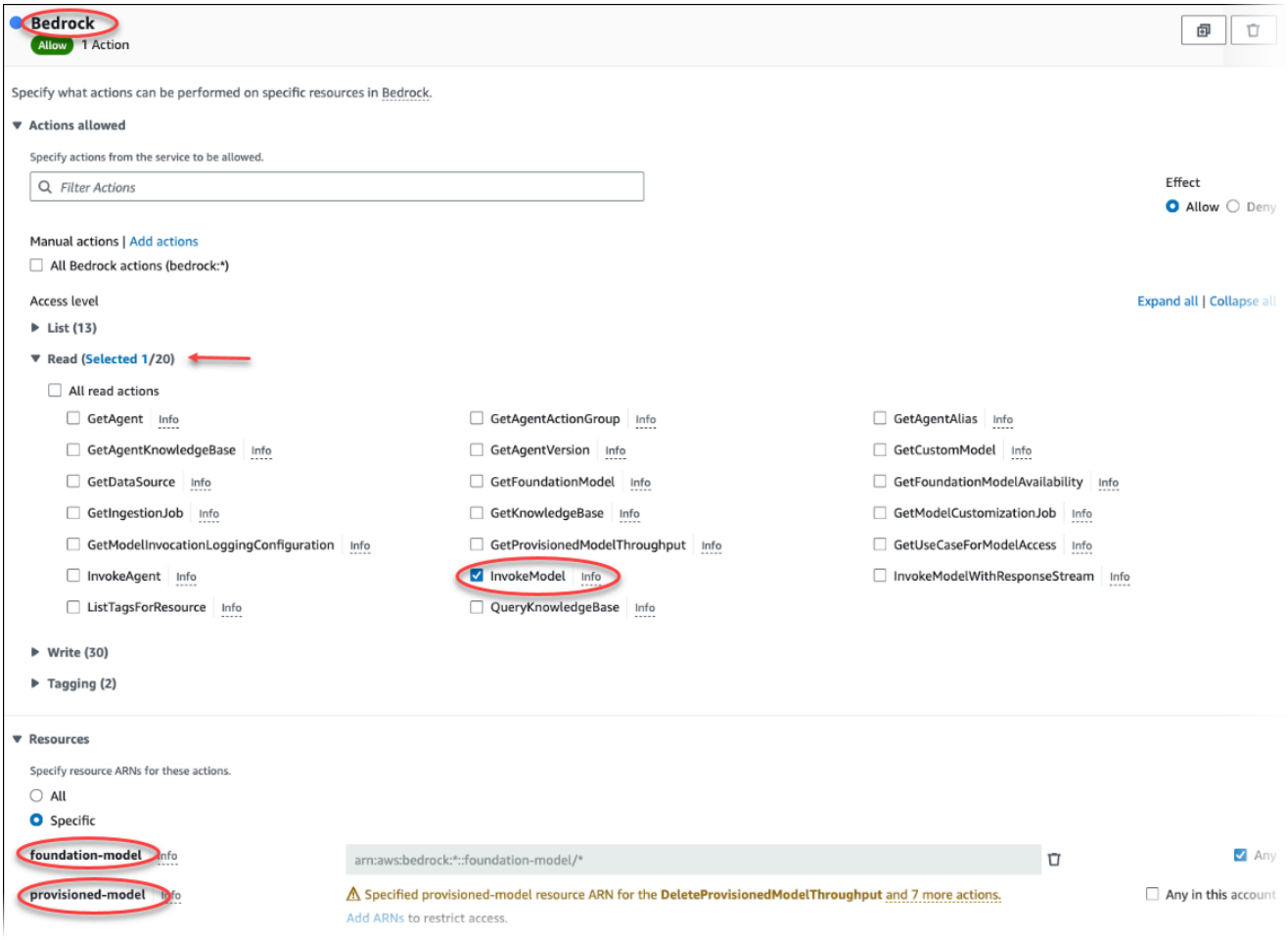
Dalam prosedur berikut, pertama-tama Anda membuat IAM peran dan kebijakan yang memberikan izin kepada Aurora Postgre Anda SQL untuk menggunakan Amazon Bedrock atas nama klaster. Anda kemudian melampirkan kebijakan ke IAM peran yang digunakan klaster SQL DB Aurora Postgre Anda untuk bekerja dengan Amazon Bedrock. Demi kesederhanaan, prosedur ini menggunakan AWS Management Console untuk menyelesaikan semua tugas.

Untuk mengatur cluster Aurora Postgre SQL DB Anda untuk menggunakan Amazon Bedrock

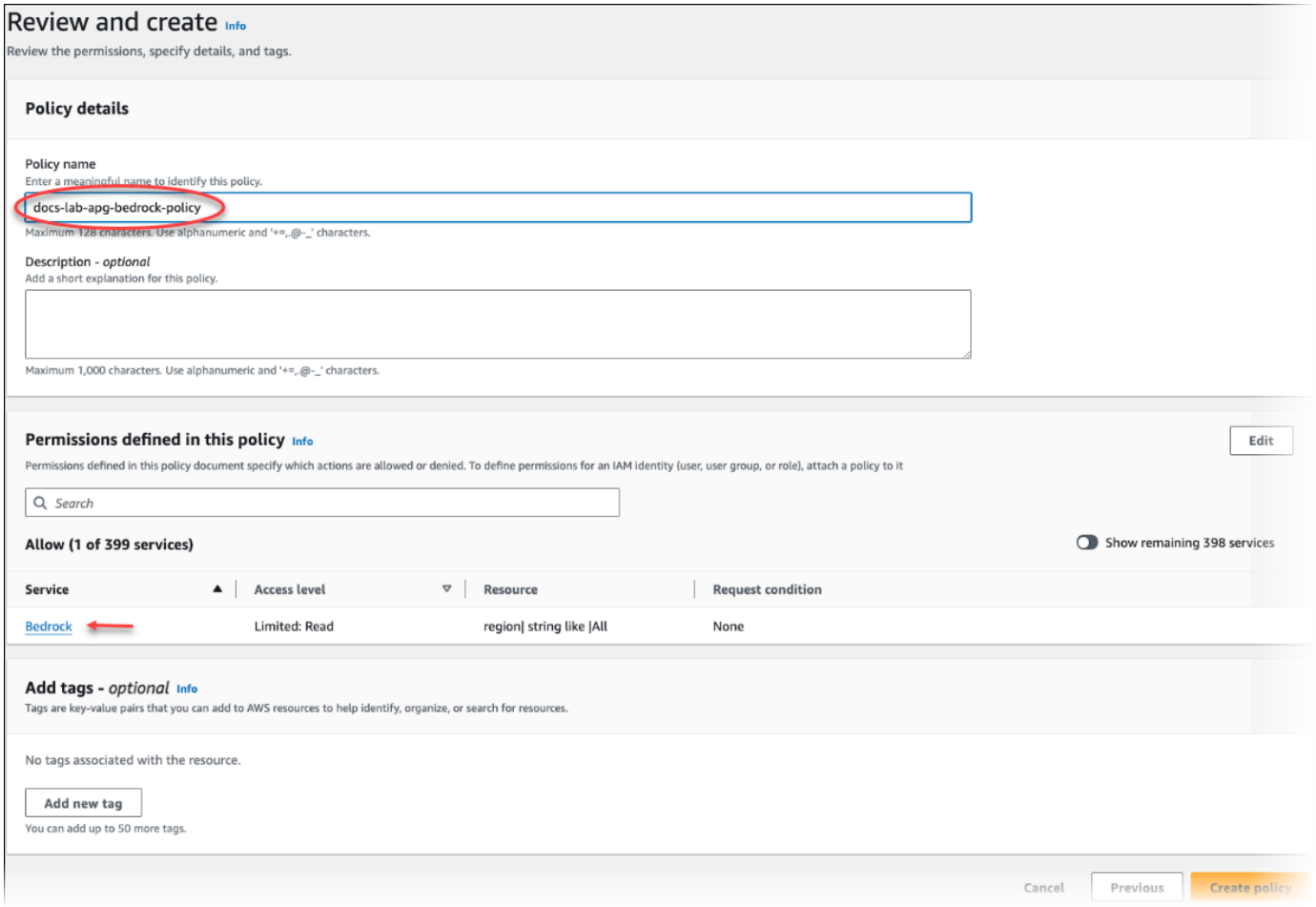
1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Buka konsol IAM di https://console.aws.amazon.com/iam/.
3. Pilih Kebijakan (di bawah Manajemen akses) pada menu AWS Identity and Access Management (IAM) Konsol.

    a. Pilih Buat kebijakan. Di halaman Editor Visual, pilih Layanan dan kemudian masukkan Bedrock di bidang Pilih layanan. Perluas tingkat akses Baca. Pilih InvokeModeldari pengaturan baca Amazon Bedrock.

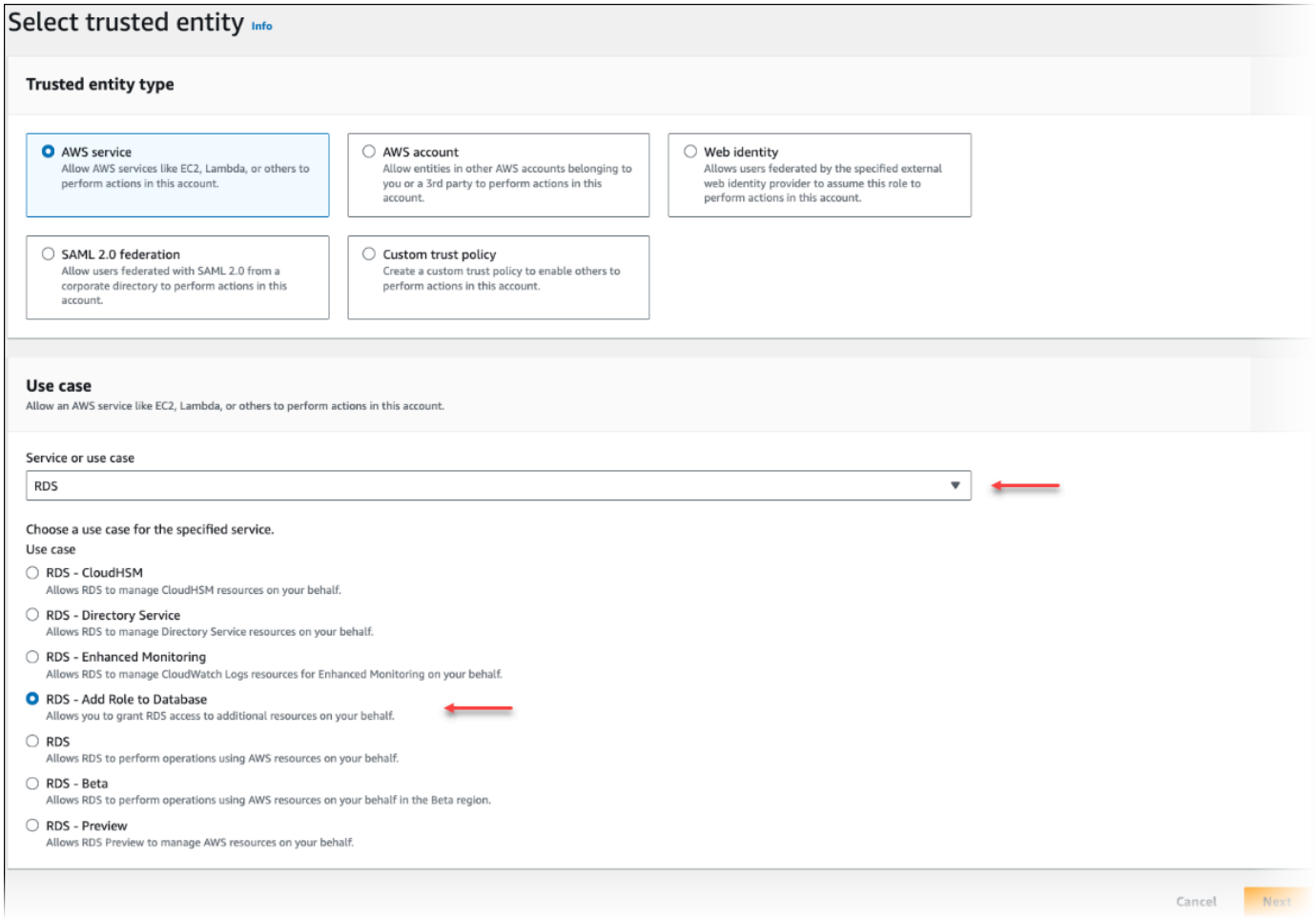
    b. Pilih model Foundation/Provisioned yang ingin Anda berikan akses baca melalui kebijakan.




4. Pilih Berikutnya: Tag dan tentukan tag apa pun (opsional). Pilih Berikutnya: Peninjauan. Masukkan nama untuk kebijakan dan deskripsi, seperti diperlihatkan pada gambar.

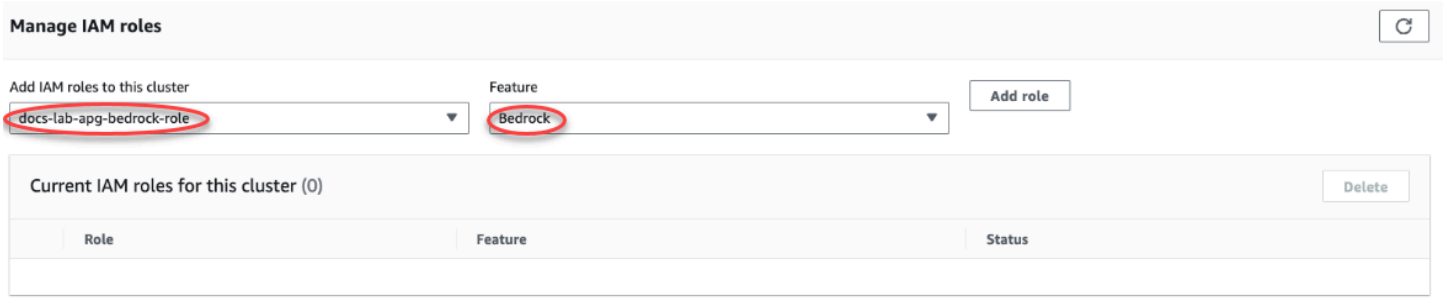



5. Pilih Buat kebijakan. Konsol menampilkan peringatan saat kebijakan telah disimpan. Anda dapat menemukannya di daftar Kebijakan.
6. Pilih Peran (di bawah Manajemen akses) di IAM Konsol.
7. Pilih Buat peran.
8. Pada halaman Pilih entitas tepercaya, pilih ubin AWS layanan, lalu pilih RDSuntuk membuka pemilih.
9. Pilih RDS— Tambahkan Peran ke Database.




10. Pilih Berikutnya. Di halaman Tambahkan izin, temukan kebijakan yang Anda buat di langkah sebelumnya dan pilih kebijakan dari daftar tersebut. Pilih Berikutnya.
11. Berikutnya: Tinjau. Masukkan nama untuk IAM peran dan deskripsi.
12. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
13. Arahkan ke Wilayah AWS tempat cluster Aurora Postgre SQL DB Anda berada.
14. Di panel navigasi, pilih Databases, dan kemudian pilih cluster Aurora Postgre SQL DB yang ingin Anda gunakan dengan Bedrock.
15. Pilih tab Konektivitas & keamanan dan gulir untuk menemukan bagian Kelola IAM peran pada halaman. Dari Tambahkan IAM peran ke pemilih klaster ini, pilih peran yang Anda buat di langkah sebelumnya. Di pemilih Fitur, pilih Batuan Dasar, lalu pilih Tambah peran.

Peran (dengan kebijakannya) dikaitkan dengan cluster Aurora SQL Postgre DB. Saat proses selesai, peran tersebut tercantum dalam Peran saat ini IAM untuk daftar klaster ini, seperti yang ditunjukkan berikut.



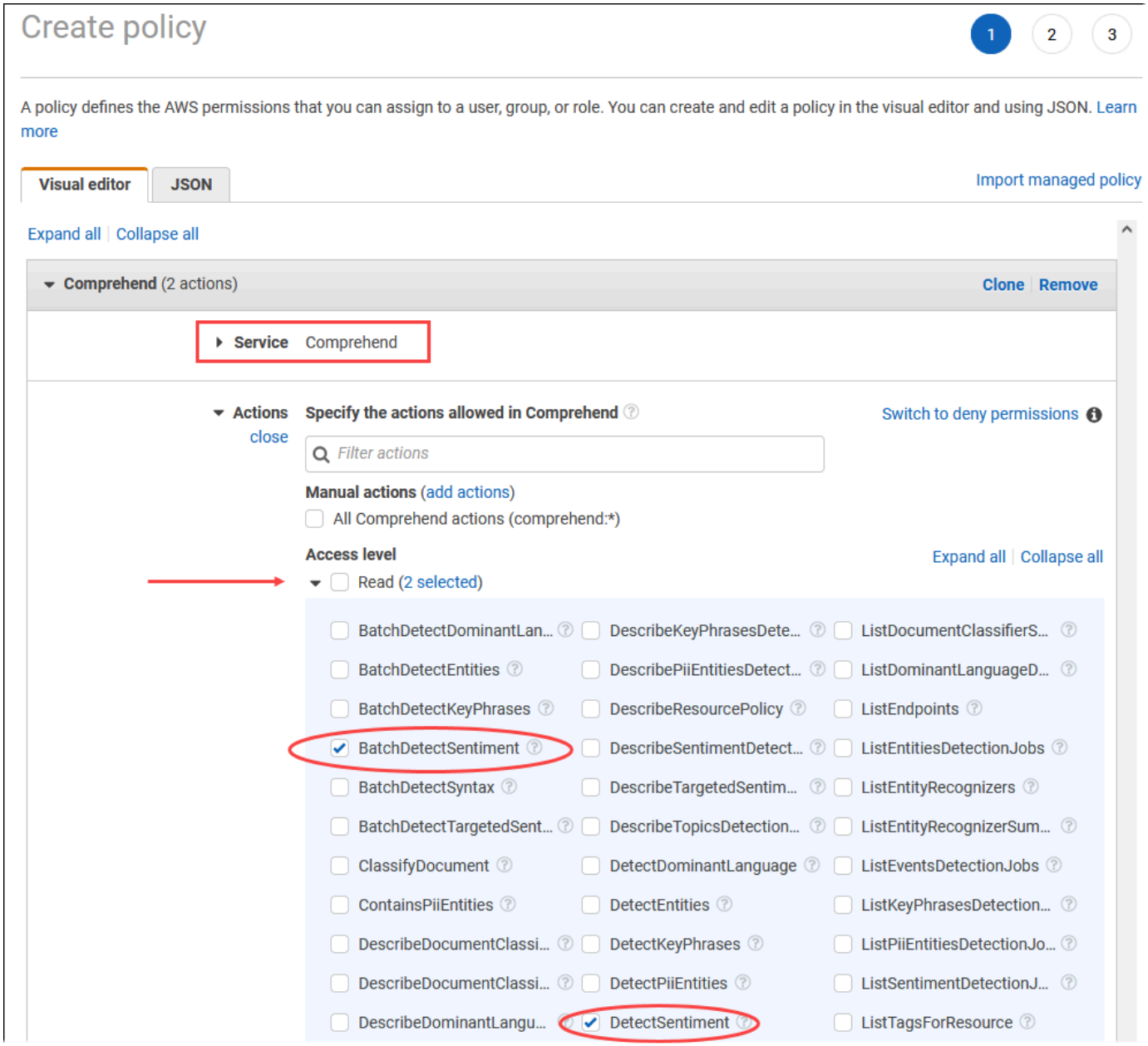

IAMPengaturan untuk Amazon Bedrock selesai. Lanjutkan menyiapkan Aurora Postgre Anda untuk SQL bekerja dengan pembelajaran mesin Aurora dengan menginstal ekstensi seperti yang dijelaskan dalam Menginstal ekstensi machine learning Aurora

## Menyiapkan Aurora Postgre SQL untuk menggunakan Amazon Comprehend

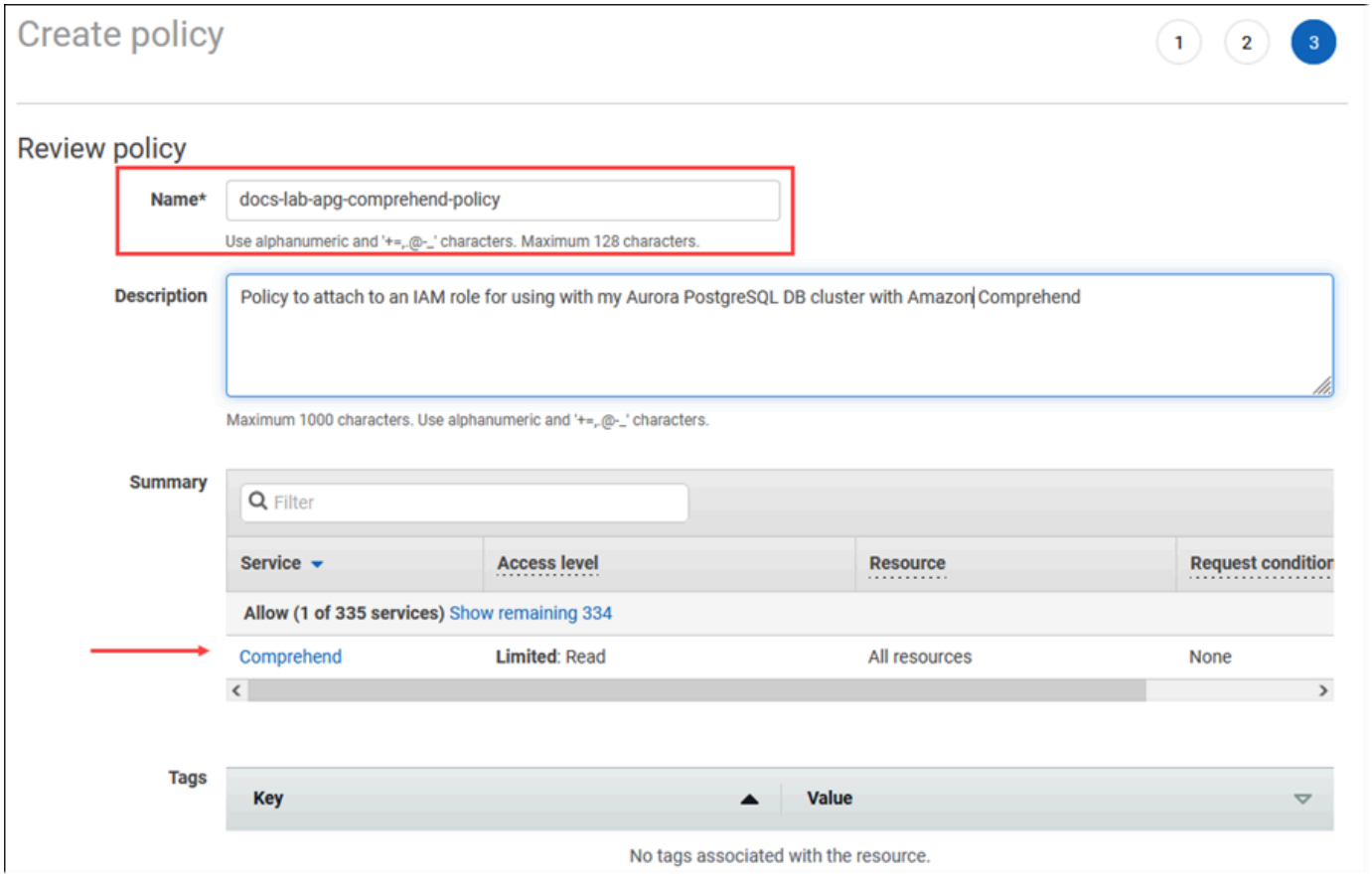
Dalam prosedur berikut, pertama-tama Anda membuat IAM peran dan kebijakan yang memberikan SQL izin kepada Aurora Postgre Anda untuk menggunakan Amazon Comprehend atas nama klaster. Anda kemudian melampirkan kebijakan ke IAM peran yang digunakan cluster Aurora Postgre SQL DB Anda untuk bekerja dengan Amazon Comprehend Demi kesederhanaan, prosedur ini menggunakan untuk menyelesaikan semua tugas. AWS Management Console

Untuk mengatur cluster Aurora Postgre SQL DB Anda untuk menggunakan Amazon Comprehend

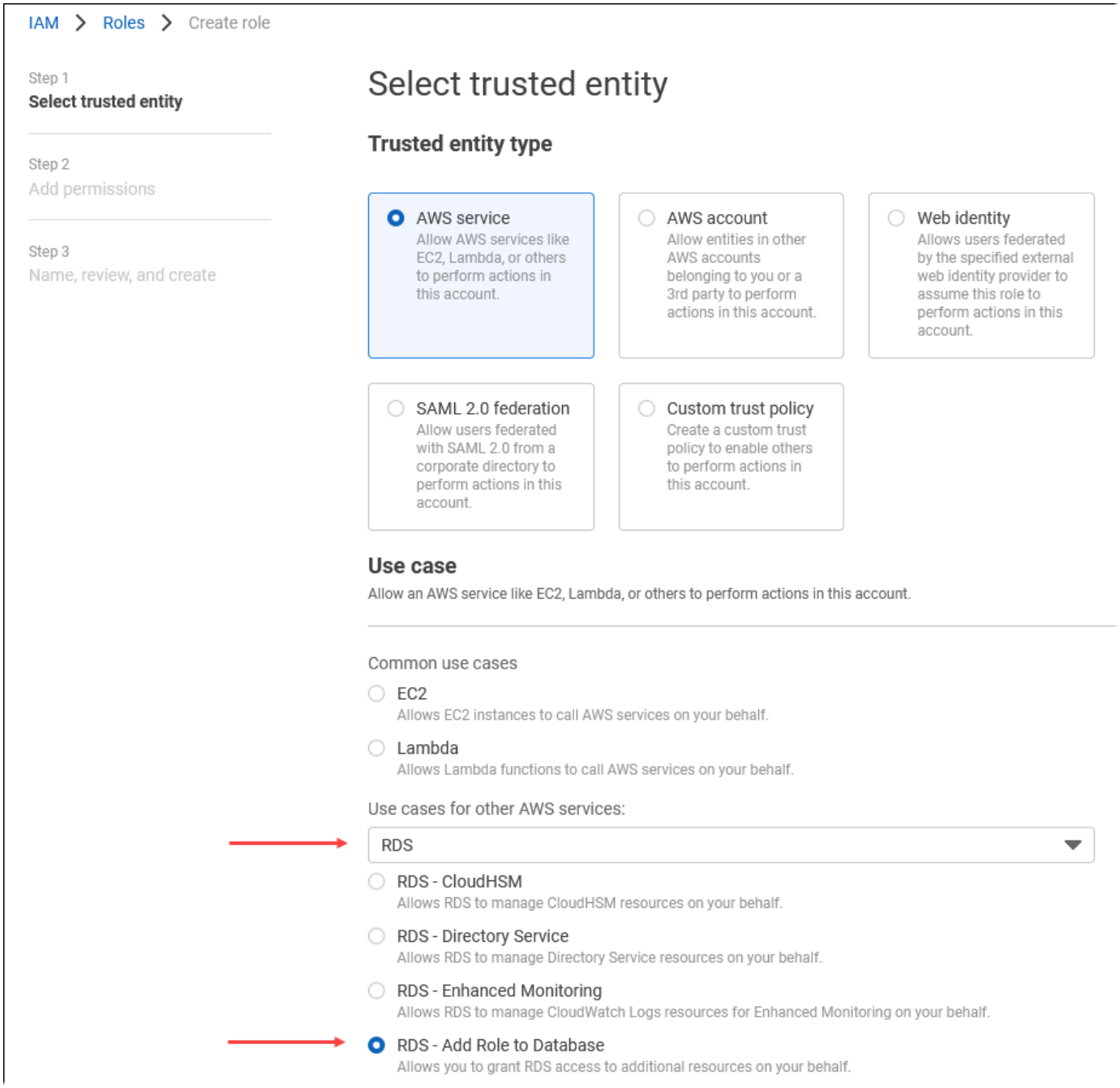
1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Buka konsol IAM di https://console.aws.amazon.com/iam/.
3. Pilih Kebijakan (di bawah Manajemen akses) pada menu AWS Identity and Access Management (IAM) Konsol.




4. Pilih Buat kebijakan. Di halaman Editor Visual, pilih Layanan, lalu masukkan Comprehend di kolom Pilih layanan. Perluas tingkat akses Baca. Pilih BatchDetectSentimentdan DetectSentimentdari pengaturan baca Amazon Comprehend.
5. Pilih Berikutnya: Tag dan tentukan tag apa pun (opsional). Pilih Berikutnya: Peninjauan. Masukkan nama untuk kebijakan dan deskripsi, seperti diperlihatkan pada gambar.



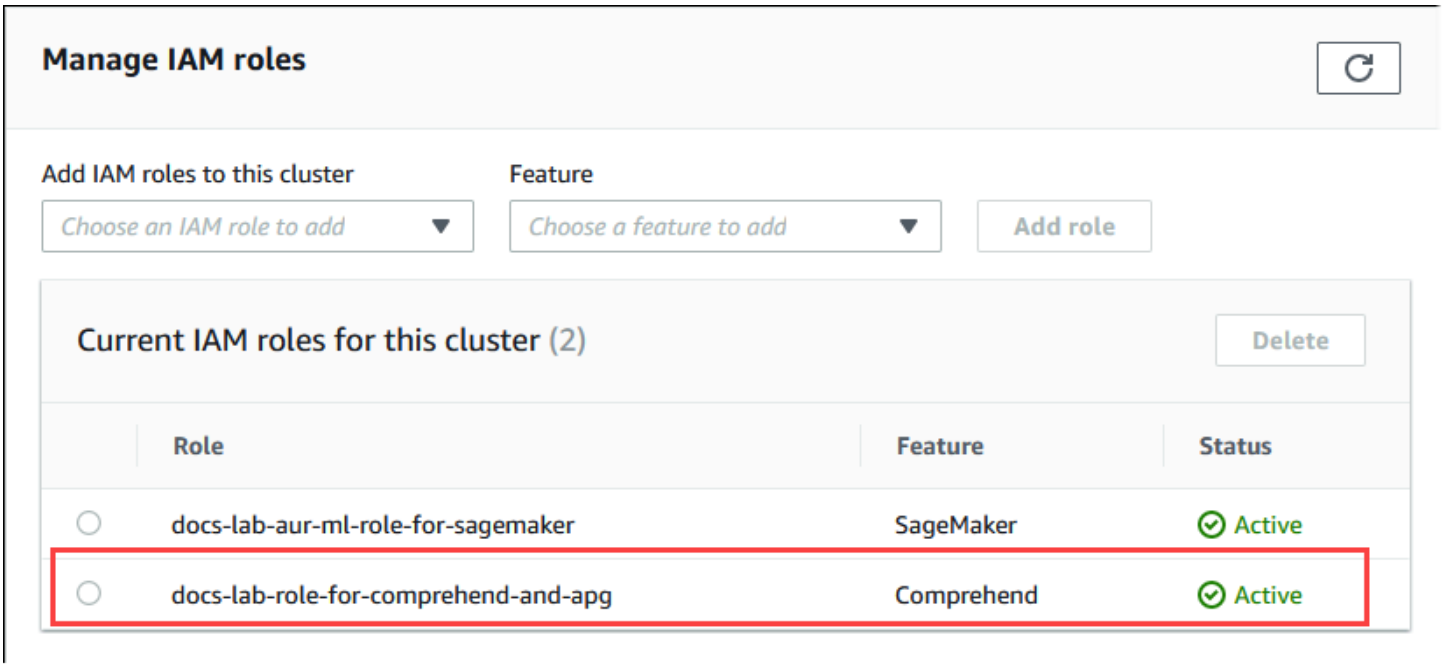

6. Pilih Buat kebijakan. Konsol menampilkan peringatan saat kebijakan telah disimpan. Anda dapat menemukannya di daftar Kebijakan.
7. Pilih Peran (di bawah Manajemen akses) di IAM Konsol.
8. Pilih Buat peran.
9. Pada halaman Pilih entitas tepercaya, pilih ubin AWS layanan, lalu pilih RDSuntuk membuka pemilih.
10. Pilih RDS— Tambahkan Peran ke Database.




11. Pilih Berikutnya. Di halaman Tambahkan izin, temukan kebijakan yang Anda buat di langkah sebelumnya dan pilih kebijakan dari daftar tersebut. Pilih Selanjutnya
12. Berikutnya: Tinjau. Masukkan nama untuk IAM peran dan deskripsi.
13. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
14. Arahkan ke Wilayah AWS tempat cluster Aurora Postgre SQL DB Anda berada.
15. Di panel navigasi, pilih Databases, lalu pilih cluster Aurora Postgre SQL DB yang ingin Anda gunakan dengan Amazon Comprehend.

16. Pilih tab Konektivitas & keamanan dan gulir untuk menemukan bagian Kelola IAM peran pada halaman. Dari Tambahkan IAM peran ke pemilih klaster ini, pilih peran yang Anda buat di langkah sebelumnya. Di pemilih Fitur, pilih Comprehend, lalu pilih Tambah peran.

Peran (dengan kebijakannya) dikaitkan dengan cluster Aurora SQL Postgre DB. Saat proses selesai, peran tersebut tercantum dalam Peran saat ini IAM untuk daftar klaster ini, seperti yang ditunjukkan berikut.




IAMPengaturan untuk Amazon Comprehend selesai. Lanjutkan menyiapkan Aurora Postgre Anda untuk SQL bekerja dengan pembelajaran mesin Aurora dengan menginstal ekstensi seperti yang dijelaskan dalam Menginstal ekstensi machine learning Aurora

## Menyiapkan Aurora Postgre untuk menggunakan Amazon AI SQL SageMaker

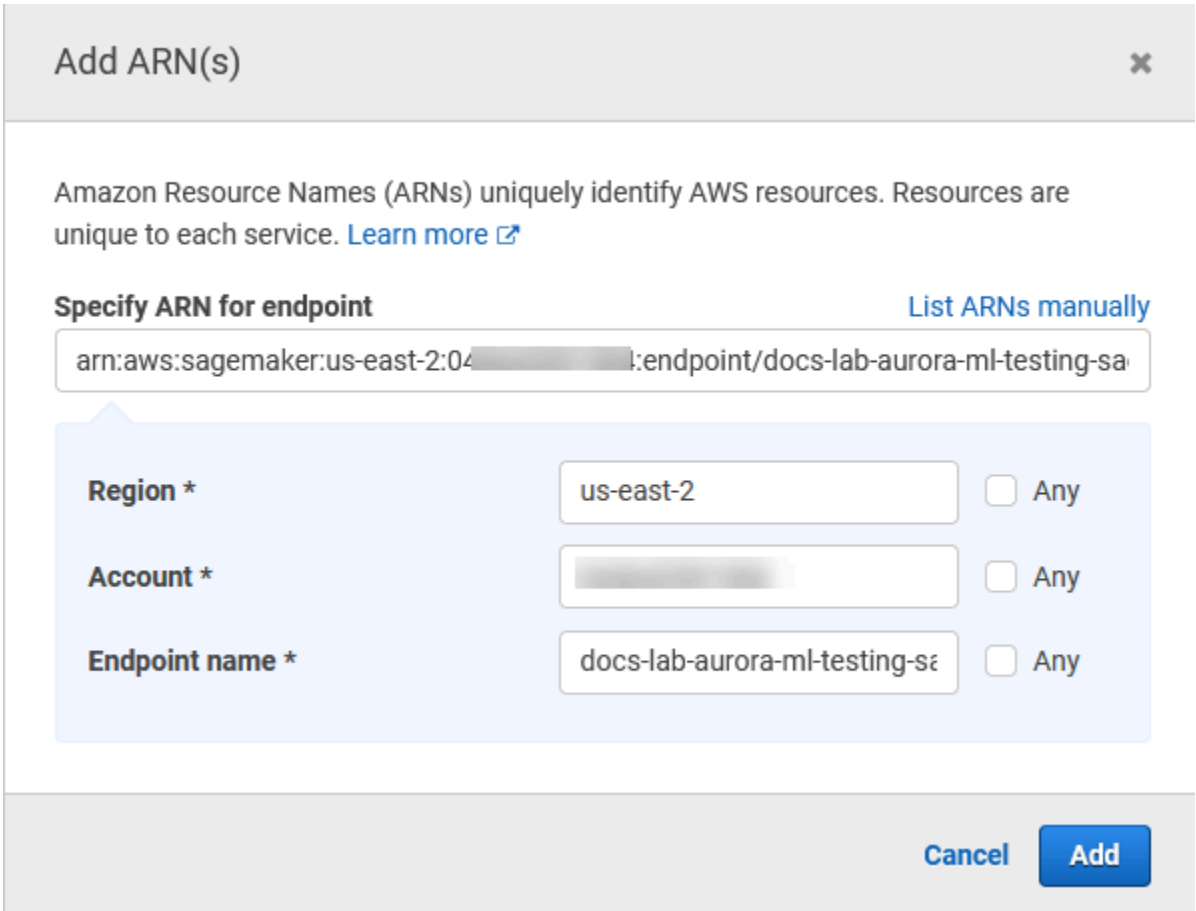
Sebelum Anda dapat membuat IAM kebijakan dan peran untuk cluster Aurora Postgre SQL DB Anda, Anda harus memiliki pengaturan model SageMaker AI dan titik akhir Anda tersedia.

Untuk mengatur cluster Aurora Postgre SQL DB Anda untuk menggunakan AI SageMaker

1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Pilih Kebijakan (di bawah Manajemen akses) di menu AWS Identity and Access Management (IAM) Konsol, lalu pilih Buat kebijakan. Di editor Visual, pilih SageMakeruntuk Layanan. Untuk

Tindakan, buka pemilih Baca (di bawah tingkat Akses) dan pilih InvokeEndpoint. Saat Anda melakukannya, ikon peringatan akan muncul.

3. Buka pemilih Sumber daya dan pilih tautan Tambahkan ARN untuk membatasi akses di bawah Tentukan sumber daya titik akhir ARN untuk tindakan. InvokeEndpoint
4. Masukkan Wilayah AWS sumber daya SageMaker AI Anda dan nama titik akhir Anda. AWS Akun Anda sudah diisi sebelumnya.




5. Pilih Tambah untuk menyimpan. Pilih Berikutnya: Tag dan Berikutnya: Tinjau untuk membuka halaman terakhir proses pembuatan kebijakan.
6. Masukkan Nama dan Deskripsi untuk kebijakan ini, lalu pilih Buat kebijakan. Kebijakan dibuat dan ditambahkan ke daftar Kebijakan. Anda melihat peringatan di Konsol saat ini terjadi.
7. Di IAM Konsol, pilih Peran.
8. Pilih Buat peran.
9. Pada halaman Pilih entitas tepercaya, pilih ubin AWS layanan, lalu pilih RDSuntuk membuka pemilih.
10. Pilih RDS— Tambahkan Peran ke Database.

11. Pilih Berikutnya. Di halaman Tambahkan izin, temukan kebijakan yang Anda buat di langkah sebelumnya dan pilih kebijakan dari daftar tersebut. Pilih Selanjutnya
12. Berikutnya: Tinjau. Masukkan nama untuk IAM peran dan deskripsi.
13. Buka RDS konsol Amazon di https://console.aws.amazon.com/rds/.
14. Arahkan ke Wilayah AWS tempat cluster Aurora Postgre SQL DB Anda berada.
15. Di panel navigasi, pilih Databases, lalu pilih cluster Aurora Postgre SQL DB yang ingin Anda gunakan dengan AI. SageMaker
16. Pilih tab Konektivitas & keamanan dan gulir untuk menemukan bagian Kelola IAM peran pada halaman. Dari Tambahkan IAM peran ke pemilih klaster ini, pilih peran yang Anda buat di langkah sebelumnya. Di pemilih Fitur, pilih SageMaker AI, lalu pilih Tambah peran.

Peran (dengan kebijakannya) dikaitkan dengan cluster Aurora SQL Postgre DB. Saat proses selesai, peran tersebut tercantum dalam Peran saat ini IAM untuk daftar klaster ini.

IAMPengaturan untuk SageMaker AI selesai. Lanjutkan menyiapkan Aurora Postgre Anda untuk SQL bekerja dengan pembelajaran mesin Aurora dengan menginstal ekstensi seperti yang dijelaskan dalam. Menginstal ekstensi machine learning Aurora

Menyiapkan Aurora Postgre untuk menggunakan SQL Amazon S3 untuk AI (Lanjutan) SageMaker

Untuk menggunakan SageMaker AI dengan model Anda sendiri daripada menggunakan komponen bawaan yang disediakan oleh SageMaker AI, Anda perlu menyiapkan bucket Amazon Simple Storage Service (Amazon S3) untuk digunakan cluster Aurora SQL Postgre DB. Ini adalah topik lanjutan, dan tidak sepenuhnya didokumentasikan dalam Panduan Pengguna Amazon Aurora ini. Proses umumnya sama dengan mengintegrasikan dukungan untuk SageMaker AI, sebagai berikut.

1. Buat IAM kebijakan dan peran untuk Amazon S3.
2. Tambahkan IAM peran dan impor atau ekspor Amazon S3 sebagai fitur pada tab Konektivitas & keamanan klaster Aurora Postgre DB Anda. SQL
3. Tambahkan peran ke grup parameter cluster DB kustom Anda untuk cluster Aurora DB Anda. ARN

Untuk informasi penggunaan dasar, lihat Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan).

## Menginstal ekstensi machine learning Aurora

Ekstensi pembelajaran mesin Aurora `aws_ml 1.0` menyediakan dua fungsi yang dapat Anda gunakan untuk memanggil Amazon Comprehend, layanan SageMaker AI, dan `aws_ml 2.0` menyediakan dua fungsi tambahan yang dapat Anda gunakan untuk menjalankan layanan Amazon Bedrock. Memasang ekstensi ini di cluster Aurora Postgre SQL DB Anda juga menciptakan peran administratif untuk fitur tersebut.

> **Note**
>
> Menggunakan fungsi-fungsi ini tergantung pada IAM penyiapan untuk layanan pembelajaran mesin Aurora (Amazon Comprehend SageMaker , AI, Amazon Bedrock) yang lengkap, seperti yang dijelaskan dalam. Menyiapkan cluster Aurora Postgre SQL DB Anda untuk menggunakan pembelajaran mesin Aurora

- aws_comprehend.detect_sentiment — Anda menggunakan fungsi ini untuk menerapkan analisis sentimen ke teks yang disimpan dalam database pada cluster Aurora Postgre DB Anda. SQL
- aws_sagemaker.invoke_endpoint — Anda menggunakan fungsi ini dalam kode Anda untuk berkomunikasi dengan titik akhir AI dari cluster Anda. SQL SageMaker
- aws_bedrock.invoke_model — Anda menggunakan fungsi ini dalam SQL kode Anda untuk berkomunikasi dengan Model Batuan Dasar dari cluster Anda. Respon fungsi ini akan dalam format aTEXT, jadi jika model merespons dalam format JSON tubuh maka output dari fungsi ini akan diteruskan dalam format string ke pengguna akhir.
- aws_bedrock.invoke_model_get_embeddings — Anda menggunakan fungsi ini dalam kode Anda untuk memanggil Model Bedrock yang mengembalikan embeddings keluaran dalam respons. SQL JSON Ini dapat dimanfaatkan saat Anda ingin mengekstrak penyematan yang terkait langsung dengan json-key untuk merampingkan respons dengan alur kerja yang dikelola sendiri.

Untuk menginstal ekstensi pembelajaran mesin Aurora di cluster DB Aurora Postgre Anda SQL

- Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora SQL Postgre DB Anda. Hubungkan ke basis data tertentu tempat untuk menginstal ekstensi `aws_ml`.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=labdb
```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
labdb=>
```

Menginstal `aws_ml` ekstensi juga menciptakan peran `aws_ml` administratif dan tiga skema baru, sebagai berikut.

- `aws_comprehend` – Skema untuk layanan Amazon Comprehend dan sumber fungsi `detect_sentiment` (`aws_comprehend.detect_sentiment`).
- `aws_sagemaker`— Skema untuk layanan SageMaker AI dan sumber `invoke_endpoint` fungsi (`aws_sagemaker.invoke_endpoint`).
- `aws_bedrock`— Skema untuk layanan Amazon Bedrock dan sumber `invoke_model(aws_bedrock.invoke_model)` dan `invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings)` fungsi.

Peran tersebut diberikan `rds_superuser` peran `aws_ml` administratif dan dibuat `OWNER` dari tiga skema pembelajaran mesin Aurora ini. Untuk mengizinkan pengguna basis data lain mengakses fungsi machine learning Aurora, `rds_superuser` harus memberikan hak istimewa `EXECUTE` pada fungsi machine learning Aurora. Secara default, hak istimewa `EXECUTE` dicabut dari `PUBLIC` pada fungsi dalam dua skema machine learning Aurora.

Dalam konfigurasi basis data multi-penghuni, Anda dapat mencegah penghuni mengakses fungsi machine learning Aurora dengan menggunakan `REVOKE USAGE` pada skema machine learning Aurora tertentu yang ingin dilindungi.

## Menggunakan Amazon Bedrock dengan cluster DB Aurora Postgre SQL Anda

Untuk Aurora PostgreSQL, pembelajaran mesin Aurora menyediakan fungsi Amazon Bedrock berikut untuk bekerja dengan data teks Anda. Fungsi ini hanya tersedia setelah Anda menginstal ekstensi `aws_ml` 2.0 dan menyelesaikan semua prosedur pengaturan. Untuk informasi selengkapnya, lihat Menyiapkan cluster Aurora Postgre SQL DB Anda untuk menggunakan pembelajaran mesin Aurora.

aws_bedrock.invoke_model

Fungsi ini mengambil teks yang diformat JSON sebagai input dan memprosesnya untuk berbagai model yang dihosting di Amazon Bedrock dan mendapatkan kembali respons JSON teks dari model. Respons ini bisa berisi teks, gambar, atau embeddings. Ringkasan dokumentasi fungsinya adalah sebagai berikut.

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `model_id`— Pengidentifikasi model.
- `content_type`— Jenis permintaan untuk model Bedrock.
- `accept_type`— Jenis respons yang diharapkan dari model Bedrock. Biasanya JSON aplikasi/ untuk sebagian besar model.
- `model_input`— Prompts; satu set input tertentu ke model dalam format seperti yang ditentukan oleh content_type. Untuk informasi lebih lanjut tentang format/struktur permintaan yang diterima model, lihat Parameter inferensi untuk model dasar.
- `model_output`— Output model Bedrock sebagai teks.

Contoh berikut menunjukkan cara memanggil model Anthropic Claude 2 untuk Bedrock menggunakan invoke_model.

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Bedrock

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
```

```
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

Output model dapat menunjuk ke embeddings vektor untuk beberapa kasus. Mengingat respons bervariasi per model, fungsi lain invoke_model_get_embeddings dapat dimanfaatkan yang berfungsi persis seperti invoke_model tetapi mengeluarkan penyematan dengan menentukan json-key yang sesuai.

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `model_id`— Pengidentifikasi model.
- `content_type`— Jenis permintaan untuk model Bedrock. Di sini, accept_type diatur ke nilai default. `application/json`
- `model_input`— Prompts; satu set input tertentu ke Model dalam format seperti yang ditentukan oleh content_type. Untuk informasi lebih lanjut tentang format/struktur permintaan yang diterima Model, lihat Parameter inferensi untuk model dasar.
- `json_key`— Referensi ke bidang untuk mengekstrak embedding dari. Ini dapat bervariasi jika model penyematan berubah.
- `model_output`— Output model Bedrock sebagai array embeddings yang memiliki desimal 16 bit.

Contoh berikut menunjukkan cara menghasilkan embedding menggunakan Titan Embeddings G1 — Model penyematan teks untuk frase Postgre I/O monitoring views. SQL

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Bedrock

```
SELECT aws_bedrock.invoke_model_get_embeddings(
   model_id      := 'amazon.titan-embed-text-v1',
```

```
    content_type  := 'application/json',
    json_key      := 'embedding',
    model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

## Menggunakan Amazon Comprehend dengan cluster DB Aurora Postgre Anda SQL

Untuk Aurora PostgreSQL, pembelajaran mesin Aurora menyediakan fungsi Amazon Comprehend berikut untuk bekerja dengan data teks Anda. Fungsi ini hanya tersedia setelah Anda menginstal ekstensi `aws_ml` dan menyelesaikan semua prosedur penyiapan. Untuk informasi selengkapnya, lihat Menyiapkan cluster Aurora Postgre SQL DB Anda untuk menggunakan pembelajaran mesin Aurora.

aws_comprehend.detect_sentiment

Fungsi ini mengambil teks sebagai input dan mengevaluasi apakah teks memiliki postur emosional positif, negatif, netral, atau campuran. Fungsi ini menghasilkan sentimen ini bersama dengan tingkat kepercayaan untuk evaluasinya. Ringkasan dokumentasi fungsinya adalah sebagai berikut.

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
   OUT confidence real)
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `input_text` – Teks untuk mengevaluasi dan menetapkan sentimen (negatif, positif, netral, campuran).
- `language_code`— Bahasa yang `input_text` diidentifikasi menggunakan pengenal 2-huruf ISO 639-1 dengan subtag regional (sesuai kebutuhan) atau kode tiga huruf ISO 639-2, yang sesuai. Misalnya, `en` adalah kode untuk bahasa Inggris, `zh` adalah kode untuk bahasa Mandarin yang disederhanakan. Untuk informasi selengkapnya, lihat Bahasa yang didukung di Panduan Developer Amazon Comprehend.
- `max_rows_per_batch` – Jumlah baris maksimum per batch untuk pemrosesan mode batch. Untuk informasi selengkapnya, lihat Memahami mode batch dan fungsi machine learning Aurora.

- `sentiment`— Sentimen teks input, diidentifikasi sebagaiPOSITIVE,, NEGATIVENEUTRAL, atauMIXED.
- `confidence` – Tingkat kepercayaan dalam keakuratan yang ditentukan `sentiment`. Rentang nilai dari 0,0 hingga 1,0.

Berikut ini, Anda dapat menemukan contoh cara menggunakan fungsi ini.

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Comprehend

Berikut adalah contoh kueri sederhana yang menginvokasi fungsi ini untuk menilai kepuasan pelanggan dengan tim dukungan Anda. Misalkan Anda memiliki tabel basis data (`support`) yang menyimpan umpan balik pelanggan setelah setiap permintaan bantuan. Contoh kueri ini menerapkan fungsi `aws_comprehend.detect_sentiment` ke teks di kolom `feedback` tabel dan menghasilkan sentimen dan tingkat kepercayaan untuk sentimen tersebut. Kueri ini juga menghasilkan urutan menurun.

```
SELECT feedback, s.sentiment,s.confidence
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                          | sentiment | confidence
 ----------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!             | POSITIVE  |   0.999771
 The latest version of this product stinks!                | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.       | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.   | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.           | POSITIVE  |   0.949491
 My problem was never resolved!                            | NEGATIVE  |   0.920644
 When will the new version of this product be released?    | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                              | NEGATIVE  |   0.895219
 Your support tech talked down to me.                      | NEGATIVE  |   0.868598
 It took me way too long to get a real person.             | NEGATIVE  |   0.481805

 (10 rows)
```

Agar tidak ditagih untuk deteksi sentimen lebih dari sekali per baris tabel, Anda dapat mewujudkan hasilnya. Lakukan ini di baris yang diinginkan. Misalnya, catatan dokter sedang diperbarui sehingga hanya yang berbahasa Prancis (`fr`) yang menggunakan fungsi deteksi sentimen.

```
UPDATE clinician_notes
```

```
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

Untuk informasi selengkapnya tentang cara mengoptimalkan panggilan fungsi Anda, lihat Pertimbangan kinerja untuk menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL.

## Menggunakan SageMaker AI dengan cluster Aurora SQL Postgre DB Anda

Setelah menyiapkan lingkungan SageMaker AI Anda dan berintegrasi dengan Aurora SQL Postgre seperti yang diuraikan, Anda dapat menjalankan Menyiapkan Aurora Postgre untuk menggunakan Amazon AI SQL SageMaker operasi dengan menggunakan fungsi tersebut. `aws_sagemaker.invoke_endpoint` Fungsi `aws_sagemaker.invoke_endpoint` hanya terhubung ke titik akhir model dalam Wilayah AWS yang sama. Jika instans database Anda memiliki replika dalam beberapa, Wilayah AWS pastikan Anda menyiapkan dan menerapkan setiap model SageMaker AI ke setiap model. Wilayah AWS

Panggilan ke `aws_sagemaker.invoke_endpoint` diautentikasi menggunakan IAM peran yang Anda atur untuk mengaitkan klaster Aurora SQL Postgre DB Anda dengan SageMaker layanan AI dan titik akhir yang Anda berikan selama proses penyiapan. SageMaker Titik akhir model AI dicakup ke akun individu dan tidak bersifat publik. `endpoint_name`URLTidak berisi ID akun. SageMaker AI menentukan ID akun dari token otentikasi yang disediakan oleh IAM peran SageMaker AI dari instance database.

aws_sagemaker.invoke_endpoint

Fungsi ini mengambil titik akhir SageMaker AI sebagai input dan jumlah baris yang harus diproses sebagai batch. Ini juga membutuhkan input berbagai parameter yang diharapkan oleh titik akhir model SageMaker AI. Dokumentasi referensi fungsi ini adalah sebagai berikut.

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
  IN max_rows_per_batch int,
  VARIADIC model_input "any",
  OUT model_output varchar
  )
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `endpoint_name`— Titik akhir URL yang Wilayah AWS—independen.
- `max_rows_per_batch` – Jumlah baris maksimum per batch untuk pemrosesan mode batch. Untuk informasi selengkapnya, lihat Memahami mode batch dan fungsi machine learning Aurora.
- `model_input` – Satu atau beberapa parameter input untuk model. Ini bisa berupa tipe data apa pun yang dibutuhkan oleh model SageMaker AI. Postgre SQL memungkinkan Anda menentukan hingga 100 parameter input untuk suatu fungsi. Tipe data array harus satu dimensi, tetapi dapat berisi elemen sebanyak yang diharapkan oleh model SageMaker AI. Jumlah input ke model SageMaker AI hanya dibatasi oleh batas ukuran pesan SageMaker AI 6 MB.
- `model_output`— Output model SageMaker AI sebagai teks.

## Membuat fungsi yang ditentukan pengguna untuk memanggil model AI SageMaker

Buat fungsi terpisah yang ditentukan pengguna `aws_sagemaker.invoke_endpoint` untuk memanggil setiap model SageMaker AI Anda. Fungsi yang ditentukan pengguna Anda mewakili titik akhir SageMaker AI yang menghosting model. Fungsi `aws_sagemaker.invoke_endpoint` berjalan dalam fungsi yang ditentukan pengguna. Fungsi yang ditentukan pengguna memberikan banyak keuntungan:

- Anda dapat memberikan nama model SageMaker AI Anda sendiri alih-alih hanya memanggil `aws_sagemaker.invoke_endpoint` semua model SageMaker AI Anda.
- Anda dapat menentukan titik akhir model hanya URL di satu tempat dalam kode SQL aplikasi Anda.
- Anda dapat mengontrol hak istimewa `EXECUTE` untuk setiap fungsi machine learning Aurora secara mandiri.
- Anda dapat mendeklarasikan jenis input dan output model menggunakan SQL tipe. SQLmemberlakukan jumlah dan jenis argumen yang diteruskan ke model SageMaker AI Anda dan melakukan konversi tipe jika perlu. Menggunakan SQL tipe juga akan diterjemahkan `SQL NULL` ke nilai default yang sesuai yang diharapkan oleh model SageMaker AI Anda.
- Anda dapat mengurangi ukuran batch maksimum jika Anda ingin menampilkan beberapa baris pertama sedikit lebih cepat.

Untuk menentukan fungsi yang ditentukan pengguna, gunakan pernyataan bahasa definisi SQL data (DDL). `CREATE FUNCTION` Saat menentukan fungsi, Anda juga menentukan:

- Parameter input ke model.
- Titik akhir SageMaker AI spesifik untuk dipanggil.
- Jenis yang ditampilkan.

Fungsi yang ditentukan pengguna mengembalikan inferensi yang dihitung oleh titik akhir SageMaker AI setelah menjalankan model pada parameter input. Contoh berikut membuat fungsi yang ditentukan pengguna untuk model SageMaker AI dengan dua parameter input.

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2                          -- model inputs are separate arguments
        )::INT                              -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Perhatikan hal berikut:

- Input fungsi `aws_sagemaker.invoke_endpoint` dapat berupa satu parameter atau lebih dari jenis data apa pun.
- Contoh ini menggunakan tipe INT output. Jika Anda mentransmisikan output dari `varchar` tipe ke tipe yang berbeda, maka itu harus dilemparkan ke tipe skalar SQL bawaan Postgre sepertiINTEGER,,, REAL atau. FLOAT NUMERIC Untuk informasi selengkapnya tentang jenis ini, lihat Tipe data dalam dokumentasi PostgreSQL.
- Tentukan `PARALLEL SAFE` untuk mengaktifkan pemrosesan kueri paralel. Untuk informasi selengkapnya, lihat Meningkatkan waktu respons dengan pemrosesan kueri paralel.
- Tentukan `COST 5000` untuk mengestimasi biaya menjalankan fungsi. Gunakan bilangan positif yang memberikan estimasi biaya eksekusi untuk fungsi tersebut, dalam unit `cpu_operator_cost`.

## Melewati array sebagai input ke model SageMaker AI

`aws_sagemaker.invoke_endpoint`Fungsi ini dapat memiliki hingga 100 parameter input, yang merupakan batas untuk fungsi PostgreSQL. Jika model SageMaker AI membutuhkan lebih dari 100 parameter dengan tipe yang sama, berikan parameter model sebagai array.

Contoh berikut mendefinisikan fungsi yang meneruskan array sebagai input ke model regresi SageMaker AI. Output dikirim ke nilai REAL.

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

## Menentukan ukuran batch saat menjalankan model AI SageMaker

Contoh berikut membuat fungsi yang ditentukan pengguna untuk model SageMaker AI yang menetapkan ukuran batch default. NULL Fungsi ini juga memungkinkan Anda untuk memberikan ukuran batch yang berbeda saat Anda menginvokasinya.

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Perhatikan hal berikut:

- Gunakan parameter `max_rows_per_batch` opsional untuk memberikan kontrol jumlah baris untuk invokasi fungsi mode batch. Jika Anda menggunakan nilaiNULL, maka pengoptimal kueri secara otomatis memilih ukuran batch maksimum. Untuk informasi selengkapnya, lihat Memahami mode batch dan fungsi machine learning Aurora.
- Secara default, meneruskan NULL sebagai nilai parameter diterjemahkan ke string kosong sebelum diteruskan ke SageMaker AI. Untuk contoh ini, input memiliki jenis-jenis yang berbeda.
- Jika Anda memiliki masukan non-teks, atau input teks yang perlu diatur ke default ke beberapa nilai selain string kosong, gunakan pernyataan `COALESCE`. Gunakan `COALESCE` untuk menerjemahkan NULL ke nilai penggantian null yang diinginkan dalam panggilan

keaws_sagemaker.invoke_endpoint. Untuk amount parameter dalam contoh ini, NULL nilai dikonversi menjadi 0,0.

### Memohon model SageMaker AI yang memiliki banyak output

Contoh berikut membuat fungsi yang ditentukan pengguna untuk model SageMaker AI yang mengembalikan beberapa output. Fungsi Anda perlu mentransmisikan output dari fungsi aws_sagemaker.invoke_endpoint ke jenis data yang terkait. Misalnya, Anda dapat menggunakan tipe SQL titik Postgre bawaan untuk pasangan (x, y) atau tipe komposit yang ditentukan pengguna.

Fungsi yang ditentukan oleh pengguna ini menampilkan nilai dari model yang menampilkan beberapa output menggunakan jenis gabungan untuk output.

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Untuk jenis komposit, gunakan kolom dalam urutan yang sama seperti yang muncul di output model dan transmisikan keluaran aws_sagemaker.invoke_endpoint ke jenis komposit Anda. Penelepon dapat mengekstrak bidang individual baik dengan nama atau dengan notasi Postgre SQL “.*”.

## Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model AI (Lanjutan)

Kami menyarankan agar Anda terbiasa dengan pembelajaran mesin Aurora dan SageMaker AI dengan menggunakan algoritme dan contoh yang disediakan daripada mencoba melatih model Anda sendiri. Untuk informasi selengkapnya, lihat Memulai dengan Amazon SageMaker AI

Untuk melatih model SageMaker AI, Anda mengekspor data ke bucket Amazon S3. Bucket Amazon S3 digunakan oleh SageMaker AI untuk melatih model Anda sebelum digunakan. Anda dapat melakukan kueri data dari cluster SQL DB Aurora Postgre dan menyimpannya langsung ke file teks yang disimpan di bucket Amazon S3. Kemudian SageMaker AI mengkonsumsi data dari bucket Amazon S3 untuk pelatihan. Untuk informasi lebih lanjut tentang pelatihan model SageMaker AI, lihat Melatih model dengan Amazon SageMaker AI.

> **Note**
>
> Saat Anda membuat bucket Amazon S3 untuk pelatihan model SageMaker AI atau penilaian batch, gunakan `sagemaker` nama bucket Amazon S3. Untuk informasi selengkapnya, lihat Menentukan Bucket Amazon S3 untuk Mengunggah Kumpulan Data Pelatihan dan Menyimpan Data Output di Panduan Pengembang Amazon SageMaker AI.

Untuk informasi lebih lanjut tentang mengekspor data Anda, lihat Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.

## Pertimbangan kinerja untuk menggunakan pembelajaran mesin Aurora dengan Aurora Postgre SQL

Layanan Amazon SageMaker Comprehend dan AI melakukan sebagian besar pekerjaan ketika dipanggil oleh fungsi pembelajaran mesin Aurora. Itu berarti Anda dapat menskalakan sumber daya tersebut sesuai kebutuhan, secara mandiri. Untuk cluster Aurora Postgre SQL DB Anda, Anda dapat membuat panggilan fungsi Anda seefisien mungkin. Berikut ini, Anda dapat menemukan beberapa pertimbangan kinerja yang perlu diperhatikan saat bekerja dengan pembelajaran mesin Aurora dari Aurora Postgre. SQL

Topik

- Memahami mode batch dan fungsi machine learning Aurora
- Meningkatkan waktu respons dengan pemrosesan kueri paralel
- Menggunakan tampilan terwujud dan kolom terwujud

### Memahami mode batch dan fungsi machine learning Aurora

Biasanya, Postgre SQL menjalankan fungsi satu baris pada satu waktu. Machine learning Aurora dapat mengurangi overhead ini dengan menggabungkan panggilan ke layanan machine learning

Aurora eksternal untuk banyak baris menjadi batch dengan pendekatan yang disebut eksekusi mode-batch. Dalam mode batch, machine learning Aurora menerima respons untuk batch baris input, lalu mengirimkan respons kembali ke kueri yang berjalan satu baris dalam satu waktu. Pengoptimalan ini meningkatkan throughput kueri Aurora Anda tanpa membatasi pengoptimal kueri Postgre. SQL

Aurora secara otomatis menggunakan mode batch jika fungsi direferensikan dari daftar `SELECT`, klausul `WHERE`, atau klausul `HAVING`. Perhatikan bahwa ekspresi `CASE` sederhana tingkat atas memenuhi syarat untuk eksekusi mode batch. Ekspresi `CASE` tingkat atas yang ditelusuri juga memenuhi syarat untuk eksekusi mode-batch asalkan klausul `WHEN` pertama adalah predikat sederhana dengan panggilan fungsi mode-batch.

Fungsi yang Anda tentukan harus berupa fungsi `LANGUAGE SQL` dan harus mencantumkan `PARALLEL SAFE` dan `COST 5000`.

Fungsi migrasi dari SELECT pernyataan ke FROM klausa

Biasanya, fungsi `aws_ml` yang memenuhi syarat untuk eksekusi mode-batch secara otomatis dimigrasikan oleh Aurora ke klausul `FROM`.

Migrasi fungsi mode batch yang memenuhi syarat ke FROM klausa dapat diperiksa secara manual pada tingkat per kueri. Untuk melakukan ini, Anda menggunakan EXPLAIN pernyataan (dan danVERBOSE) ANALYZE dan menemukan informasi “Pemrosesan Batch” di bawah setiap mode batch`Function Scan`. Anda juga dapat menggunakan EXPLAIN (withVERBOSE) tanpa menjalankan kueri. Selanjutnya, amati apakah panggilan ke fungsi tersebut muncul sebagai `Function Scan` di dalam loop join bertingkat yang tidak ditentukan dalam pernyataan asli.

Dalam contoh berikut, operator loop join bertingkat dalam rencana menunjukkan bahwa Aurora memigrasi fungsi `anomaly_score`. Ini memigrasikan fungsi ini dari SELECT daftar ke FROM klausa, yang memenuhi syarat untuk eksekusi mode batch.

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                         QUERY PLAN
-------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
         Output: anomaly_score.anomaly_score
```

```
          Function Call: anomaly_score((r.description)::text)
```

Untuk menonaktifkan eksekusi mode-batch, atur parameter `apg_enable_function_migration` ke `false`. Ini mencegah migrasi `aws_ml` fungsi dari klausa SELECT ke `FROM` klausa. Berikut caranya.

```
SET apg_enable_function_migration = false;
```

`apg_enable_function_migration`Parameter adalah parameter Grand Unified Configuration (GUC) yang dikenali oleh ekstensi Aurora SQL `apg_plan_mgmt` Postgre untuk manajemen rencana kueri. Untuk menonaktifkan migrasi fungsi dalam sesi, gunakan manajemen rencana kueri untuk menyimpan rencana yang dihasilkan sebagai rencana `approved`. Saat runtime, manajemen rencana kueri menerapkan rencana `approved` dengan pengaturan `apg_enable_function_migration`-nya. Penegakan ini terjadi terlepas dari pengaturan `apg_enable_function_migration` GUC parameter. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL.

Menggunakan parameter max_rows_per_batch

Baik fungsi `aws_comprehend.detect_sentiment` maupun fungsi `aws_sagemaker.invoke_endpoint` memiliki parameter `max_rows_per_batch`. Parameter ini menentukan jumlah baris yang dapat dikirim ke layanan machine learning Aurora. Semakin besar set data yang diproses oleh fungsi Anda, semakin besar ukuran batch yang bisa Anda buat.

Fungsi mode batch meningkatkan efisiensi dengan membangun batch baris yang menyebarkan biaya panggilan fungsi machine learning Aurora melalui banyak baris. Namun, jika pernyataan `SELECT` selesai lebih awal karena klausul `LIMIT`, maka konsep batch dapat dibuat di lebih banyak baris daripada yang digunakan kueri. Pendekatan ini dapat mengakibatkan biaya tambahan ke AWS akun Anda. Untuk memanfaatkan eksekusi mode batch, tetapi menghindari pembuatan batch yang terlalu besar, gunakan nilai yang lebih kecil untuk parameter `max_rows_per_batch` dalam panggilan fungsi Anda.

Jika Anda melakukan `EXPLAIN` (`VERBOSE`, `ANALYZE`) kueri yang menggunakan eksekusi mode batch, Anda akan melihat operator `FunctionScan` yang berada di bawah loop join bertingkat. Jumlah loop yang dilaporkan `EXPLAIN` sama dengan frekuensi sebuah baris diambil dari operator `FunctionScan`. Jika pernyataan menggunakan LIMIT klausa, jumlah pengambilan konsisten. Untuk mengoptimalkan ukuran batch, atur parameter `max_rows_per_batch` ke nilai ini. Namun, jika fungsi mode batch direferensikan dalam predikat dalam klausul `WHERE` atau klausul `HAVING`, Anda

mungkin tidak dapat mengetahui jumlah pengambilan sebelumnya. Dalam kasus ini, gunakan loop sebagai pedoman dan percobaan dengan `max_rows_per_batch` untuk menemukan pengaturan yang mengoptimalkan performa.

Memverifikasi eksekusi mode batch

Untuk melihat apakah fungsi berjalan dalam mode batch, gunakan `EXPLAIN ANALYZE`. Jika eksekusi mode batch digunakan, rencana kueri akan menyertakan informasi di bagian "Pemrosesan Batch".

```
EXPLAIN ANALYZE SELECT user-defined-function();
 Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                 avg/min/max batch call time=146.273/146.273/146.273
```

Dalam contoh ini, ada 1 batch yang berisi 3.333 baris, yang memerlukan waktu proses 146,273 md. Bagian "Pemrosesan Batch" menunjukkan berikut:

- Banyaknya batch yang tersedia untuk operasi pemindaian fungsi ini
- Ukuran batch rata-rata, minimum, dan maksimum
- Rata-rata waktu eksekusi batch, minimum, dan maksimum

Biasanya batch akhir lebih kecil daripada batch lain, yang sering kali menghasilkan ukuran batch minimum yang jauh lebih kecil dari rata-rata.

Untuk menampilkan beberapa baris pertama dengan lebih cepat, atur parameter `max_rows_per_batch` ke nilai yang lebih kecil.

Untuk mengurangi jumlah panggilan mode batch ke layanan ML saat Anda menggunakan `LIMIT` dalam fungsi yang ditentukan pengguna, atur parameter `max_rows_per_batch` ke nilai yang lebih kecil.

## Meningkatkan waktu respons dengan pemrosesan kueri paralel

Untuk mendapatkan hasil secepat mungkin dari banyak baris, Anda dapat menggabungkan pemrosesan kueri paralel dengan pemrosesan mode batch. Anda dapat menggunakan pemrosesan kueri paralel untuk pernyataan `SELECT`, `CREATE TABLE AS SELECT`, dan `CREATE MATERIALIZED VIEW`.

> **Note**
>
> Postgre SQL belum mendukung query paralel untuk pernyataan bahasa manipulasi data (DML).

Pemrosesan kueri paralel terjadi baik dalam basis data maupun dalam layanan ML. Jumlah inti dalam kelas instans basis data membatasi tingkat paralelisme yang dapat digunakan saat menjalankan kueri. Server basis data dapat membuat konsep rencana eksekusi kueri paralel yang membagi tugas di antara sekumpulan pekerja paralel. Selanjutnya, setiap pekerja ini bisa membangun permintaan batch yang berisi puluhan ribu baris (atau sebanyak yang diizinkan oleh setiap layanan).

Permintaan batch dari semua pekerja paralel dikirim ke titik akhir SageMaker AI. Tingkat paralelisme yang dapat didukung titik akhir dibatasi oleh jumlah dan jenis instans yang mendukungnya. Untuk K tingkat paralelisme, Anda memerlukan kelas instans basis data yang memiliki setidaknya K inti. Anda juga perlu mengonfigurasi titik akhir SageMaker AI agar model Anda memiliki K instance awal dari kelas instans berkinerja cukup tinggi.

Untuk menggunakan pemrosesan kueri paralel, Anda dapat mengatur parameter `parallel_workers` penyimpanan dari tabel yang berisi data yang ingin Anda teruskan. Anda mengatur `parallel_workers` ke fungsi mode-batch seperti `aws_comprehend.detect_sentiment`. Jika pengoptimal memilih paket query paralel, layanan AWS ML dapat dipanggil baik dalam batch maupun paralel.

Anda dapat menggunakan parameter berikut dengan fungsi `aws_comprehend.detect_sentiment` untuk mendapatkan rencana dengan paralelisme empat arah. Jika Anda mengubah salah satu dari dua parameter berikut, Anda harus memulai ulang instans basis data untuk menerapkan perubahan.

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);
```

```
-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

Untuk informasi selengkapnya tentang mengendalikan kueri paralel, lihat Rencana paralel dalam dokumentasi PostgreSQL.

## Menggunakan tampilan terwujud dan kolom terwujud

Ketika Anda memanggil AWS layanan seperti SageMaker AI atau Amazon Comprehend dari database Anda, akun Anda dibebankan sesuai dengan kebijakan harga layanan tersebut. Untuk meminimalkan biaya ke akun Anda, Anda dapat mewujudkan hasil pemanggilan AWS layanan ke kolom yang terwujud sehingga AWS layanan tidak dipanggil lebih dari sekali per baris input. Jika diinginkan, Anda dapat menambahkan kolom stempel waktu `materializedAt` untuk mencatat waktu saat kolom tersebut dibuat.

Latensi pernyataan `INSERT` baris tunggal biasa biasanya jauh lebih kecil daripada latensi pemanggilan fungsi mode batch. Dengan demikian, Anda mungkin tidak dapat memenuhi persyaratan latensi aplikasi Anda jika menginvokasi fungsi mode batch untuk setiap baris tunggal `INSERT` yang dijalankan oleh aplikasi Anda. Untuk mewujudkan hasil pemanggilan AWS layanan ke dalam kolom yang terwujud, aplikasi berkinerja tinggi umumnya perlu mengisi kolom yang terwujud. Untuk melakukannya, aplikasi tersebut secara berkala mengeluarkan pernyataan `UPDATE` yang beroperasi pada batch baris yang besar secara bersamaan.

`UPDATE` menggunakan penguncian level baris yang dapat memengaruhi aplikasi yang sedang berjalan. Jadi, Anda mungkin perlu menggunakan `SELECT ... FOR UPDATE SKIP LOCKED`, atau menggunakan `MATERIALIZED VIEW`.

Kueri analitik yang beroperasi pada banyak baris secara real-time dapat menggabungkan perwujudan mode batch dengan pemrosesan real-time. Untuk melakukan ini, kueri ini mengumpulkan `UNION ALL` dari hasil yang telah diwujudkan sebelumnya dengan kueri di atas baris yang belum memiliki hasil yang terwujud. Dalam beberapa kasus, `UNION ALL` seperti itu diperlukan di banyak tempat, atau kueri dibuat oleh aplikasi pihak ketiga. Jika demikian, Anda dapat membuat a `VIEW` untuk merangkum `UNION ALL` operasi sehingga detail ini tidak terpapar ke aplikasi lainnya. SQL

Anda bisa menggunakan tampilan terwujud untuk mewujudkan hasil dari pernyataan `SELECT` arbitrer pada snapshot tepat waktu. Anda juga dapat menggunakannya untuk menyegarkan

tampilan terwujud kapan saja di masa mendatang. Saat ini Postgre SQL tidak mendukung penyegaran tambahan, jadi setiap kali tampilan terwujud disegarkan, tampilan terwujud dihitung ulang sepenuhnya.

Anda dapat me-refresh tampilan terwujud dengan opsi `CONCURRENTLY`, yang memperbarui konten tampilan terwujud tanpa mengambil kunci eksklusif. Melakukan hal ini memungkinkan SQL aplikasi untuk membaca dari tampilan terwujud saat sedang disegarkan.

## Memantau machine learning Aurora

Anda dapat memantau fungsi `aws_ml` dengan mengatur parameter `track_functions` di grup parameter klaster DB kustom Anda ke `all`. Secara default, parameter ini diatur ke `pl` yang berarti bahwa hanya fungsi bahasa prosedur yang akan dilacak. Dengan mengubah ini menjadi `all`, fungsi `aws_ml` juga akan dilacak. Untuk informasi selengkapnya, lihat Statistik Run-time dalam dokumentasi SQL Postgre.

Untuk informasi tentang pemantauan kinerja operasi SageMaker AI yang disebut dari fungsi pembelajaran mesin Aurora, lihat Memantau Amazon SageMaker AI di Panduan Pengembang Amazon SageMaker AI.

Dengan `track_functions` diatur ke `all`, Anda dapat mengueri tampilan `pg_stat_user_functions` untuk mendapatkan statistik tentang fungsi yang Anda tentukan dan gunakan untuk menginvokasi layanan machine learning Aurora. Untuk setiap fungsi, tampilan ini memberikan jumlah `calls`, `total_time`, dan `self_time`.

Untuk melihat statistik untuk fungsi `aws_sagemaker.invoke_endpoint` dan `aws_comprehend.detect_sentiment`, Anda dapat memfilter hasil berdasarkan nama skema menggunakan kueri berikut.

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
    LIKE 'aws_%';
```

Untuk menghapus statistik, ikuti langkah berikut.

```
SELECT pg_stat_reset();
```

Anda bisa mendapatkan nama SQL fungsi Anda yang memanggil `aws_sagemaker.invoke_endpoint` fungsi dengan menanyakan katalog sistem Postgre

SQLpg_proc. Katalog ini menyimpan informasi tentang fungsi, prosedur, dan banyak lagi. Untuk informasi selengkapnya, lihat pg_proc di dokumentasi Postgre. SQL Berikut ini adalah contoh mengueri tabel untuk mendapatkan nama-nama fungsi (proname) yang sumbernya (prosrc) mencakup teks invoke_endpoint.

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

# Contoh kode untuk Aurora menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan Aurora dengan kit pengembangan AWS perangkat lunak (SDK).

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memulai

## Halo Aurora

Contoh kode berikut ini menunjukkan cara mulai menggunakan Aurora.

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk CMake file CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Kode untuk file sumber hello_aurora.cpp.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
```

```
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }
```

```
    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK for Java 2.x API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import boto3

# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
)

# Iterate through the pages of the response
clusters_found = False
```

```
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- Untuk detail API, lihat Menjelaskan DBClusters dalam AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)

# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
```

```
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
```

```
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di AWS SDK untuk referensi API Rust.

Contoh kode

- Contoh dasar untuk Aurora menggunakan AWS SDKs
  - Halo Aurora
  - Pelajari dasar-dasar Aurora dengan SDK AWS
  - Tindakan untuk Aurora menggunakan AWS SDKs
    - Gunakan CreateDBCluster dengan AWS SDK
    - Gunakan CreateDBClusterParameterGroup dengan AWS SDK
    - Gunakan CreateDBClusterSnapshot dengan AWS SDK
    - Gunakan CreateDBInstance dengan AWS SDK
    - Gunakan DeleteDBCluster dengan AWS SDK
    - Gunakan DeleteDBClusterParameterGroup dengan AWS SDK
    - Gunakan DeleteDBInstance dengan AWS SDK
    - Gunakan DescribeDBClusterParameterGroups dengan AWS SDK
    - Gunakan DescribeDBClusterParameters dengan AWS SDK
    - Gunakan DescribeDBClusterSnapshots dengan AWS SDK

    - Gunakan DescribeDBClusters dengan AWS SDK
    - Gunakan DescribeDBEngineVersions dengan AWS SDK
    - Gunakan DescribeDBInstances dengan AWS SDK
    - Gunakan DescribeOrderableDBInstanceOptions dengan AWS SDK atau CLI
    - Gunakan ModifyDBClusterParameterGroup dengan AWS SDK
- Skenario untuk Aurora menggunakan AWS SDKs
    - Membuat API REST pustaka peminjaman
    - Buat pelacak butir kerja Aurora Nirserver


# Contoh dasar untuk Aurora menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan dasar-dasar Amazon Aurora dengan. AWS SDKs

Contoh


- Halo Aurora
- Pelajari dasar-dasar Aurora dengan SDK AWS
- Tindakan untuk Aurora menggunakan AWS SDKs
    - Gunakan CreateDBCluster dengan AWS SDK
    - Gunakan CreateDBClusterParameterGroup dengan AWS SDK
    - Gunakan CreateDBClusterSnapshot dengan AWS SDK
    - Gunakan CreateDBInstance dengan AWS SDK
    - Gunakan DeleteDBCluster dengan AWS SDK
    - Gunakan DeleteDBClusterParameterGroup dengan AWS SDK
    - Gunakan DeleteDBInstance dengan AWS SDK
    - Gunakan DescribeDBClusterParameterGroups dengan AWS SDK
    - Gunakan DescribeDBClusterParameters dengan AWS SDK
    - Gunakan DescribeDBClusterSnapshots dengan AWS SDK
    - Gunakan DescribeDBClusters dengan AWS SDK
    - Gunakan DescribeDBEngineVersions dengan AWS SDK
    - Gunakan DescribeDBInstances dengan AWS SDK

- Gunakan DescribeOrderableDBInstanceOptions dengan AWS SDK atau CLI
- Gunakan ModifyDBClusterParameterGroup dengan AWS SDK

## Halo Aurora

Contoh kode berikut ini menunjukkan cara mulai menggunakan Aurora.

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
```

```
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk CMake file CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
```

```
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Kode untuk file sumber hello_aurora.cpp.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
```

```
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)
```

```
// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }

    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK for Java 2.x API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import boto3

# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
)

# Iterate through the pages of the response
clusters_found = False
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- Untuk detail API, lihat Menjelaskan DBClusters dalam AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)

# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
```

```
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Pelajari dasar-dasar Aurora dengan SDK AWS

Contoh kode berikut ini menunjukkan cara:

- Membuat grup parameter klaster DB Aurora dan mengatur nilai parameter.
- Membuat klaster DB yang menggunakan grup parameter.
- Membuat instans DB yang berisi basis data.
- Mengambil snapshot klaster DB, lalu membersihkan sumber daya.

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
using Amazon.RDS;
using Amazon.RDS.Model;
```

```
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group
 using the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the
 DescribeDBClusterParametersAsync method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
 DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
 parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
 method.
    11. Display and select from a list of instance classes available for the
 selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
    12. Create a database instance in the cluster using the CreateDBInstanceAsync
 method.
```

```
   13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
   16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

       Console.WriteLine(sepBar);
```

```
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;
```

```
            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
```

```
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family
 by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
 new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
```

```
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
 parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
```

```
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
 List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
 auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
 parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
 group.");

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
 "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }
```

```
    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
 choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
 dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
 dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
 family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
```

```
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
```

```
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
```

```
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
```

```
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {
            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
 auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");
```

```
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
{cluster.DBClusterIdentifier}.");
        var snapshot = await
auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
```

```
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }
```

```
            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
 group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
 {parameterGroup.DBClusterParameterGroupName}.");
            await
 auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Metode pembungkus yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
```

```
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
```

```
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
_amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
```

```
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
```

```
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
```

```
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
```

```
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
```

```
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
```

```
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });
```

```
        return response.DBInstance;
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter
  - Jelaskan DBCluster Snapshots
  - Jelaskan DBClusters
  - Jelaskan DBEngine Versi
  - Jelaskan DBInstances
  - DescribeOrderableDBInstancePilihan
  - Memodifikasi DBCluster ParameterGroup

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();

    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
```

```
                    CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }

        std::cout << "Getting available parameter group families for " <<
DB_ENGINE
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
```

```
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "Let's set some parameter values in your cluster parameter
group."
              << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
    // 4.  Get the parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                                NO_SOURCE,
                                autoIncrementParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

    for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
        if (autoIncParameter.GetIsModifiable() &&
            (autoIncParameter.GetDataType() == "integer")) {
            std::cout << "The " << autoIncParameter.GetParameterName()
                      << " is described as: " <<
                      autoIncParameter.GetDescription() << "." << std::endl;
```

```
            if (autoIncParameter.ParameterValueHasBeenSet()) {
                std::cout << "The current value is "
                          << autoIncParameter.GetParameterValue()
                          << "." << std::endl;
            }
            std::vector<int> splitValues = splitToInts(
                    autoIncParameter.GetAllowedValues(), '-');
            if (splitValues.size() == 2) {
                int newValue = askQuestionForIntRange(
                        Aws::String("Enter a new value between ") +
                        autoIncParameter.GetAllowedValues() + ": ",
                        splitValues[0], splitValues[1]);
                autoIncParameter.SetParameterValue(std::to_string(newValue));
                updateParameters.push_back(autoIncParameter);

            }
            else {
                std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                          << std::endl;
            }
        }
    }

    {
        // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

```
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                                userParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "  " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
                  << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;

    Aws::RDS::Model::DBCluster dbCluster;
    // 7.  Check if the DB cluster already exists.
    if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::String engineVersionName;
    Aws::String engineName;
    if (dbCluster.DBClusterIdentifierHasBeenSet()) {
        std::cout << "The DB cluster already exists." << std::endl;
        engineVersionName = dbCluster.GetEngineVersion();
        engineName = dbCluster.GetEngine();

    }
    else {
        std::cout << "Let's create a DB cluster." << std::endl;
        const Aws::String administratorName = askQuestion(
```

```
                "Enter an administrator username for the database: ");
        const Aws::String administratorPassword = askQuestion(
                "Enter a password for the administrator (at least 8 characters):
");
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 8.  Get a list of engine versions for the parameter group family.
        if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                 client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }

        std::cout << "The available engines for your parameter group family are:"
                  << std::endl;

        int index = 1;
        for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
            std::cout << "  " << index << ": " <<
engineVersion.GetEngineVersion()
                      << std::endl;
            ++index;
        }
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                              1];

        engineName = engineVersion.GetEngine();
        engineVersionName = engineVersion.GetEngineVersion();
        std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                  << "' and database '" << DB_NAME << "'.\n"
                  << "The DB cluster is configured to use your custom cluster
parameter group '"
                  << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;
```

```
        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
    }

    std::cout << "Waiting for the DB cluster to become available." << std::endl;

    int counter = 0;
    // 11. Wait for the DB cluster to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for cluster to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
```

```
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
""",
                             client);
            return false;
        }

        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;
```

```
        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
    }

    std::cout << "Waiting for the DB instance to become available." << std::endl;

    counter = 0;
    // 14. Wait for the DB instance to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for instance to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        dbInstance = Aws::RDS::Model::DBInstance();
```

```
        if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB instance status is '"
                      << dbInstance.GetDBInstanceStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbInstance.GetDBInstanceStatus() != "available");

    if (dbInstance.GetDBInstanceStatus() == "available") {
        std::cout << "The DB instance has been created." << std::endl;
    }

    // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
    displayConnection(dbCluster);

    printAsterisksLine();

    if (askYesNoQuestion(
            "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
        Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                               Aws::String(Aws::Utils::UUID::RandomUUID()));
        {
            std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
            std::cout << "This typically takes a few minutes." << std::endl;

            // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
```

```
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
        }

        std::cout << "Waiting for the snapshot to become available." <<
std::endl;

        Aws::RDS::Model::DBClusterSnapshot snapshot;
        counter = 0;
        do {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }

    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB cluster, DB instance, and parameter
 group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                  DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                  client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
```

```
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                             const Aws::String &namePrefix,
                                             const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                             const Aws::RDS::RDSClient &client) {
```

```
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
```

```
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

```
    } while (!marker.empty());

    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

```
//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
```

```
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);
```

```
            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    if (!dbClusterIdentifier.empty()) {
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
```

```
                        << outcome.GetError().GetMessage()
                        << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
counter
                      << " seconds." << std::endl;
            return false;
        }

        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus() << "." <<
std::endl;
            }

            if (clusterDeleting) {
```

```
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk C++ .
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter

- Jelaskan DBCluster Snapshots
- Jelaskan DBClusters
- Jelaskan DBEngine Versi
- Jelaskan DBInstances
- DescribeOrderableDBInstancePilihan
- Memodifikasi DBCluster ParameterGroup

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di prompt perintah.

```
import (
 "aurora/actions"
 "context"
 "fmt"
 "log"
 "slices"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/google/uuid"
)
```

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()
```

```
 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine,
 parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 cluster := scenario.CreateCluster(ctx, clusterName, dbEngine, dbName,
 parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(ctx, cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(ctx, clusterName)
 scenario.Cleanup(ctx, dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
```

```
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName,
 "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
```

```
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(ctx, parameterGroupName,
 updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
 userParameters, err := scenario.dbClusters.GetParameters(ctx,
 parameterGroupName, "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(ctx context.Context, clusterName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
```

```
 cluster, err := scenario.dbClusters.GetDbCluster(ctx, clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   ctx, clusterName, *parameterGroup.DBClusterParameterGroupName, dbName,
 dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(ctx, clusterName)
   if err != nil {
    panic(err)
   }
   log.Println("Cluster created and available.")
  }
 }
```

```
 log.Println("Cluster data:")
 log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
 log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
 log.Printf("\tStatus: %v\n", *cluster.Status)
 log.Printf("\tEngine: %v\n", *cluster.Engine)
 log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
 log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
 log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
 log.Println(strings.Repeat("-", 88))
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(ctx context.Context, cluster
 *types.DBCluster) *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   ctx, *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  slices.Sort(instChoices)
  instChoices = slices.Compact(instChoices)
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
  log.Println("Creating a database instance. This typically takes several
 minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
```

```
    ctx, *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier,
  *cluster.Engine,
    instChoices[instIndex])
   if err != nil {
    panic(err)
   }
   for *dbInstance.DBInstanceStatus != "available" {
    scenario.helper.Pause(30)
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
  *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Println("Instance data:")
  log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
  log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
  log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
  log.Printf("\tEngine: %v\n", *dbInstance.Engine)
  log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
  log.Println(strings.Repeat("-", 88))
  return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
  and tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
  log.Println(
   "You can now connect to your database using your favorite MySql client.\n" +
    "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
  +
    "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
    "port, and administrator user name to 'mysql' and enter your password\n" +
    "when prompted:")
  log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
   *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
  log.Println("For more information, see the User Guide for Aurora:\n" +
   "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
  log.Println(strings.Repeat("-", 88))
}
```

```
// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(ctx context.Context,
 clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(ctx, clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(ctx context.Context, dbInstance
 *types.DBInstance, cluster *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {
```

```
 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(ctx, *cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(ctx,
 *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err = scenario.dbClusters.DeleteParameterGroup(ctx,
 *parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}
```

```
// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Tentukan fungsi-fungsi yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
```

```
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
```

```
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

```
// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
```

```
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName
 string, parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

```
// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context,
 clusterName string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context,
 clusterName string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
```

```
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName
 string) error {
```

```
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx,
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {
```

```
 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter
  - Jelaskan DBCluster Snapshots
  - Jelaskan DBClusters
  - Jelaskan DBEngine Versi
  - Jelaskan DBInstances

- DescribeOrderableDBInstancePilihan
- Memodifikasi DBCluster ParameterGroup

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
```

```
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
    public static long sleepTime = 20;
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = "\n" +
                "Usage:\n" +
                "    <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
 <dbSnapshotIdentifier><secretName>"
                +
                "Where:\n" +
                "    dbClusterGroupName - The name of the DB cluster parameter
 group. \n" +
                "    dbParameterGroupFamily - The DB cluster parameter group
 family name (for example, aurora-mysql5.7). \n"
                +
                "    dbInstanceClusterIdentifier - The instance cluster
 identifier value.\n" +
                "    dbInstanceIdentifier - The database instance identifier.\n"
 +
                "    dbName - The database name.\n" +
                "    dbSnapshotIdentifier - The snapshot identifier.\n" +
                "    secretName - The name of the AWS Secrets Manager secret that
 contains the database credentials\"\n";
        ;

        if (args.length != 7) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String dbClusterGroupName = args[0];
        String dbParameterGroupFamily = args[1];
        String dbInstanceClusterIdentifier = args[2];
        String dbInstanceIdentifier = args[3];
        String dbName = args[4];
        String dbSnapshotIdentifier = args[5];
        String secretName = args[6];

        // Retrieve the database credentials using AWS Secrets Manager.
        Gson gson = new Gson();
        User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
        String username = user.getUsername();
        String userPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient,
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier,
dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)
```

```
.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
   }

   private static String getSecretValues(String secretName) {
       SecretsManagerClient secretClient = getSecretClient();
       GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
               .secretId(secretName)
               .build();

       GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
       return valueResponse.secretString();
   }

   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
```

```
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
```

```
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
            String dbInstanceClusterIdentifier) {
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }
```

```
            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
            String dbSnapshotIdentifier) {
        try {
            CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
```

```
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
```

```
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
```

```
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
    public static void modifyDBClusterParas(RdsClient rdsClient, String
dClusterGroupName) {
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
```

```
            ModifyDbClusterParameterGroupRequest groupRequest =
ModifyDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dClusterGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbClusterParameterGroupResponse response =
rdsClient.modifyDBClusterParameterGroup(groupRequest);
            System.out.println(
                    "The parameter group " +
response.dbClusterParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
        try {
            DescribeDbClusterParametersRequest dbParameterGroupsRequest;
            if (flag == 0) {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .build();
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
```

```
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
```

```
   }

   public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
           String dbParameterGroupFamily) {
       try {
           CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
           System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }

   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
```

```
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter
  - Jelaskan DBCluster Snapshots
  - Jelaskan DBClusters
  - Jelaskan DBEngine Versi
  - Jelaskan DBInstances
  - DescribeOrderableDBInstancePilihan
  - Memodifikasi DBCluster ParameterGroup

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
```

```
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 7) {
        println(usage)
        exitProcess(1)
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
 User::class.java)
    val username = user.username
    val userPassword = user.password

    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()
```

```
    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")

    println("9. Wait for DB instance to be ready")
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)

    println("10. Get a list of instance classes available for the selected
engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)

    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)
```

```
    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
```

```
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}

suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}

suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
```

```
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}

suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
```

```
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
```

```
            dbInstanceClass = instanceClassVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest =
        DescribeOrderableDbInstanceOptionsRequest {
            engine = "aurora-mysql"
            maxRecords = 20
        }
    var instanceClass = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbClustersRequest {
            dbClusterIdentifier = dbClusterIdentifierVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
```

```
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }
```

```
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
```

```
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest =
        DescribeDbEngineVersionsRequest {
            engine = "aurora-mysql"
            defaultOnly = true
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK For Kotlin.
  - Buat DBCluster

- Buat DBCluster ParameterGroup
- Buat DBCluster Snapshot
- Buat DBInstance
- Hapus DBCluster
- Hapus DBCluster ParameterGroup
- Hapus DBInstance
- Jelaskan DBCluster ParameterGroups
- Jelaskan DBCluster Parameter
- Jelaskan DBCluster Snapshots
- Jelaskan DBClusters
- Jelaskan DBEngine Versi
- Jelaskan DBInstances
- DescribeOrderableDBInstancePilihan
- Memodifikasi DBCluster ParameterGroup

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di prompt perintah.

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper
```

```
    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
engine and
        create a DB cluster parameter group that is compatible with a selected
engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
parameter group.
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB cluster parameter group named
{parameter_group_name}."
        )
        parameter_group =
self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group

    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.
```

```
        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
```

```
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [
                ver["EngineVersionDescription"] for ver in engine_versions
            ]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
                engine_versions[engine_index]["EngineVersion"],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
```

```
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.

        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list(
                {
                    opt["DBInstanceClass"] + ", storage type: " +
opt["StorageType"]
                    for opt in inst_opts
                }
            )
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
                f"Creating a database instance. This typically takes several
minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name,
```

```
                cluster_name,
                cluster["Engine"],
                inst_opts[inst_index]["DBInstanceClass"],
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
 how to
        connect to it.

        :param cluster: The DB cluster to display.
        """
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
```

```
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)

    def cleanup(self, db_inst, cluster, parameter_group):
        """
        Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
group.
        Before the DB cluster parameter group can be deleted, all associated DB
instances and
        DB clusters must first be deleted.

        :param db_inst: The DB instance to delete.
        :param cluster: The DB cluster to delete.
        :param parameter_group: The DB cluster parameter group to delete.
        """
        cluster_name = cluster["DBClusterIdentifier"]
        parameter_group_name = parameter_group["DBClusterParameterGroupName"]
        if q.ask(
            "\nDo you want to delete the database instance, DB cluster, and
parameter "
            "group (y/n)? ",
            q.is_yesno,
        ):
```

```
            print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(f"Deleting database cluster {cluster_name}.")
            self.aurora_wrapper.delete_db_cluster(cluster_name)
            print(
                "Waiting for the DB instance and DB cluster to delete.\n"
                "This typically takes several minutes."
            )
            while db_inst is not None or cluster is not None:
                wait(30)
                if db_inst is not None:
                    db_inst = self.aurora_wrapper.get_db_instance(
                        db_inst["DBInstanceIdentifier"]
                    )
                if cluster is not None:
                    cluster = self.aurora_wrapper.get_db_cluster(
                        cluster["DBClusterIdentifier"]
                    )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.aurora_wrapper.delete_parameter_group(parameter_group_name)

    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
parameter_group)
        wait(5)
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)
```

```
        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

Tentukan fungsi-fungsi yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.
```

```
        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
```

```
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
```

```
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
```

```
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster


    def create_db_cluster(
```

```
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
```

```
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

```
    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
```

```
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter
  - Jelaskan DBCluster Snapshots
  - Jelaskan DBClusters
  - Jelaskan DBEngine Versi
  - Jelaskan DBInstances
  - DescribeOrderableDBInstancePilihan
  - Memodifikasi DBCluster ParameterGroup

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankan di Repositori Contoh AWS Kode.

Pustaka yang berisi fungsi khusus skenario untuk skenario Aurora.

```
use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
struct MetadataError {
    message: Option<String>,
```

```
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
```

```
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}

pub struct AuroraScenario {
    rds: crate::rds::Rds,
```

```
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
 ScenarioError> {
        let describe_db_engine_versions =
 self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
```

```
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
```

```
            .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }

    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }
```

```
        Ok(())
    }

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }

    pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
        let describe_db_clusters_output = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_ref()
                    .expect("cluster identifier")
                    .as_str(),
            )
            .await;
        if let Err(err) = describe_db_clusters_output {
            return Err(ScenarioError::new("Failed to get cluster", &err));
        }

        let db_cluster = describe_db_clusters_output
            .unwrap()
            .db_clusters
            .and_then(|output| output.first().cloned());

        db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
    }

    // Get the parameter group. rds.DescribeDbClusterParameterGroups
```

```
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
           .collect::<Vec<_>>();

       Ok(parameters)
   }

   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   pub async fn update_auto_increment(
       &self,
       offset: u8,
       increment: u8,
   ) -> Result<(), ScenarioError> {
       let modify_db_cluster_parameter_group = self
           .rds
           .modify_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
```

```
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
```

```
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
```

```
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
```

```
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
```

```
   // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
   pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
       let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
       let snapshot = self
           .rds
           .snapshot_cluster(id, format!("{id}_{name}").as_str())
           .await;
       match snapshot {
           Ok(output) => match output.db_cluster_snapshot {
               Some(snapshot) => Ok(snapshot),
               None => Err(ScenarioError::with("Missing Snapshot")),
           },
           Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
       }
   }

   pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
       let mut clean_up_errors: Vec<ScenarioError> = vec![];

       // Delete the instance. rds.DeleteDbInstance.
       let delete_db_instance = self
           .rds
           .delete_db_instance(
               self.db_instance_identifier
                   .as_deref()
                   .expect("instance identifier"),
           )
           .await;
       if let Err(err) = delete_db_instance {
           let identifier = self
               .db_instance_identifier
               .as_deref()
               .unwrap_or("Missing Instance Identifier");
           let message = format!("failed to delete db instance {identifier}");
           clean_up_errors.push(ScenarioError::new(message, &err));
       } else {
           // Wait for the instance to delete
           let waiter = Waiter::default();
           while waiter.sleep().await.is_ok() {
               let describe_db_instances =
self.rds.describe_db_instances().await;
```

```
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;
```

```
        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
```

```
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
}

#[cfg(test)]
pub mod tests;
```

Pengujian untuk pustaka menggunakan automock di sekitar pembungkus Klien RDS.

```
use crate::rds::MockRdsImpl;
```

```
use super::*;

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
```

```
use secrecy::ExposeSecret;

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));
```

```
    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
```

```
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
```

```
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
```

```
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}

#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
```

```
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
```

```
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
```

```
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}
```

```
#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
```

```
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()
```

```
 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
```

```
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
```

```
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
```

```
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
```

```
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}
```

```
#[tokio::test]
async fn test_scenario_snapshot_invalid() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
```

Biner untuk menjalankan skenario dari depan ke ujung, menggunakan inquirer sehingga pengguna dapat membuat beberapa keputusan.

```
use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }
```

```
    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
```

```
        return Err(anyhow!("Failed to get available engines: {}", error));
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
 rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
 version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for
 engine: {err}")),
    }

    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
```

```
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
    show_parameters(scenario, warnings).await;

    let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
    let increment = prompt_number_or_default(warnings,
 "auto_increment_increment", 3);

    // Modify both the auto_increment_offset and auto_increment_increment
 parameters in one call in the custom parameter group. Set their ParameterValue
 fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    let update_auto_increment = scenario.update_auto_increment(offset,
 increment).await;

    if let Err(error) = update_auto_increment {
        warnings.push("Failed to update auto increment", error);
        return Err(());
    }

    // Get and display the updated parameters. Specify Source of 'user' to get
 just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
    show_parameters(scenario, warnings).await;

    let username = inquire::Text::new("Username for the database (default
'testuser')")
        .with_default("testuser")
        .with_initial_value("testuser")
        .prompt();

    if let Err(error) = username {
        warnings.push(
            "Failed to get username, using default",
            ScenarioError::with(format!("Error from inquirer: {error}")),
        );
        return Err(());
    }
    let username = username.unwrap();

    let password = inquire::Text::new("Password for the database (minimum 8
 characters)")
        .with_validator(|i: &str| {
            if i.len() >= 8 {
                Ok(inquire::validator::Validation::Valid)
            } else {
```

```
                Ok(inquire::validator::Validation::Invalid(
                    "Password must be at least 8 characters".into(),
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}",);

    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
```

```
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
```

```
    }
}

#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
```

```
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

Pembungkus di sekitar layanan Amazon RDS yang memungkinkan automocking untuk pengujian.

```
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
```

```
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
        RdsImpl { inner }
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
```

```
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
```

```
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
```

```
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }

    pub async fn snapshot_cluster(
```

```
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster(
```

```
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
}
```

Cargo.toml dengan dependensi yang digunakan dalam skenario ini.

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
```

```
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Rust.
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter
  - Jelaskan DBCluster Snapshots
  - Jelaskan DBClusters
  - Jelaskan DBEngine Versi
  - Jelaskan DBInstances
  - DescribeOrderableDBInstancePilihan
  - Memodifikasi DBCluster ParameterGroup

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

# Tindakan untuk Aurora menggunakan AWS SDKs

Contoh kode berikut menunjukkan bagaimana melakukan tindakan Aurora individu dengan. AWS SDKs Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Kutipan ini memanggil Aurora API dan merupakan kutipan kode dari program yang lebih besar yang harus dijalankan dalam konteks. Anda dapat melihat tindakan dalam konteks diSkenario untuk Aurora menggunakan AWS SDKs .

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat Referensi Amazon Aurora API.

Contoh

- Gunakan CreateDBCluster dengan AWS SDK
- Gunakan CreateDBClusterParameterGroup dengan AWS SDK
- Gunakan CreateDBClusterSnapshot dengan AWS SDK
- Gunakan CreateDBInstance dengan AWS SDK
- Gunakan DeleteDBCluster dengan AWS SDK
- Gunakan DeleteDBClusterParameterGroup dengan AWS SDK
- Gunakan DeleteDBInstance dengan AWS SDK
- Gunakan DescribeDBClusterParameterGroups dengan AWS SDK
- Gunakan DescribeDBClusterParameters dengan AWS SDK
- Gunakan DescribeDBClusterSnapshots dengan AWS SDK
- Gunakan DescribeDBClusters dengan AWS SDK
- Gunakan DescribeDBEngineVersions dengan AWS SDK
- Gunakan DescribeDBInstances dengan AWS SDK
- Gunakan DescribeOrderableDBInstanceOptions dengan AWS SDK atau CLI
- Gunakan ModifyDBClusterParameterGroup dengan AWS SDK

## Gunakan **`CreateDBCluster`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`CreateDBCluster`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,
    string parameterGroupName,
    string dbEngine,
    string dbEngineVersion,
    string adminName,
    string adminPassword)
{
    var request = new CreateDBClusterRequest
    {
        DatabaseName = dbName,
        DBClusterIdentifier = clusterIdentifier,
        DBClusterParameterGroupName = parameterGroupName,
        Engine = dbEngine,
```

```
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- Untuk detail API, lihat Membuat DBCluster di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::CreateDBClusterRequest request;
    request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
    request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
    request.SetEngine(engineName);
    request.SetEngineVersion(engineVersionName);
    request.SetMasterUsername(administratorName);
    request.SetMasterUserPassword(administratorPassword);

    Aws::RDS::Model::CreateDBClusterOutcome outcome =
            client.CreateDBCluster(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB cluster creation has started."
                  << std::endl;
```

```
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
```

- Untuk detail API, lihat Membuat DBCluster di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
```

```
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName
 string, parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- Untuk detail API, lihat Membuat DBCluster di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
           String dbClusterIdentifier, String userName, String password) {
       try {
           CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                   .databaseName(dbName)
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .dbClusterParameterGroupName(dbParameterGroupFamily)
                   .engine("aurora-mysql")
                   .masterUsername(userName)
                   .masterUserPassword(password)
                   .build();

           CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
           return response.dbCluster().dbClusterArn();

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
       return "";
   }
```

- Untuk detail API, lihat Membuat DBCluster di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
```

```
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}
```

- Untuk detail API, lihat Membuat DBCluster di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client
```

```
    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
```

```
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster
```

- Untuk detail API, lihat Membuat DBCluster di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).
```

```
   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
           .db_cluster
           .and_then(|c| c.db_cluster_identifier);

       if self.db_cluster_identifier.is_none() {
           return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
       }

       info!(
           "Started a db cluster: {}",
           self.db_cluster_identifier
               .as_deref()
```

```
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
```

```
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
```

```
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
```

```
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
```

```
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
```

```
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
```

```
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
```

```
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat referensi Create DBCluster in AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`CreateDBClusterParameterGroup`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`CreateDBClusterParameterGroup`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di Referensi AWS SDK untuk C ++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
```

```
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
           String dbParameterGroupFamily) {
       try {
           CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
           System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
```

```
        }
    }
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
 parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
 group.
        :param parameter_group_family: The family that is used as the basis of
 the new
                                       parameter group.
```

```
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
       self.engine_family = Some(engine.to_string());
       self.engine_version = Some(version.to_string());
```

```
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
```

```
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
```

```
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- Untuk detail API, lihat referensi Create DBCluster ParameterGroup in AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`CreateDBClusterSnapshot`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`CreateDBClusterSnapshot`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
```

```
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterSnapshotIdentifier(snapshotID);

        Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                client.CreateDBClusterSnapshot(request);

        if (outcome.IsSuccess()) {
            std::cout << "Snapshot creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context,
 clusterName string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
```

```
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .build();

           CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
           System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
```

```
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
```

```
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
```

```
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
           .db_cluster
           .and_then(|c| c.db_cluster_identifier);

       if self.db_cluster_identifier.is_none() {
           return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
       }

       info!(
           "Started a db cluster: {}",
           self.db_cluster_identifier
               .as_deref()
               .unwrap_or("Missing ARN")
       );
```

```
        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
```

```
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }
```

```
        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
 SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
```

```
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
```

```
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
```

```
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
```

```
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di AWS SDK untuk referensi Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`CreateDBInstance`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`CreateDBInstance`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- Untuk detail API, lihat Membuat DBInstance di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
```

- Untuk detail API, lihat Membuat DBInstance di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context,
 clusterName string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
```

```
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- Untuk detail API, lihat Membuat DBInstance di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
```

```
        }
        return "";
    }
```

- Untuk detail API, lihat Membuat DBInstance di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- Untuk detail API, lihat Membuat DBInstance di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
 that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
 created.
        :param db_engine: The database engine of a database to create in the DB
 instance.
```

```
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Untuk detail API, lihat Membuat DBInstance di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
```

```
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
```

```
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
```

```
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
```

```
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
```

```
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

  .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat referensi Create DBInstance in AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DeleteDBCluster`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DeleteDBCluster`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- Untuk detail API, lihat Menghapus DBCluster di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- Untuk detail API, lihat Menghapus DBCluster di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat Menghapus DBCluster di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
       try {
           DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .skipFinalSnapshot(true)
                   .build();

           rdsClient.deleteDBCluster(deleteDbClusterRequest);
           System.out.println(dbInstanceClusterIdentifier + " was deleted!");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat Menghapus DBCluster di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- Untuk detail API, lihat Menghapus DBCluster di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
```

```
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

- Untuk detail API, lihat Menghapus DBCluster di AWS SDK for Python (Boto3) Referensi API.

## Rust

### SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
```

```
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
```

```
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
```

```
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
```

```
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
```

```
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
```

```
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat Menghapus DBCluster di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DeleteDBClusterParameterGroup`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DeleteDBClusterParameterGroup`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
    request.SetDBClusterParameterGroupName(parameterGroupName);

    Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
            client.DeleteDBClusterParameterGroup(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB parameter group was successfully deleted."
                  << std::endl;
    }
    else {
        std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        result = false;
    }
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
```

```
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
```

```
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
```

```
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
```

```
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
```

```
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
```

```
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
```

```
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Cluster
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
```

```
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DeleteDBInstance`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DeleteDBInstance`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- Untuk detail API, lihat Menghapus DBInstance di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- Untuk detail API, lihat Menghapus DBInstance di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx,
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat Menghapus DBInstance di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
       try {
           DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .deleteAutomatedBackups(true)
                   .skipFinalSnapshot(true)
                   .build();

           DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
           System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat Menghapus DBInstance di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- Untuk detail API, lihat Menghapus DBInstance di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Untuk detail API, lihat Menghapus DBInstance di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
```

```
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
```

```
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
```

```
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
```

```
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
```

```
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
```

```
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
```

```
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat Menghapus DBInstance di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeDBClusterParameterGroups`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusterParameterGroups`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
```

```
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);
```

```
        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
 parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return parameter_group
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups dalam AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeDBClusterParameters`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusterParameters`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
```

```
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                             const Aws::String &namePrefix,
                                             const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                             const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
```

```
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
               // auto_increment_increment.
               paraName = para.parameterName();
               if ((paraName.compareTo("auto_increment_offset") == 0)
                       || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                   System.out.println("*** The parameter name is  " + paraName);
                   System.out.println("*** The parameter value is  " +
para.parameterValue());
```

```
                    System.out.println("*** The parameter data type is " +
 para.dataType());
                    System.out.println("*** The parameter description is " +
 para.description());
                    System.out.println("*** The parameter allowed values  is " +
 para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
```

```
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
${para.description}")
                    println("*** The parameter allowed values  is
${para.allowedValues}")
                }
            }
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
 filtered
                            to contain only parameters that start with this
 prefix.
        :param source: When specified, only parameters from this source are
 retrieved.
                       For example, a source of 'user' retrieves only parameters
 that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
 self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
```

```
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter dalam AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
```

```
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
                format!("Failed to retrieve parameters for
 {DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
 FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
 SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
```

```
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeDBClusterSnapshots`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusterSnapshots`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
```

```
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
```

```
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();

           while (!snapshotReady) {
               DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
               List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
               for (DBClusterSnapshot snapshot : snapshotList) {
                   snapshotReadyStr = snapshot.status();
                   if (snapshotReadyStr.contains("available")) {
                       snapshotReady = true;
                   } else {
                       System.out.println(".");
                       Thread.sleep(sleepTime * 5000);
                   }
               }
           }

           System.out.println("The Snapshot is available!");
```

```
        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
```

```
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeDBClusters`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusters`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Returns a list of DB clusters.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
   /// <returns>List of DB clusters.</returns>
   public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
   {
       var results = new List<DBCluster>();

       DescribeDBClustersResponse response;
       DescribeDBClustersRequest request = new DescribeDBClustersRequest
       {
           DBClusterIdentifier = dbClusterIdentifier
       };
       // Get the full list if there are multiple pages.
       do
       {
           response = await _amazonRDS.DescribeDBClustersAsync(request);
           if (response.DBClusters != null)
           {
               results.AddRange(response.DBClusters);
           }
           request.Marker = response.Marker;
       }
```

```
        while (response.Marker is not null);
        return results;
    }
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
```

```
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
               // auto_increment_increment.
               paraName = para.parameterName();
               if ((paraName.compareTo("auto_increment_offset") == 0)
                       || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                   System.out.println("*** The parameter name is  " + paraName);
                   System.out.println("*** The parameter value is  " +
para.parameterValue());
                   System.out.println("*** The parameter data type is " +
para.dataType());
                   System.out.println("*** The parameter description is " +
para.description());
                   System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
               }
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
```

```
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
```

```
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- Untuk detail API, lihat Menjelaskan DBClusters dalam AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
```

```
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
```

```
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));
```

```
            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
```

```
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
```

```
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
```

```
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
```

```
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
```

```
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeDBEngineVersions`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBEngineVersions`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
```

```
    }
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi dalam Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
```

```
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi dalam Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
```

```
  return output.DBEngineVersions, nil
 }
}
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi dalam Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
               System.out.println("The version number of the database engine " +
engineOb.engineVersion());
```

```
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi dalam Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
 list of
                                       engine versions to those that are
 compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
```

```
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi dalam AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");
```

```
        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
```

```
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}
```

```
#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeDBInstances`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBInstances`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Returns a list of DB instances.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
   /// <returns>List of DB instances.</returns>
   public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
   {
       var results = new List<DBInstance>();
       var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
           new DescribeDBInstancesRequest
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });
       // Get the entire list using the paginator.
       await foreach (var instances in instancesPaginator.DBInstances)
       {
           results.Add(instances);
       }
       return results;
   }
```

- Untuk detail API, lihat Menjelaskan DBInstances di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
```

```
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- Untuk detail API, lihat Menjelaskan DBInstances di Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- Untuk detail API, lihat Menjelaskan DBInstances di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
```

```
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat Menjelaskan DBInstances di Referensi AWS SDK for Java 2.x API.

## Kotlin

### SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- Untuk detail API, lihat Menjelaskan DBInstances di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
```

```
            return db_inst
```

- Untuk detail API, lihat Menjelaskan DBInstances dalam AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
```

```
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
```

```
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
```

```
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
```

```
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
```

```
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
```

```
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat Menjelaskan DBInstances di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`DescribeOrderableDBInstanceOptions`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DescribeOrderableDBInstanceOptions`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
```

```
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
```

```
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi AWS SDK untuk C++ API.

## Go

### SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
```

```
}
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
               System.out.println("The version number of the database engine " +
engineOb.engineVersion());
           }
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi AWS SDK for Java 2.x API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mencantumkan versi mesin DB yang mendukung kelas instans DB tertentu dalam file Wilayah AWS.

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

Contoh 2: Contoh ini mencantumkan kelas instans DB yang didukung untuk versi mesin DB tertentu dalam file Wilayah AWS.

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mencantumkan versi mesin DB yang mendukung kelas instans DB tertentu dalam file Wilayah AWS.

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

Contoh 2: Contoh ini mencantumkan kelas instans DB yang didukung untuk versi mesin DB tertentu dalam file Wilayah AWS.

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client
```

```
    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
       let describe_orderable_db_instance_options_items = self
           .rds
           .describe_orderable_db_instance_options(
               DB_ENGINE,
               self.engine_version
                   .as_ref()
                   .expect("engine version for db instance options")
                   .as_str(),
           )
           .await;

       describe_orderable_db_instance_options_items
           .map(|options| {
               options
                   .iter()
                   .filter(|o| o.storage_type() == Some("aurora"))
                   .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                   .collect::<Vec<String>>()
           })
           .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
   }

   pub async fn describe_orderable_db_instance_options(
       &self,
```

```
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
 SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
```

```
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Gunakan **`ModifyDBClusterParameterGroup`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`ModifyDBClusterParameterGroup`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
```

```
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
 _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup dalam Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup dalam Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup dalam Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup dalam Referensi AWS SDK for Java 2.x API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup di AWS SDK untuk referensi API Kotlin.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
```

```
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup dalam AWS SDK for Python (Boto3) Referensi API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   pub async fn update_auto_increment(
       &self,
       offset: u8,
       increment: u8,
   ) -> Result<(), ScenarioError> {
       let modify_db_cluster_parameter_group = self
           .rds
           .modify_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               vec![
                   Parameter::builder()
                       .parameter_name("auto_increment_offset")
```

```
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
 SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
```

```
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
```

```
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Skenario untuk Aurora menggunakan AWS SDKs

Contoh kode berikut menunjukkan kepada Anda bagaimana menerapkan skenario umum di Aurora dengan. AWS SDKs Skenario ini menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam Aurora atau dikombinasikan dengan yang lain. Layanan AWS Setiap skenario menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode.

Skenario menargetkan tingkat pengalaman menengah untuk membantu Anda memahami tindakan layanan dalam konteks.

Contoh

- Membuat API REST pustaka peminjaman
- Buat pelacak butir kerja Aurora Nirserver

### Membuat API REST pustaka peminjaman

Contoh kode berikut menunjukkan cara membuat pustaka peminjaman tempat pelanggan dapat meminjam dan mengembalikan buku dengan menggunakan API REST yang didukung oleh basis data Amazon Aurora.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan API Amazon Relational Database Service (Amazon RDS) dan AWS Chalice untuk membuat REST API yang didukung oleh database Amazon Aurora. AWS SDK untuk Python (Boto3) Layanan web sepenuhnya nirserver dan mewakili pustaka peminjaman sederhana tempat pelanggan dapat meminjam dan mengembalikan buku. Pelajari cara:

- Membuat dan mengelola klaster basis data Aurora nirserver.
- Gunakan AWS Secrets Manager untuk mengelola kredensi database.
- Menerapkan lapisan penyimpanan data yang menggunakan Amazon RDS untuk memindahkan data masuk dan keluar dari basis data.
- Gunakan AWS Chalice untuk menerapkan REST API tanpa server ke Amazon API Gateway dan. AWS Lambda
- Menggunakan paket Permintaan untuk mengirim permintaan ke layanan web.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- Aurora
- Lambda
- Secrets Manager

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

## Buat pelacak butir kerja Aurora Nirserver

Contoh-contoh kode berikut menunjukkan cara membuat aplikasi web yang melacak butir kerja dalam basis data Amazon Aurora Nirserver dan menggunakan Amazon Simple Email Service (Amazon SES) untuk mengirim laporan.

.NET

SDK untuk .NET

Menunjukkan cara menggunakan AWS SDK untuk .NET untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon Aurora dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan front end yang dibangun dengan React.js untuk berinteraksi dengan backend RESTful .NET.

- Integrasikan aplikasi web React dengan AWS layanan.
- Cantumkan, tambahkan, perbarui, dan hapus butir di tabel Aurora.
- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

C++

SDK for C++

Menunjukkan cara membuat aplikasi web yang melacak dan melaporkan butir kerja yang tersimpan dalam basis data Amazon Aurora Nirserver.

Untuk kode sumber lengkap dan instruksi tentang cara menyiapkan C++ REST API yang menanyakan data Amazon Aurora Tanpa Server dan untuk digunakan oleh aplikasi React, lihat contoh lengkapnya di. GitHub

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS

- Layanan Data Amazon RDS
- Amazon SES

Java

SDK for Java 2.x

Menunjukkan cara membuat aplikasi web yang melacak dan melaporkan butir kerja yang tersimpan dalam basis data Amazon RDS.

Untuk kode sumber lengkap dan petunjuk tentang cara menyiapkan Spring REST API yang menanyakan data Amazon Aurora Tanpa Server dan untuk digunakan oleh aplikasi React, lihat contoh lengkapnya di. GitHub

Untuk kode sumber lengkap dan instruksi tentang cara menyiapkan dan menjalankan contoh yang menggunakan JDBC API, lihat contoh lengkapnya di. GitHub

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan AWS SDK untuk JavaScript (v3) untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon Aurora dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan sisi depan yang dibangun dengan React.js untuk berinteraksi dengan backend Express Node.js.

- Integrasikan aplikasi web React.js dengan Layanan AWS.
- Cantumkan, tambahkan, dan perbarui butir di tabel Aurora.
- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Kotlin

SDK for Kotlin

Menunjukkan cara membuat aplikasi web yang melacak dan melaporkan butir kerja yang tersimpan dalam basis data Amazon RDS.

Untuk kode sumber lengkap dan petunjuk tentang cara menyiapkan Spring REST API yang menanyakan data Amazon Aurora Tanpa Server dan untuk digunakan oleh aplikasi React, lihat contoh lengkapnya di. GitHub

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

PHP

SDK for PHP

Menunjukkan cara menggunakan AWS SDK untuk PHP untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon RDS dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan front end yang dibangun dengan React.js untuk berinteraksi dengan backend RESTful PHP.

- Integrasikan aplikasi web React.js dengan AWS layanan.
- Cantumkan, tambahkan, perbarui, dan hapus butir di tabel Amazon RDS.

- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) untuk membuat layanan REST yang melacak item pekerjaan di database Amazon Aurora Tanpa Server dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan rangka kerja web Flask untuk menangani perutean HTTP dan terintegrasi dengan halaman web React untuk menyajikan aplikasi web yang berfungsi penuh.

- Bangun layanan Flask REST yang terintegrasi dengan. Layanan AWS
- Baca, tulis, dan perbarui butir kerja yang tersimpan dalam basis data Aurora Nirserver.
- Buat AWS Secrets Manager rahasia yang berisi kredensi database dan gunakan untuk mengautentikasi panggilan ke database.
- Gunakan Amazon SES untuk mengirim laporan email tentang item pekerjaan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS

- Amazon SES

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

# Praktik terbaik dengan Amazon Aurora

Di bagian berikut ini, Anda dapat menemukan informasi tentang praktik terbaik dan opsi untuk menggunakan atau memigrasikan data ke klaster DB Amazon Aurora.

Beberapa praktik terbaik untuk Amazon Aurora yang dikhususkan untuk mesin basis data tertentu. Untuk informasi selengkapnya tentang praktik terbaik Aurora yang dikhususkan untuk sebuah mesin basis data, lihat hal berikut.

| Mesin basis data | Praktik terbaik |
| --- | --- |
| Amazon Aurora MySQL | Lihat Praktik terbaik dengan Amazon Aurora My SQL |
| Amazon Aurora PostgreSQL | Lihat Praktik terbaik dengan Amazon Aurora PostgreSQL |

> **Note**
>
> Untuk rekomendasi umum terkait Aurora, lihat .

Topik

- Pedoman operasional dasar untuk Amazon Aurora
- Rekomendasi RAM instans DB
- AWS driver basis data
- Memantau Amazon Aurora
- Menggunakan grup parameter DB dan grup parameter klaster DB
- Video praktik terbaik Amazon Aurora

## Pedoman operasional dasar untuk Amazon Aurora

Berikut ini adalah pedoman operasional dasar yang harus diikuti setiap orang saat menggunakan Amazon Aurora. Perjanjian Tingkat Layanan Amazon RDS mewajibkan Anda untuk mengikuti pedoman berikut ini:

- Memantau penggunaan memori, CPU, dan penyimpanan Anda. Anda dapat mengatur Amazon CloudWatch untuk memberi tahu Anda saat pola penggunaan berubah atau saat Anda mendekati kapasitas penerapan. Dengan begitu, Anda dapat mempertahankan performa sistem dan ketersediaan.
- Jika aplikasi klien Anda menyimpan data Domain Name Service (DNS) dari instans DB Anda, tetapkan nilai time-to-live (TTL) kurang dari 30 detik. Alamat IP yang mendasari dari instans DB dapat berubah setelah failover. Dengan demikian, meng-cache data DNS untuk waktu yang lama dapat menyebabkan kegagalan koneksi jika aplikasi Anda mencoba terhubung ke alamat IP yang tidak lagi digunakan. Klaster DB Aurora dengan beberapa replika baca juga dapat mengalami kegagalan koneksi ketika koneksi menggunakan titik akhir pembaca dan salah satu instans replika baca berada dalam pemeliharaan atau dihapus.
- Uji failover klaster DB Anda untuk memahami berapa lama prosesnya untuk kasus penggunaan Anda. Pengujian failover dapat membantu Anda memastikan bahwa aplikasi yang mengakses klaster DB Anda dapat secara otomatis terhubung ke klaster DB baru setelah failover.

## Rekomendasi RAM instans DB

Untuk mengoptimalkan performa, alokasikan RAM yang cukup sehingga working set berada hampir sepenuhnya di dalam memori. Untuk menentukan apakah set kerja Anda hampir semuanya ada dalam memori, periksa metrik berikut di Amazon CloudWatch:

- `VolumeReadIOPS` – Metrik ini mengukur jumlah rata-rata operasi I/O baca dari volume klaster, yang dilaporkan pada interval 5 menit. Nilai `VolumeReadIOPS` harus kecil dan stabil. Dalam beberapa kasus, Anda mungkin menemukan I/O baca Anda melonjak atau lebih tinggi dari biasanya. Jika demikian, selidiki instans DB dalam klaster DB Anda untuk melihat instans DB mana yang menyebabkan peningkatan I/O.

  > **Tip**
  >
  > Jika klaster Aurora MySQL Anda menggunakan kueri paralel, Anda mungkin melihat peningkatan nilai `VolumeReadIOPS`. Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kuerinya cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.

- `BufferCacheHitRatio` – Metrik ini mengukur persentase permintaan yang dilayani oleh cache buffer instans DB dalam klaster DB Anda. Metrik ini memberikan wawasan tentang jumlah data yang dilayani dari memori.

  Rasio hit tinggi menunjukkan bahwa instans DB Anda memiliki cukup memori yang tersedia. Rasio hit rendah menunjukkan bahwa kueri Anda pada instans DB ini sering kali masuk ke disk. Selidiki beban kerja Anda untuk melihat kueri mana yang menyebabkan perilaku ini.

Jika, setelah menyelidiki beban kerja Anda, Anda mendapati bahwa Anda memerlukan lebih banyak memori, pertimbangkan untuk menaikkan skala kelas instans DB ke kelas yang memiliki lebih banyak RAM. Setelah itu, Anda dapat menyelidiki metrik yang dibahas sebelumnya dan terus menaikkan skalanya sesuai kebutuhan. Jika klaster Aurora Anda lebih besar dari 40 TB, jangan gunakan kelas instans db.t2, db.t3, atau db.t4g.

Untuk informasi selengkapnya, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

## AWS driver basis data

Kami merekomendasikan AWS rangkaian driver untuk konektivitas aplikasi. Driver telah dirancang untuk memberikan dukungan untuk waktu peralihan dan failover yang lebih cepat, dan otentikasi dengan, AWS Identity and Access Management (IAM) AWS Secrets Manager, dan Federated Identity. AWS Driver mengandalkan pemantauan status cluster DB dan menyadari topologi cluster untuk menentukan penulis baru. Pendekatan ini mengurangi waktu peralihan dan failover menjadi satu digit detik, dibandingkan dengan puluhan detik untuk driver open-source.

Ketika fitur layanan baru diperkenalkan, tujuan dari AWS rangkaian driver adalah untuk memiliki dukungan bawaan untuk fitur layanan ini.

Untuk informasi selengkapnya, lihat Menghubungkan ke cluster Aurora DB dengan driver AWS.

## Memantau Amazon Aurora

Amazon Aurora menyediakan berbagai metrik dan wawasan yang dapat Anda pantau untuk menentukan kondisi dan performa klaster DB Aurora Anda. Anda dapat menggunakan berbagai alat, seperti,, dan CloudWatch API AWS Management Console AWS CLI, untuk melihat metrik Aurora. Anda dapat melihat gabungan Performance Insights dan CloudWatch metrik di dasbor Performance Insights dan memantau instans DB Anda. Untuk menggunakan tampilan pemantauan ini, Wawasan

Performa harus diaktifkan untuk instans DB Anda. Untuk informasi tentang tampilan pemantauan ini, lihat Melihat metrik gabungan dengan dasbor Performance Insights.

Anda dapat membuat laporan analisis performa untuk periode waktu tertentu dan melihat wawasan yang diidentifikasi serta rekomendasi untuk menyelesaikan masalah. Untuk informasi selengkapnya, lihat Membuat laporan analisis kinerja di Performance Insights.

## Menggunakan grup parameter DB dan grup parameter klaster DB

Kami menyarankan agar Anda mencoba perubahan grup parameter DB dan grup parameter klaster DB pada klaster DB uji sebelum menerapkan perubahan grup parameter ke klaster DB produksi Anda. Pengaturan parameter mesin DB yang tidak tepat dapat menimbulkan efek merugikan yang tidak diinginkan, termasuk penurunan performa dan ketidakstabilan sistem.

Berhati-hatilah selalu saat memodifikasi parameter mesin DB dan cadangkan klaster DB sebelum memodifikasi grup parameter DB. Untuk informasi tentang pencadangan klaster DB Anda, lihat Mencadangkan dan memulihkan klaster DB Amazon Aurora.

## Video praktik terbaik Amazon Aurora

Saluran AWS Online Tech Talks on YouTube menyertakan presentasi video tentang praktik terbaik untuk membuat dan mengonfigurasi kluster Amazon Aurora DB agar lebih aman dan sangat tersedia. Lihat Praktik terbaik Amazon Aurora untuk ketersediaan tinggi.

# Melakukan pembuktian konsep dengan Amazon Aurora

Dalam informasi berikut ini, Anda dapat menemukan penjelasan tentang cara menyiapkan dan menjalankan bukti konsep untuk Aurora. Pembuktian konsep adalah penyelidikan yang Anda lakukan untuk melihat apakah Aurora cocok dengan aplikasi Anda. Pembuktian konsep dapat membantu Anda memahami fitur Aurora dalam konteks aplikasi basis data Anda sendiri dan perbandingan Aurora dengan lingkungan basis data Anda saat ini. Ini juga dapat menunjukkan tingkat upaya apa yang Anda butuhkan untuk memindahkan data, SQL kode port, menyesuaikan kinerja, dan menyesuaikan prosedur manajemen Anda saat ini.

Dalam topik ini, Anda dapat menemukan ikhtisar dan step-by-step garis besar prosedur dan keputusan tingkat tinggi yang terlibat dalam menjalankan bukti konsep, tercantum berikut. Untuk petunjuk mendetail, Anda dapat membuka tautan ke dokumentasi lengkap untuk subjek spesifik.

## Gambaran umum bukti konsep Aurora

Saat Anda melakukan bukti konsep untuk Amazon Aurora, Anda mempelajari apa yang diperlukan untuk mem-port data dan SQL aplikasi yang ada ke Aurora. Anda mencoba aspek penting Aurora dalam skala besar, menggunakan volume data dan aktivitas yang sesuai dengan lingkungan produksi Anda. Tujuannya adalah untuk menjadi yakin bahwa kecanggihan Aurora memang cocok untuk mengatasi tantangan yang menyebabkan Anda tidak dapat lagi mengandalkan infrastruktur basis data sebelumnya. Pada akhir pembuktian konsep, Anda akan memiliki rencana yang andal untuk melakukan penilaian tolok ukur performa dan pengujian aplikasi berskala lebih besar. Pada tahap ini, Anda memahami item pekerjaan terbesar dalam proses menuju deployment produksi.

Saran berikut tentang praktik terbaik dapat membantu Anda menghindari kesalahan umum yang dapat menyebabkan masalah selama proses penilaian tolok ukur. Namun, topik ini tidak mencakup step-by-step proses melakukan tolok ukur dan melakukan penyetelan kinerja. Prosedur tersebut bisa bervariasi, tergantung pada beban kerja Anda dan fitur Aurora yang Anda gunakan. Untuk informasi selengkapnya, baca dokumentasi yang terkait dengan performa seperti Mengelola performa dan penskalaan untuk klaster DB Aurora, Peningkatan performa Amazon Aurora MySQL, Kinerja dan penskalaan untuk Amazon Aurora PostgreSQL, dan Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Informasi dalam topik ini berlaku terutama untuk aplikasi di mana organisasi Anda menulis kode dan mendesain skema dan yang mendukung mesin database SQL sumber terbuka My SQL dan Postgre.

Jika Anda menguji aplikasi komersial atau kode yang dihasilkan oleh kerangka kerja aplikasi, Anda mungkin tidak memiliki fleksibilitas untuk menerapkan semua pedoman. Dalam kasus seperti itu, tanyakan kepada AWS perwakilan Anda untuk melihat apakah ada praktik terbaik Aurora atau studi kasus untuk jenis aplikasi Anda.

# 1. Identifikasi tujuan Anda

Saat Anda mengevaluasi Aurora sebagai bagian dari pembuktian konsep, Anda perlu memilih jenis pengukuran apa yang akan dilakukan dan cara mengevaluasi keberhasilan percobaan ini.

Anda harus memastikan bahwa semua fungsionalitas aplikasi Anda kompatibel dengan Aurora. Karena versi utama Aurora kompatibel dengan kabel dengan versi utama My SQL dan Postgre yang sesuaiSQL, sebagian besar aplikasi yang dikembangkan untuk mesin tersebut juga kompatibel dengan Aurora. Meski demikian, Anda masih harus memvalidasi kompatibilitas per aplikasi.

Misalnya, beberapa pilihan konfigurasi yang Anda buat saat menyiapkan klaster Aurora akan memengaruhi apakah Anda bisa atau harus menggunakan fitur basis data tertentu. Anda dapat memulai dengan klaster Aurora yang paling umum, yang dikenal sebagai klaster terprovisi. Anda kemudian dapat memutuskan apakah konfigurasi khusus seperti kueri nirserver atau kueri paralel mampu memberi manfaat untuk menangani beban kerja Anda.

Gunakan pertanyaan-pertanyaan berikut untuk membantu mengidentifikasi dan mengukur tujuan Anda:

- Apakah Aurora mendukung semua kasus penggunaan fungsional beban kerja Anda?
- Berapa ukuran set data atau tingkat beban yang Anda inginkan? Bisakah Anda meningkatkan skala ke tingkat itu?
- Apa persyaratan latensi atau throughput kueri spesifik Anda? Bisakah Anda memenuhi persyaratannya?
- Berapa lama waktu henti minimum terencana atau tidak terencana yang Anda sanggupi untuk beban kerja Anda? Bisakah Anda mencapainya?
- Apa metrik yang diperlukan untuk meraih efisiensi operasional? Bisakah Anda memantaunya secara akurat?
- Apakah Aurora mendukung tujuan bisnis spesifik Anda, seperti pengurangan biaya, peningkatan deployment, atau kecepatan penyediaan? Apakah Anda memiliki cara untuk mengukur tujuan ini?
- Dapatkah Anda memenuhi semua persyaratan keamanan dan kepatuhan untuk beban kerja Anda?

Luangkan waktu untuk membangun pengetahuan tentang mesin basis data dan kemampuan platform Aurora, serta tinjau dokumentasi layanan ini. Catat semua fitur yang dapat membantu Anda mencapai hasil yang Anda inginkan. Salah satunya mungkin konsolidasi beban kerja, dijelaskan dalam posting Blog AWS Database Cara merencanakan dan mengoptimalkan Amazon Aurora dengan kompatibilitas SQL Saya untuk beban kerja terkonsolidasi. Fitur lainnya adalah penskalaan berbasis permintaan, yang dijelaskan dalam Auto Scaling Amazon Aurora dengan Replika Aurora di Panduan Pengguna Amazon Aurora. Fiturnya yang lain adalah keuntungan performa atau operasi basis data yang disederhanakan.

## 2. Pahami karakteristik beban kerja Anda

Evaluasi Aurora dalam konteks kasus penggunaan yang Anda inginkan. Aurora adalah pilihan yang baik untuk pemrosesan transaksi online (OLTP) beban kerja. Anda juga dapat menjalankan laporan pada klaster yang menyimpan OLTP data real-time tanpa menyediakan klaster gudang data terpisah. Anda dapat mengenali apakah kasus penggunaan Anda termasuk dalam kategori ini atau tidak dengan melihat ciri-ciri berikut:

- Konkurensi yang tinggi, dengan puluhan, ratusan, atau bahkan ribuan klien simultan.
- Kueri berlatensi rendah dalam volume besar (milidetik hingga detik).
- Transaksi singkat dan waktu nyata.
- Pola kueri yang sangat selektif, dengan pencarian berbasis indeks.
- UntukHTAP, kueri analitis yang dapat memanfaatkan kueri paralel Aurora.

Salah satu faktor utama yang memengaruhi pilihan basis data Anda adalah kecepatan datanya. Kecepatan tinggi berkaitan dengan data yang sangat sering disisipkan dan diperbarui. Sistem seperti itu mungkin memiliki ribuan koneksi dan ratusan ribu kueri simultan yang membaca dan menulis ke basis data. Kueri dalam sistem berkecepatan tinggi biasanya akan memengaruhi jumlah baris yang relatif kecil, dan biasanya mengakses beberapa kolom dalam baris yang sama.

Aurora dirancang untuk menangani data berkecepatan tinggi. Bergantung pada beban kerjanya, klaster Aurora dengan satu instans DB r4.16xlarge dapat memproses lebih dari 600.000 pernyataan `SELECT` per detik. Sekali lagi, bergantung pada beban kerjanya, klaster seperti itu dapat memproses 200.000 pernyataan `INSERT`, `UPDATE`, dan `DELETE` per detik. Aurora adalah database toko baris dan sangat cocok untuk beban kerja volume tinggi, throughput tinggi, dan sangat paralel. OLTP

Aurora juga dapat menjalankan kueri pelaporan pada cluster yang sama yang menangani beban kerja. OLTP Aurora mendukung hingga 15 replika, yang masing-masing memiliki lag replikasi rata-

rata 10-20 milidetik dari instans primer. Analis dapat meminta OLTP data secara real time tanpa menyalin data ke cluster gudang data terpisah. Dengan menggunakan fitur kueri paralel untuk klaster Aurora, Anda dapat memindahkan sebagian besar pemrosesan, pemfilteran, dan agregasi ke subsistem penyimpanan Aurora yang didistribusikan secara masif.

Gunakan fase perencanaan ini untuk membiasakan diri dengan kemampuan Aurora, AWS layanan lain AWS Management Console, dan. AWS CLI Selain itu, periksa bagaimana kecocokan fungsinya dengan alat-alat lain yang nantinya Anda gunakan dalam pembuktian konsep.

# 3. Berlatih dengan AWS Management Console atau AWS CLI

Sebagai langkah selanjutnya, berlatih dengan AWS Management Console atau AWS CLI, untuk menjadi akrab dengan alat-alat ini dan dengan Aurora.

## Berlatihlah dengan AWS Management Console

Aktivitas awal berikut dengan cluster database Aurora terutama agar Anda dapat membiasakan diri dengan AWS Management Console lingkungan dan berlatih menyiapkan dan memodifikasi cluster Aurora. Jika Anda menggunakan mesin database My SQL -compatible dan Postgre SQL -compatible dengan AmazonRDS, Anda dapat membangun pengetahuan itu ketika Anda menggunakan Aurora.

Dengan memanfaatkan model dan fitur penyimpanan bersama dari Aurora seperti replikasi dan snapshot, Anda dapat memperlakukan seluruh klaster basis data sebagai jenis objek lain yang dapat Anda manipulasi dengan bebas. Anda dapat menyusun, merombak, dan mengubah kapasitas klaster Aurora selama proses pembuktian konsep. Anda tidak akan terkunci pada pilihan awal tentang kapasitas, pengaturan basis data, dan tata letak data fisik.

Untuk memulai, siapkan klaster Aurora yang kosong. Pilih jenis kapasitas terprovisi dan lokasi wilayah untuk eksperimen awal Anda.

Connect ke cluster itu menggunakan program klien seperti aplikasi SQL command-line. Awalnya, Anda akan terhubung menggunakan titik akhir klaster. Anda terhubung ke titik akhir tersebut untuk melakukan operasi penulisan apa pun, seperti pernyataan bahasa definisi data (DDL) dan proses ekstrak, transformasi, load (ETL). Selanjutnya, dalam pembuktian konsep, Anda perlu menghubungkan sesi sarat kueri menggunakan titik akhir pembaca, yang mendistribusikan beban kerja kueri ke beberapa instans DB di klaster.

Skalakan ke luar klaster dengan menambahkan lebih banyak Replika Aurora. Untuk prosedurnya, lihat Replikasi dengan Amazon Aurora. Skala instance DB naik atau turun dengan mengubah

kelas AWS instance. Pahami cara Aurora dapat menyederhanakan jenis operasi ini, sehingga jika perkiraan awal Anda untuk kapasitas sistem tidak akurat, Anda dapat menyesuaikannya nanti tanpa harus memulai dari awal.

Buat snapshot dan pulihkan ke klaster yang berbeda.

Periksa metrik klaster untuk melihat aktivitas dari waktu ke waktu, dan kecocokan metrik tersebut untuk instans DB di klaster.

Sangat berguna untuk menjadi akrab dengan bagaimana melakukan hal-hal ini AWS Management Console di awal. Setelah memahami apa yang dapat Anda lakukan dengan Aurora, Anda dapat melanjutkan untuk mengotomatiskan operasi tersebut menggunakan AWS CLI. Di bagian berikut, Anda dapat menemukan detail lebih lanjut tentang prosedur dan praktik terbaik untuk kegiatan ini selama proof-of-concept periode tersebut.

## Berlatihlah dengan AWS CLI

Kami merekomendasikan untuk mengotomatiskan prosedur penerapan dan manajemen, bahkan dalam pengaturan. proof-of-concept Untuk melakukannya, menjadi akrab dengan AWS CLI jika Anda belum melakukannya. Jika Anda menggunakan mesin database My SQL -compatible dan Postgre SQL -compatible dengan AmazonRDS, Anda dapat membangun pengetahuan itu ketika Anda menggunakan Aurora.

Aurora biasanya menangani grup instans DB yang disusun dalam klaster. Dengan demikian, banyak operasi akan mengharuskan Anda menentukan instans DB mana yang terkait dengan sebuah klaster, lalu melakukan operasi administratif dalam satu rangkaian untuk semua instans.

Misalnya, Anda dapat mengotomatiskan langkah-langkah seperti pembuatan klaster Aurora, lalu menaikkan skalanya dengan kelas instans yang lebih besar atau menskalakan klaster tersebut ke luar dengan instans DB tambahan. Dengan demikian, Anda akan dapat mengulangi tahapan mana pun dalam proses pembuktian konsep Anda dan mengkaji skenario bagaimana-jika dengan berbagai jenis atau konfigurasi klaster Aurora.

Pelajari kemampuan dan batasan alat deployment infrastruktur seperti AWS CloudFormation. Anda mungkin menemukan aktivitas yang Anda lakukan dalam proof-of-concept konteks tidak cocok untuk penggunaan produksi. Misalnya, AWS CloudFormation perilaku modifikasi adalah membuat instance baru dan menghapus yang sekarang, termasuk datanya. Untuk detail selengkapnya tentang perilaku ini, lihat Pembaruan perilaku sumber daya tumpukan dalam Panduan Pengguna AWS CloudFormation .

# 4. Buat klaster Aurora Anda

Dengan Aurora, Anda dapat mengkaji skenario bagaimana-jika dengan menambahkan instans DB ke klaster dan menaikkan skala instans DB ke kelas instans yang berperforma lebih tinggi. Anda juga dapat membuat klaster dengan pengaturan konfigurasi yang berbeda-beda untuk menjalankan beban kerja yang sama secara berdampingan. Dengan Aurora, Anda memiliki banyak fleksibilitas untuk menyiapkan, merombak, dan mengonfigurasi ulang klaster DB. Mengingat hal ini, akan sangat membantu untuk mempraktikkan teknik-teknik ini pada tahap awal proof-of-concept proses. Untuk prosedur umum membuat klaster Aurora, lihat Membuat klaster DB Amazon Aurora.

Jika memungkinkan, mulailah dengan sebuah klaster yang menggunakan pengaturan berikut. Lewati langkah ini jika Anda memiliki kasus penggunaan khusus tertentu. Misalnya, Anda mungkin perlu melewati langkah ini jika kasus penggunaan Anda memerlukan jenis klaster Aurora khusus. Atau Anda dapat melewati langkah ini jika Anda memerlukan kombinasi mesin dan versi basis data khusus.

- Nonaktifkan Pembuatan mudah. Sebagai pembuktian konsep, kami menyarankan Anda untuk mengetahui semua pengaturan yang Anda pilih sehingga Anda dapat membuat klaster yang identik atau sedikit berbeda nantinya.
- Gunakan versi mesin DB terbaru. Kombinasi mesin database dan versi ini memiliki kompatibilitas yang luas dengan fitur Aurora lainnya dan penggunaan pelanggan yang substansif untuk aplikasi produksi.
  - Aurora SQL Versi saya 3.x (Kompatibilitas 8.0 sayaSQL)
  - Aurora Postgre SQL versi 15.x atau 16.x
- Pilih templat Dev/Tes. Pilihan ini tidak signifikan untuk proof-of-concept aktivitas Anda.
- Untuk Kelas instans DB, pilih Kelas teroptimasi memori dan salah satu kelas instans xlarge. Anda dapat menyesuaikan kelas instans ke atas atau ke bawah nantinya.
- Di bagian Deployment Multi-AZ, pilih Buat Replika Aurora atau simpul Pembaca di AZ yang berbeda. Banyak aspek Aurora yang paling berguna menangani klaster yang terdiri dari beberapa instans DB. Sebaiknya mulai selalu dengan setidaknya dua instans DB di klaster baru. Penggunaan Zona Ketersediaan yang berbeda untuk instans DB kedua akan membantu menguji berbagai skenario ketersediaan tinggi.
- Saat memilih nama untuk instans DB, gunakan konvensi penamaan yang generik. Jangan merujuk ke instans DB cluster apa pun sebagai “penulis,” karena instance DB yang berbeda mengambil peran tersebut sesuai kebutuhan. Kami merekomendasikan format seperti `clustername-az-`

`serialnumber`, misalnya `myprodappdb-a-01`. Bagian-bagian ini mengidentifikasi secara unik instans DB dan penempatannya.

- Tetapkan retensi cadangan yang tinggi untuk klaster Aurora. Dengan periode retensi yang lama, Anda dapat melakukan point-in-time pemulihan (PITR) untuk jangka waktu hingga 35 hari. Anda dapat mengatur ulang database Anda ke status yang diketahui setelah menjalankan pengujian yang melibatkan DDL dan pernyataan bahasa manipulasi data (DML). Anda juga dapat memulihkannya jika Anda tidak sengaja menghapus atau mengubah data.
- Aktifkan fitur pemulihan, pencatatan log, dan pemantauan tambahan pada pembuatan klaster. Aktifkan semua pilihan yang tersedia di Backtrack, Performance Insights, Monitoring, dan ekspor Log. Dengan mengaktifkan berbagai fitur ini, Anda dapat menguji kesesuaian fitur seperti backtracking, Pemantauan yang Ditingkatkan, atau Wawasan Performa untuk beban kerja Anda. Anda juga dapat dengan mudah memeriksa performa dan melakukan pemecahan masalah selama proses pembuktian konsep.

## 5. Siapkan skema Anda

Pada klaster Aurora, siapkan basis data, tabel, indeks, kunci asing, dan objek skema lainnya untuk aplikasi Anda. Jika Anda pindah dari sistem database My SQL -compatible atau Postgre SQL -compatible lainnya, harapkan tahap ini sederhana dan mudah. Anda menggunakan SQL sintaks dan baris perintah yang sama atau aplikasi klien lain yang Anda kenal untuk mesin database Anda.

Untuk mengeluarkan SQL pernyataan di klaster Anda, temukan titik akhir klaster dan berikan nilai tersebut sebagai parameter koneksi ke aplikasi klien Anda. Anda dapat menemukan titik akhir klaster di tab Konektivitas di halaman detail klaster Anda. Titik akhir klaster adalah yang berlabel Penulis. Titik akhir lainnya, yang berlabel Pembaca, merepresentasikan koneksi hanya baca yang dapat Anda berikan kepada pengguna akhir yang menjalankan laporan atau kueri hanya baca lainnya. Untuk memperoleh bantuan atas masalah apa pun terkait koneksi ke klaster Anda, lihat Menghubungkan ke klaster DB Amazon Aurora.

Jika Anda mem-porting skema dan data Anda dari sistem basis data yang berbeda, Anda perlu membuat beberapa perubahan skema pada saat ini. Perubahan skema ini sesuai dengan SQL sintaks dan kemampuan yang tersedia di Aurora. Anda dapat menghapus kolom, batasan, pemicu, atau objek skema tertentu pada tahap ini. Dengan melakukannya, Anda akan memperoleh manfaat, terutama jika objek ini memerlukan pengerjaan ulang untuk kompatibilitas Aurora dan tidak signifikan untuk tujuan Anda dengan pembuktian konsep.

Jika Anda bermigrasi dari sistem database dengan mesin dasar yang berbeda dari Aurora, pertimbangkan untuk menggunakan AWS Schema Conversion Tool (AWS SCT) untuk menyederhanakan proses. Untuk detailnya, lihat Panduan Pengguna AWS Schema Conversion Tool. Untuk detail umum tentang aktivitas migrasi dan porting, lihat whitepaper Migrasi Database Anda ke Amazon Aurora. AWS

Selama tahap ini, Anda dapat mengevaluasi apakah terdapat inefisiensi dalam penyiapan skema Anda, misalnya dalam strategi pengindeksan atau struktur tabel lain seperti tabel yang dipartisi. Inefisiensi seperti itu akan meningkat saat Anda men-deploy aplikasi Anda pada klaster dengan beberapa instans DB dan beban kerja yang berat. Pertimbangkan apakah Anda dapat menyempurnakan aspek performa tersebut sekarang, atau selama aktivitas selanjutnya seperti pengujian tolok ukur lengkap.

## 6. Impor data Anda

Selama pembuktian konsep, Anda akan membawa data, atau sampel representatif, dari sistem basis data sebelumnya. Jika memungkinkan, siapkan setidaknya beberapa data di setiap tabel Anda. Hal tersebut akan membantu menguji kompatibilitas semua jenis data dan fitur skema. Setelah Anda mencoba berbagai fitur Aurora dasar, tingkatkan jumlah datanya. Pada saat Anda menyelesaikan bukti konsep, Anda harus menguji ETL alat, kueri, dan beban kerja Anda secara keseluruhan dengan kumpulan data yang cukup besar untuk menarik kesimpulan yang akurat.

Anda dapat menggunakan beberapa teknik untuk mengimpor data cadangan fisik atau logis ke Aurora. Untuk detail, lihat Migrasi data ke cluster Amazon Aurora My DB SQL atau Migrasi data ke Amazon Aurora dengan kompatibilitas Postgre SQL sesuai dengan mesin basis data yang Anda gunakan dalam bukti konsep.

Bereksperimenlah dengan ETL alat dan teknologi yang sedang Anda pertimbangkan. Lihat mana yang paling sesuai dengan kebutuhan Anda. Pertimbangkan throughput dan fleksibilitasnya. Misalnya, beberapa ETL alat melakukan transfer satu kali, dan yang lain melibatkan replikasi berkelanjutan dari sistem lama ke Aurora.

Jika Anda bermigrasi dari sistem yang SQL kompatibel dengan Saya ke Aurora MySQL, Anda dapat menggunakan alat transfer data asli. Hal yang sama berlaku jika Anda bermigrasi dari SQL sistem yang kompatibel dengan Postgre ke Aurora Postgre. SQL Jika Anda bermigrasi dari sistem database yang menggunakan mesin dasar yang berbeda dari Aurora, Anda dapat bereksperimen dengan AWS Database Migration Service ().AWS DMS Untuk detailnya AWS DMS, lihat Panduan AWS Database Migration Service Pengguna.

Untuk detail tentang aktivitas migrasi dan porting, lihat buku AWS putih buku pegangan migrasi Aurora.

# 7. Port SQL kode Anda

Mencoba SQL dan aplikasi terkait membutuhkan tingkat upaya yang berbeda tergantung pada kasus yang berbeda. Secara khusus, tingkat upaya tergantung pada apakah Anda pindah dari sistem My SQL -compatible atau Postgre SQL -compatible atau jenis lain.

- Jika Anda pindah dari RDS untuk My SQL atau RDS PostgreSQL, SQL perubahannya cukup kecil sehingga Anda dapat mencoba SQL kode asli dengan Aurora dan secara manual memasukkan perubahan yang diperlukan.
- Demikian pula, jika Anda berpindah dari database lokal yang kompatibel dengan My SQL atau PostgreSQL, Anda dapat mencoba SQL kode asli dan memasukkan perubahan secara manual.
- Jika Anda berasal dari database komersial yang berbeda, SQL perubahan yang diperlukan lebih luas. Dalam hal ini, pertimbangkan untuk menggunakan AWS SCT.

Selama tahap ini, Anda dapat mengevaluasi apakah terdapat inefisiensi dalam penyiapan skema Anda, misalnya dalam strategi pengindeksan atau struktur tabel lain seperti tabel yang dipartisi. Pertimbangkan apakah Anda dapat menyempurnakan aspek performa tersebut sekarang, atau selama aktivitas selanjutnya seperti pengujian tolok ukur lengkap.

Anda dapat memverifikasi logika koneksi basis data di aplikasi Anda. Untuk memanfaatkan pemrosesan terdistribusi Aurora, Anda mungkin perlu menggunakan koneksi terpisah untuk operasi baca dan tulis, dan menggunakan sesi yang relatif singkat untuk operasi kueri. Untuk informasi tentang koneksi, lihat 9. Hubungkan ke Aurora.

Pertimbangkan apakah Anda harus membuat kompromi dan pengorbanan untuk mengatasi masalah dalam basis data produksi Anda. Bangun waktu ke dalam proof-of-concept jadwal untuk melakukan perbaikan pada desain dan kueri skema Anda. Untuk menilai apakah Anda dapat meraih keuntungan dari segi performa, biaya pengoperasian, dan skalabilitas yang baik, coba aplikasi asli dan yang dimodifikasi secara berdampingan di klaster Aurora yang berbeda.

Untuk detail tentang aktivitas migrasi dan porting, lihat buku AWS putih buku pegangan migrasi Aurora.

# 8. Tentukan pengaturan konfigurasi

Anda juga dapat meninjau parameter konfigurasi database Anda sebagai bagian dari latihan Aurora proof-of-concept . Anda mungkin sudah memiliki pengaturan SQL konfigurasi Saya SQL atau Postgre yang disetel untuk kinerja dan skalabilitas di lingkungan Anda saat ini. Subsistem penyimpanan Aurora diubah dan diatur untuk lingkungan berbasis cloud terdistribusi dengan subsistem penyimpanan berkecepatan tinggi. Akibatnya, banyak pengaturan mesin basis data sebelumnya tidak berlaku. Kami merekomendasikan untuk melakukan eksperimen awal Anda dengan pengaturan konfigurasi Aurora default. Terapkan kembali pengaturan dari lingkungan Anda saat ini hanya jika Anda mengalami hambatan performa dan skalabilitas. Jika Anda tertarik, Anda dapat melihat lebih dalam subjek ini dalam Memperkenalkan mesin penyimpanan Aurora di Blog AWS Database.

Aurora memudahkan penggunaan kembali pengaturan konfigurasi optimal untuk aplikasi atau kasus penggunaan tertentu. Alih-alih mengedit file konfigurasi terpisah untuk setiap instans DB, Anda mengelola sekumpulan parameter yang Anda tetapkan ke seluruh klaster atau instans DB tertentu. Misalnya, pengaturan zona waktu berlaku untuk semua instans DB di klaster, dan Anda dapat menyesuaikan pengaturan ukuran cache halaman untuk setiap instans DB.

Anda memulai dengan salah satu set parameter default, dan menerapkan perubahan ke parameter yang perlu Anda sesuaikan saja. Untuk detail tentang menggunakan grup parameter, lihat Parameter klaster DB dan instans DB Amazon Aurora. Untuk pengaturan konfigurasi yang berlaku atau tidak berlaku untuk klaster Aurora, lihat Parameter konfigurasi Aurora MySQL atau Parameter Amazon Aurora PostgreSQL tergantung mesin basis data Anda.

# 9. Hubungkan ke Aurora

Seperti yang Anda temukan saat membuat skema awal dan pengaturan data serta menjalankan kueri sampel, Anda dapat terhubung ke titik akhir yang berbeda-beda dalam klaster Aurora. Titik akhir yang akan digunakan bergantung pada apakah operasinya adalah pembacaan, seperti pernyataan `SELECT` atau penulisan, seperti pernyataan `CREATE` atau `INSERT`. Saat Anda meningkatkan beban kerja pada klaster Aurora dan bereksperimen dengan fitur Aurora, penting bagi aplikasi Anda untuk menetapkan setiap operasi ke titik akhir yang sesuai.

Dengan menggunakan titik akhir klaster untuk operasi penulisan, Anda selalu terhubung ke instans DB di klaster yang memiliki kemampuan baca/tulis. Secara default, hanya satu instans DB dalam klaster Aurora yang memiliki kemampuan baca/tulis. Instans DB ini disebut instans primer. Jika

instans primer asli tidak tersedia, Aurora mengaktifkan mekanisme failover dan instans DB yang berbeda akan mengambil alih sebagai instans primer.

Demikian pula, dengan mengarahkan pernyataan SELECT ke titik akhir pembaca, Anda menyebarkan pemrosesan kueri di antara instans DB dalam klaster. Setiap koneksi pembaca ditugaskan ke instance DB yang berbeda menggunakan resolusi round-robinDNS. Melakukan sebagian besar pekerjaan kueri pada Replika Aurora DB hanya-baca mengurangi beban pada instance utama, membebaskannya untuk menangani dan pernyataan. DDL DML

Menggunakan titik akhir ini akan mengurangi ketergantungan pada nama host yang di-hard-coding, dan membantu aplikasi Anda dipulihkan lebih cepat jika ada kegagalan instans DB.

> **Note**
>
> Aurora juga memiliki titik akhir kustom yang Anda buat. Titik akhir tersebut biasanya tidak diperlukan selama proses pembuktian konsep.

Replika Aurora dapat mengalami lag replika, meskipun lag ini biasanya berlangsung 10 hingga 20 milidetik. Anda bisa memantau lag replikasi dan memutuskan apakah lag ini masih dalam kisaran persyaratan konsistensi data Anda. Dalam beberapa kasus, kueri baca Anda mungkin memerlukan konsistensi baca yang kuat (read-after-writekonsistensi). Jika demikian, Anda dapat terus menggunakan titik akhir klaster dan bukan titik akhir pembaca.

Untuk memanfaatkan kemampuan Aurora secara utuh untuk eksekusi paralel terdistribusi, Anda mungkin perlu mengubah logika koneksinya. Tujuan Anda adalah menghindari pengiriman semua permintaan baca ke instans primer. Replika Aurora hanya baca akan bersiaga, dengan semua data yang sama, untuk menangani pernyataan SELECT. Kodekan logika aplikasi Anda untuk menggunakan titik akhir yang sesuai untuk setiap jenis operasi. Ikuti pedoman umum berikut:

- Hindari menggunakan satu string koneksi hard-code untuk semua sesi basis data.
- Jika praktis, lampirkan operasi tulis seperti DDL dan DML pernyataan dalam fungsi dalam kode aplikasi klien Anda. Dengan demikian, Anda dapat membuat berbagai jenis operasi menggunakan koneksi tertentu.
- Buat fungsi terpisah untuk operasi kueri. Aurora menetapkan setiap koneksi baru dengan titik akhir pembaca ke Replika Aurora yang berbeda untuk menyeimbangkan beban aplikasi sarat baca.
- Untuk operasi yang menangani sekumpulan kueri, tutup dan buka kembali koneksi ke titik akhir pembaca saat setiap kumpulan kueri terkait selesai. Gunakan pooling koneksi jika fiturnya tersedia

di tumpukan perangkat lunak Anda. Mengarahkan kueri ke koneksi yang berbeda akan membantu Aurora mendistribusikan beban kerja baca di beberapa instans DB di klaster.

Untuk informasi umum tentang manajemen koneksi dan titik akhir untuk Aurora, lihat Menghubungkan ke klaster DB Amazon Aurora. Untuk menyelam lebih dalam tentang hal ini, lihat Buku pegangan administrator SQL database Aurora saya — Manajemen koneksi.

## 10. Jalankan beban kerja Anda

Setelah pengaturan skema, data, dan konfigurasi dilakukan, Anda dapat mulai mencoba klaster dengan menjalankan beban kerja Anda. Gunakan beban kerja sebagai bukti konsep yang mencerminkan aspek utama beban kerja produksi Anda. Kami merekomendasikan untuk selalu membuat keputusan tentang kinerja menggunakan tes dan beban kerja dunia nyata daripada tolok ukur sintetis seperti sysbench atau -C. TPC Di mana pun praktis, kumpulkan pengukuran berdasarkan skema, pola kueri, dan volume penggunaan Anda sendiri.

Sebisa mungkin, tiru kondisi sebenarnya yang akan dialami aplikasi yang berjalan nantinya. Misalnya, Anda biasanya menjalankan kode aplikasi pada EC2 instance Amazon di AWS Region yang sama dan virtual private cloud (VPC) yang sama dengan cluster Aurora. Jika aplikasi produksi Anda berjalan pada beberapa EC2 instance yang mencakup beberapa Availability Zone, siapkan proof-of-concept lingkungan Anda dengan cara yang sama. Untuk informasi selengkapnya tentang AWS Wilayah, lihat Wilayah dan Zona Ketersediaan di Panduan RDS Pengguna Amazon. Untuk mempelajari lebih lanjut tentang VPC layanan Amazon, lihat Apa itu AmazonVPC? di Panduan VPC Pengguna Amazon.

Setelah Anda memastikan bahwa fitur dasar aplikasi Anda berfungsi dan Anda dapat mengakses data melalui Aurora, Anda akan dapat mencoba berbagai aspek klaster Aurora. Beberapa fitur yang mungkin ingin Anda coba adalah koneksi konkuren dengan penyeimbangan beban, transaksi konkuren, dan replikasi otomatis.

Pada tahap ini, mekanisme transfer datanya seharusnya sudah dikenal, sehingga Anda dapat menjalankan pengujian dengan proporsi data sampel yang lebih besar.

Di tahap inilah Anda dapat melihat efek dari perubahan pengaturan konfigurasi seperti batas memori dan batas koneksi. Tinjau kembali prosedur yang Anda kaji dalam 8. Tentukan pengaturan konfigurasi.

Anda juga dapat bereksperimen dengan mekanisme seperti membuat dan memulihkan snapshot. Misalnya, Anda dapat membuat cluster dengan kelas AWS instance yang berbeda, jumlah AWS

Replicas, dan sebagainya. Kemudian, di setiap klaster, Anda dapat memulihkan snapshot yang sama yang berisi skema dan semua data Anda. Untuk detail tentang siklus tersebut, lihat Membuat snapshot klaster DB dan Memulihkan dari snapshot klaster DB.

# 11. Ukur performa

Praktik terbaik dalam aspek ini dirancang untuk memastikan bahwa semua alat dan proses disiapkan secara tepat untuk dapat mengisolasi perilaku abnormal selama operasi beban kerja. Praktik terbaik ini juga disiapkan untuk memastikan bahwa Anda dapat mengidentifikasi kemungkinan penyebab masalah yang berlaku.

Anda selalu dapat melihat status klaster Anda saat ini, atau memeriksa tren dari waktu ke waktu, dengan memeriksa tab Pemantauan. Tab ini tersedia dari halaman detail konsol untuk setiap klaster atau instans DB Aurora. Ini menampilkan metrik dari layanan CloudWatch pemantauan Amazon dalam bentuk bagan. Anda dapat memfilter metrik berdasarkan nama, instans DB, dan periode waktu.

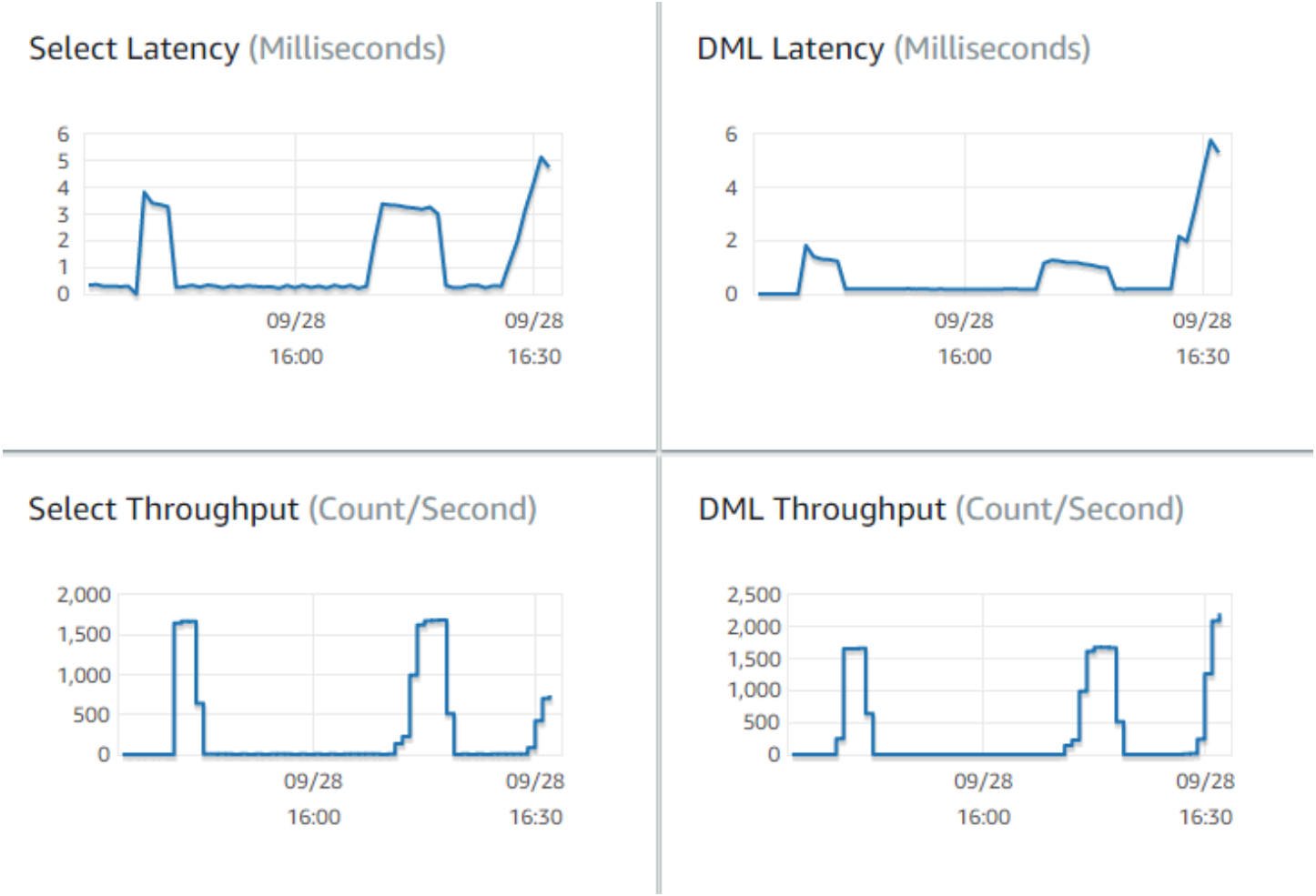
Untuk memiliki lebih banyak pilihan di tab Pemantauan, aktifkan Pemantauan yang Ditingkatkan dan Wawasan Performa di pengaturan klaster. Anda juga dapat mengaktifkan pilihan tersebut nanti jika Anda tidak memilihnya saat menyiapkan klaster.

Untuk mengukur performa, Anda sebagian besar akan mengandalkan bagan yang menampilkan aktivitas untuk seluruh klaster Aurora. Anda dapat memastikan apakah Replika Aurora memiliki waktu pemuatan dan waktu respons yang sama. Anda juga dapat melihat pemisahan pekerjaan antara instans primer baca/tulis dan Replika Aurora hanya baca. Jika ada ketidakseimbangan antara instans DB atau masalah yang hanya memengaruhi satu instans DB, Anda dapat memeriksa tab Pemantauan untuk instans yang bermasalah tersebut.

Setelah lingkungan dan beban kerja sebenarnya disiapkan untuk meniru aplikasi produksi Anda, Anda dapat mengukur seberapa baik performa Aurora. Berikut adalah pertanyaan terpenting yang perlu dijawab:

- Berapa banyak kueri per detik yang diproses Aurora? Anda dapat memeriksa metrik Throughput untuk melihat angka untuk berbagai jenis operasi yang ada.
- Berapa lama rata-rata waktu yang diperlukan Aurora untuk memproses kueri tertentu? Anda dapat memeriksa metrik Latensi untuk melihat angka untuk berbagai jenis operasi yang ada.

Untuk melihat metrik throughput dan latensi, periksa tab Monitoring untuk klaster Aurora tertentu di konsol Amazon. RDS Tangkapan layar berikut menunjukkan contoh metrik Select Latency, DMLLatency, Select Throughput, dan DML Throughput pada tab Monitoring.




Jika memungkinkan, tetapkan nilai acuan dasar untuk metrik ini di lingkungan Anda saat ini. Jika tidak memungkinkan, maka buat acuan dasar di klaster Aurora dengan menjalankan beban kerja yang setara dengan aplikasi produksi Anda. Misalnya, jalankan beban kerja Aurora Anda dengan jumlah pengguna dan kueri simultan yang sama. Kemudian, amati perubahan nilainya saat Anda bereksperimen dengan berbagai kelas instans, ukuran klaster, pengaturan konfigurasi, dan sebagainya.

Jika jumlah throughput lebih rendah dari yang Anda harapkan, periksa lebih lanjut faktor-faktor yang memengaruhi performa basis data untuk beban kerja Anda. Selain itu, lakukan pemeriksaan lebih lanjut jika angka latensi lebih tinggi dari yang Anda harapkan. Untuk melakukannya, pantau metrik sekunder untuk server DB (CPU, memori, dan sebagainya). Anda dapat melihat apakah instans DB mulai mendekati batasnya. Anda juga dapat melihat berapa banyak kapasitas ekstra yang dimiliki

instans DB Anda untuk menangani lebih banyak kueri konkuren, kueri terhadap tabel yang lebih besar, dan sebagainya.

> **Tip**
>
> Untuk mendeteksi nilai metrik yang berada di luar rentang yang diharapkan, atur CloudWatch alarm.

Saat mengevaluasi ukuran dan kapasitas klaster Aurora yang ideal, Anda akan dapat menemukan konfigurasi yang mendekati performa aplikasi maksimal tanpa penyediaan sumber daya yang berlebihan. Salah satu faktor penting adalah menemukan ukuran yang sesuai untuk instans DB di klaster Aurora. Mulailah dengan memilih ukuran instans yang memiliki kapasitas memori CPU dan serupa dengan lingkungan produksi Anda saat ini. Kumpulkan angka throughput dan latensi untuk beban kerja pada ukuran instans tersebut. Kemudian, naikkan skala instans ini ke ukuran yang lebih besar berikutnya. Lihat apakah angka throughput dan latensi menjadi bertambah baik. Selain itu, turunkan ukuran instans, dan lihat apakah angka latensi dan throughput tetap sama. Sasaran Anda adalah mendapatkan throughput tertinggi, dengan latensi terendah, pada instans sekecil mungkin.

> **Tip**
>
> Tentukan ukuran klaster Aurora Anda dan instans DB terkait dengan kapasitas yang cukup untuk menangani lonjakan lalu lintas yang tiba-tiba dan tidak dapat diprediksi. Untuk database mission-critical, sisakan setidaknya 20 persen kapasitas cadangan CPU dan memori.

Jalankan uji performa selama mungkin untuk mengukur performa basis data dalam kondisi siap dan stabil. Anda mungkin perlu menjalankan beban kerja selama beberapa menit atau bahkan beberapa jam sebelum mencapai kondisi stabil ini. Varians yang bermunculan di awal proses menjalankan beban kerja adalah hal yang normal. Varians tersebut terjadi karena setiap Replika Aurora menyiapkan cache-nya berdasarkan kueri `SELECT` yang ditanganinya.

Aurora beperforma paling baik untuk beban kerja transaksional yang menangani beberapa pengguna dan kueri secara bersamaan. Untuk memastikan bahwa Anda memiliki beban yang cukup untuk performa yang optimal, jalankan penilaian tolok ukur yang menggunakan multithreading, atau jalankan beberapa uji performa secara bersamaan. Ukur performa dengan ratusan atau bahkan ribuan thread konkuren klien. Simulasikan jumlah thread konkuren yang Anda perkirakan di dalam

lingkungan produksi Anda. Anda juga dapat melakukan uji stres tambahan dengan lebih banyak thread untuk mengukur skalabilitas Aurora.

## 12. Coba ketersediaan tinggi Aurora

Sebagian besar fitur utama Aurora mendukung ketersediaan yang tinggi. Fitur-fitur ini termasuk replikasi otomatis, failover otomatis, backup otomatis dengan point-in-time restore, dan kemampuan untuk menambahkan instans DB ke cluster. Keamanan dan keandalan dari fitur-fitur seperti ini penting untuk aplikasi yang bersifat vital.

Pola pikir tertentu diperlukan untuk mengevaluasi fitur tersebut. Dalam aktivitas sebelumnya, seperti pengukuran performa, Anda mengamati bagaimana sistem berfungsi ketika semuanya dapat bekerja dengan benar. Menguji ketersediaan tinggi mengharuskan Anda mempertimbangkan perilaku kasus terburuk. Anda harus mempertimbangkan potensi berbagai macam kegagalan, sekalipun kondisi kegagalan tersebut jarang terjadi. Anda dapat sengaja menimbulkan masalah untuk memastikan bahwa sistem dipulihkan dengan benar dan cepat.

> **Tip**
>
> Untuk bukti konsep, siapkan semua instance DB di cluster Aurora dengan kelas instance AWS yang sama. Dengan melakukannya, Anda akan dapat mencoba fitur ketersediaan Aurora tanpa perubahan besar pada performa dan skalabilitas saat Anda membuat instans DB offline untuk mensimulasikan kegagalan.

Kami merekomendasikan penggunaan setidaknya dua instans di setiap klaster Aurora. Instans DB dalam cluster Aurora dapat menjangkau hingga tiga Availability Zones AZs (). Tempatkan dua atau tiga instans DB pertama di AZ yang berbeda. Saat Anda mulai menggunakan cluster yang lebih besar, sebarkan instans DB Anda di semua wilayah Anda AWS . AZs Dengan melakukannya, kemampuan toleransi kesalahan akan meningkat. Bahkan jika ada masalah yang memengaruhi seluruh AZ, Aurora dapat beralih ke instans DB di AZ yang berbeda. Jika Anda menjalankan cluster dengan lebih dari tiga instance, distribusikan instans DB secara merata sebanyak yang Anda bisa di ketiganya. AZs

> **Tip**
>
> Penyimpanan untuk klaster Aurora tidak tergantung pada instans DB. Penyimpanan untuk setiap cluster Aurora selalu mencakup tiga. AZs

> Saat Anda menguji fitur ketersediaan tinggi, selalu gunakan instans DB dengan kapasitas yang sama di klaster pengujian Anda. Hal tersebut akan menghindari perubahan yang tidak dapat diprediksi dalam performa, latensi, dan sebagainya setiap kali ada satu instans DB yang mengambil alih instans DB lain.

Untuk mempelajari cara mensimulasikan kondisi kegagalan untuk menguji fitur ketersediaan tinggi, lihat Menguji Amazon Aurora My SQL menggunakan kueri injeksi kesalahan.

Sebagai bagian dari proof-of-concept latihan Anda, salah satu tujuannya adalah untuk menemukan jumlah instans DB yang ideal dan kelas instans optimal untuk instans DB tersebut. Untuk melakukannya, diperlukan keseimbangan antara ketersediaan tinggi dan performa.

Untuk Aurora, semakin banyak instans DB yang Anda miliki dalam sebuah klaster, semakin besar manfaat untuk ketersediaan tinggi. Memiliki lebih banyak instans DB juga meningkatkan skalabilitas aplikasi sarat baca. Aurora dapat mendistribusikan beberapa koneksi untuk kueri `SELECT` di beberapa Replika Aurora hanya baca.

Di sisi lain, membatasi jumlah instans DB akan mengurangi lalu lintas replikasi dari simpul primer. Lalu lintas replikasi akan menggunakan bandwidth jaringan, yang merupakan aspek lain dari performa dan skalabilitas secara keseluruhan. Jadi, untuk OLTP aplikasi intensif tulis, lebih suka memiliki jumlah instans DB besar yang lebih kecil daripada banyak instans DB kecil.

Dalam cluster Aurora yang khas, satu instance DB (instance utama) menangani semua pernyataan DDL danDML. Instans DB lainnya (Replika Aurora) hanya menangani pernyataan `SELECT`. Meskipun instans DB tidak melakukan jumlah pekerjaan yang persis sama, kami merekomendasikan penggunaan kelas instans yang sama untuk semua instans DB dalam klaster. Dengan begitu, jika terjadi kegagalan dan Aurora menjadikan salah satu instans DB hanya baca sebagai instans primer yang baru, instans primer ini akan memiliki kapasitas yang sama seperti sebelumnya.

Jika Anda perlu menggunakan instans DB dengan kapasitas berbeda dalam klaster yang sama, maka atur tingkat failover untuk instans DB. Tingkatan ini menentukan urutan promosi Replika Aurora oleh mekanisme failover. Tempatkan instans DB yang jauh lebih besar atau lebih kecil dari yang lain ke tingkat failover yang lebih rendah. Dengan melakukannya, Anda akan memastikan bahwa instans tersebut akan dipilih terakhir untuk promosi.

Latih fitur pemulihan data Aurora, seperti pemulihan otomatis, snapshot dan point-in-time pemulihan manual, dan backtracking cluster. Jika sesuai, salin snapshot ke AWS Wilayah lain dan pulihkan ke AWS Wilayah lain untuk meniru skenario DR.

Selidiki persyaratan organisasi Anda untuk tujuan waktu pemulihan (RTO), tujuan titik pemulihan (RPO), dan redundansi geografis. Sebagian besar organisasi mengelompokkan item ini ke dalam kategori pemulihan bencana. Evaluasi fitur ketersediaan tinggi Aurora yang dijelaskan dalam bagian ini dalam konteks proses pemulihan bencana Anda untuk memastikan bahwa Anda RTO dan RPO persyaratan terpenuhi.

# 13. Apa yang harus dilakukan selanjutnya

Di akhir proof-of-concept proses yang sukses, Anda mengonfirmasi bahwa Aurora adalah solusi yang cocok untuk Anda berdasarkan beban kerja yang diantisipasi. Sepanjang proses sebelumnya, Anda telah memeriksa cara kerja Aurora dalam lingkungan operasional yang realistis dan mengukurnya dengan kriteria kesuksesan Anda.

Setelah lingkungan basis data Anda aktif dan berfungsi dengan Aurora, Anda dapat melanjutkan ke langkah evaluasi yang lebih mendetail, yang mengarah ke migrasi akhir dan deployment produksi. Tergantung pada situasi Anda, langkah-langkah lain ini mungkin atau mungkin tidak termasuk dalam proof-of-concept proses. Untuk detail tentang aktivitas migrasi dan porting, lihat buku AWS putih buku pegangan migrasi Aurora.

Di langkah berikutnya, pertimbangkan konfigurasi keamanan yang relevan dengan beban kerja Anda dan yang dirancang untuk memenuhi persyaratan keamanan Anda dalam lingkungan produksi. Rencanakan kontrol apa yang akan diterapkan untuk melindungi akses ke kredensial pengguna master klaster Aurora. Tentukan peran dan tanggung jawab pengguna basis data untuk mengontrol akses ke data yang disimpan di klaster Aurora. Pertimbangkan persyaratan akses basis data untuk aplikasi, skrip, dan alat atau layanan pihak ketiga. Jelajahi AWS layanan dan fitur seperti AWS Secrets Manager dan AWS Identity and Access Management (IAM) otentikasi.

Pada tahap ini, Anda akan memahami prosedur dan praktik terbaik untuk menjalankan pengujian penilaian tolok ukur dengan Aurora. Anda mungkin perlu melakukan penyesuaian performa tambahan. Untuk detailnya, lihat Mengelola performa dan penskalaan untuk klaster DB Aurora, Peningkatan performa Amazon Aurora MySQL, Kinerja dan penskalaan untuk Amazon Aurora PostgreSQL, dan Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. Jika Anda melakukan penyesuaian tambahan, pastikan Anda memahami metrik yang Anda kumpulkan selama pembuktian konsep. Untuk langkah berikutnya, Anda mungkin perlu membuat klaster baru dengan pilihan berbeda untuk pengaturan konfigurasi, mesin basis data, dan versi basis data. Anda juga dapat membuat jenis klaster Aurora khusus untuk menyesuaikan dengan kebutuhan kasus penggunaan tertentu.

Misalnya, Anda dapat menjelajahi klaster kueri paralel Aurora untuk aplikasi transaksi/pemrosesan analitik hibrid (). HTAP Jika distribusi geografis yang luas sangat penting untuk pemulihan bencana atau untuk meminimalkan latensi, Anda dapat mengkaji basis data global Aurora. Jika beban kerja Anda bersifat intermiten atau Anda menggunakan Aurora dalam skenario pengembangan/pengujian, Anda dapat mengkaji klaster Aurora Serverless.

Klaster produksi Anda mungkin juga perlu menangani koneksi masuk dalam jumlah yang besar. Untuk mempelajari teknik-teknik tersebut, lihat AWS whitepaper Buku pegangan administrator SQL database Aurora My — Manajemen koneksi.

Jika, setelah pembuktian konsep, Anda memutuskan bahwa kasus penggunaan Anda tidak sesuai untuk Aurora, pertimbangkan layanan AWS lainnya berikut ini:

- Untuk kasus penggunaan analitik murni, beban kerja mendapat manfaat dari format penyimpanan kolumnar dan fitur lain yang lebih cocok untuk beban kerja. OLAP AWS layanan yang menangani kasus penggunaan tersebut meliputi:
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- Sebagian besar beban kerja akan mendapatkan keuntungan dari kombinasi Aurora dengan satu atau beberapa layanan ini. Anda dapat memindahkan data di antara layanan-layanan ini dengan menggunakan hal berikut ini:
  - AWS Glue
  - AWS DMS
  - Mengimpor dari Amazon S3, sebagaimana dijelaskan di Panduan Pengguna Amazon Aurora
  - Mengekspor ke Amazon S3, sebagaimana dijelaskan di Panduan Pengguna Amazon Aurora
  - Banyak ETL alat populer lainnya

# Keamanan dalam Amazon Aurora

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menggambarkan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara berkala menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program kepatuhan AWS. Untuk mempelajari tentang program kepatuhan yang berlaku untuk Aurora (Aurora), AWS lihat layanan dalam cakupan berdasarkan program kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, termasuk sensitivitas data, persyaratan perusahaan, serta hukum dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Aurora. Topik berikut menunjukkan cara mengonfigurasi Aurora untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan AWS layanan lain yang membantu Anda memantau dan mengamankan sumber daya Amazon Aurora Anda.

Metode yang Anda gunakan untuk mengelola akses bergantung pada jenis tugas apa yang perlu dilakukan pengguna dengan Aurora:

- Jalankan cluster DB Anda di cloud pribadi virtual (VPC) berdasarkan layanan Amazon VPC untuk kontrol akses jaringan sebaik mungkin. Untuk informasi selengkapnya tentang membuat cluster DB di VPC, lihat. Amazon VPC dan Aurora
- Gunakan kebijakan AWS Identity and Access Management (IAM) untuk menetapkan izin yang menentukan siapa yang diizinkan mengelola sumber daya Amazon RDS Aurora. Misalnya, Anda dapat menggunakan IAM untuk menentukan siapa yang diizinkan membuat, mendeskripsikan, memodifikasi, dan menghapus cluster DB, menandai sumber daya, atau memodifikasi grup keamanan.

Untuk meninjau contoh kebijakan IAM, lihatContoh kebijakan berbasis identitas untuk  Amazon Aurora.

- Gunakan grup keamanan untuk mengontrol alamat IP atau EC2 instans Amazon yang dapat terhubung ke database Anda di cluster DB. Saat pertama kali membuat cluster DB, firewall-nya mencegah akses database apa pun kecuali melalui aturan yang ditentukan oleh grup keamanan terkait.
- Gunakan koneksi Secure Socket Layer (SSL) atau Transport Layer Security (TLS) dengan cluster DB yang menjalankan Aurora MySQL atau Aurora PostgreSQL. Untuk informasi selengkapnya tentang penggunaan SSL/TLS dengan cluster DB, lihat. Menggunakan SSL/TLS untuk mengenkripsi koneksi ke
- Gunakan enkripsi Amazon untuk mengamankan instans DB cluster dan snapshot DB saat istirahat. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
- Gunakan fitur keamanan mesin DB Anda untuk mengontrol siapa yang dapat masuk ke database pada cluster DB. Fitur ini berfungsi seolah-olah basis data berada di jaringan lokal Anda.

  Untuk informasi tentang keamanan dengan Aurora MySQL, lihat. Keamanan dengan Amazon Aurora MySQL Untuk informasi tentang keamanan dengan Aurora PostgreSQL, lihat. Keamanan dengan Amazon Aurora PostgreSQL

Aurora adalah bagian dari layanan basis data terkelola Amazon Relational Database Service (Amazon RDS). Amazon RDS adalah layanan web yang memudahkan penyiapan, pengoperasian, dan penskalaan basis data relasional di cloud. Jika Anda belum memahami tentang Amazon RDS, lihat Panduan pengguna Amazon RDS.

Aurora mencakup subsistem penyimpanan performa tinggi. Mesin basis data-nya yang kompatibel dengan MySQL dan PostgreSQL disesuaikan untuk memanfaatkan penyimpanan terdistribusi cepat tersebut. Aurora juga mengotomatiskan dan membakukan pengelompokan dan replikasi basis data, yang biasanya termasuk dalam aspek konfigurasi dan administrasi basis data yang paling menantang.

Untuk Amazon RDS dan Aurora, Anda dapat mengakses RDS API secara terprogram, dan Anda dapat menggunakannya untuk mengakses RDS API secara AWS CLI interaktif. Beberapa pengoperasian API RDS dan perintah AWS CLI berlaku untuk Amazon RDS dan Aurora, sementara yang lainnya berlaku untuk Amazon RDS atau Aurora. Untuk informasi selengkapnya tentang pengoperasian API RDS, lihat Referensi API Amazon RDS. Untuk informasi selengkapnya tentang AWS CLI, lihat AWS Command Line Interface referensi untuk Amazon RDS.

> **Note**
>
> Anda hanya perlu mengonfigurasikan keamanan untuk kasus penggunaan Anda. Anda tidak perlu mengonfigurasi akses keamanan untuk proses yang dikelola Amazon Aurora. Ini termasuk membuat cadangan, failover otomatis, dan proses lainnya.

Untuk informasi selengkapnya tentang mengelola akses ke sumber daya Amazon Aurora dan database Anda di cluster instans , lihat topik berikut.

Topik


- Autentikasi basis data dengan Amazon Aurora
- Manajemen kata sandi dengan dan AWS Secrets Manager
- Perlindungan data di Amazon RDS
- Manajemen identitas dan akses untuk Aurora
- Pencatatan log dan pemantauan di Amazon Aurora
- Validasi kepatuhan untuk Aurora
- Ketangguhan di Amazon Aurora
- Keamanan infrastruktur di Aurora
- API Amazon RDS dan titik akhir VPC antarmuka (AWS PrivateLink)
- Praktik terbaik keamanan untuk Amazon Aurora
- Mengontrol akses dengan grup keamanan
- Hak akses akun pengguna master
- Menggunakan peran terkait layanan untuk Amazon Aurora
- Amazon VPC dan Aurora


# Autentikasi basis data dengan Amazon Aurora

Amazon Aurora mendukung beberapa cara untuk mengautentikasi pengguna database.

Autentikasi kata sandi tersedia secara default untuk semua klaster basis data. Untuk Aurora MySQL dan Aurora PostgreSQL, Anda juga dapat menambahkan salah satu atau kedua autentikasi basis data IAM dan autentikasi Kerberos bagi klaster basis data yang sama.

Autentikasi kata sandi, Kerberos, dan basis data IAM menggunakan metode autentikasi yang berbeda ke basis data. Oleh karena itu, pengguna tertentu dapat masuk ke basis data dengan menggunakan hanya satu metode autentikasi.

Untuk PostgreSQL, gunakan hanya salah satu dari setelah peran berikut untuk pengguna basis data tertentu:

- Untuk menggunakan autentikasi basis data IAM, tetapkan peran `rds_iam` untuk pengguna.
- Untuk menggunakan autentikasi Kerberos, tetapkan peran `rds_ad` untuk pengguna.
- Untuk menggunakan autentikasi kata sandi, jangan tetapkan peran `rds_iam` atau `rds_ad` untuk pengguna.

Jangan tetapkan kedua peran `rds_iam` dan `rds_ad` untuk pengguna basis data PostgreSQL baik secara langsung maupun tidak langsung dengan akses pemberian bersarang. Jika peran `rds_iam` ditambahkan ke pengguna master, autentikasi IAM diutamakan atas autentikasi kata sandi sehingga pengguna master harus masuk sebagai pengguna IAM.

> **Important**
>
> Sebaiknya jangan gunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik penggunaan pengguna basis data yang dibuat dengan privilese minimal yang diperlukan untuk aplikasi Anda.

Topik


- Autentikasi kata sandi
- Autentikasi basis data IAM
- Autentikasi Kerberos


## Autentikasi kata sandi

Dengan autentikasi kata sandi, basis data Anda melakukan semua administrasi akun pengguna. Anda membuat pengguna dengan pernyataan SQL seperti `CREATE USER`, dengan klausa yang tepat yang diperlukan oleh mesin basis data untuk menentukan kata sandi. Misalnya, di MySQL pernyataannya adalah, sementara di PostgreSQL, pernyataannya `CREATE USER` *`name`* `IDENTIFIED BY` *`password`* adalah. `CREATE USER` *`name`* `WITH PASSWORD` *`password`*

Dengan autentikasi kata sandi, basis data Anda mengendalikan dan mengautentikasi akun pengguna. Jika mesin basis data memiliki fitur pengelolaan kata sandi kuat, mesin itu dapat meningkatkan keamanan. Autentikasi basis data mungkin lebih mudah dikelola dengan menggunakan autentikasi kata sandi apabila komunitas pengguna Anda kecil. Karena kata sandi teks yang jelas dihasilkan dalam kasus ini, mengintegrasikan dengan AWS Secrets Manager dapat meningkatkan keamanan.

Untuk informasi tentang menggunakan Secrets Manager dengan Amazon Aurora, lihat Membuat rahasia dasar dan Memutar rahasia untuk database Amazon RDS yang didukung di Panduan Pengguna.AWS Secrets Manager  Lihat informasi tentang cara mengambil rahasia secara terprogram pada aplikasi kustom Anda di Mengambil nilai rahasia dalam Panduan Pengguna AWS Secrets Manager .

## Autentikasi basis data IAM

Anda dapat mengautentikasi ke cluster DB Anda menggunakan otentikasi database AWS Identity and Access Management (IAM). Dengan metode otentikasi ini, Anda tidak perlu menggunakan kata sandi saat terhubung ke cluster DB. Sebagai gantinya, Anda menggunakan token autentikasi.

Untuk informasi selengkapnya tentang autentikasi database IAM, termasuk informasi tentang ketersediaan untuk mesin DB tertentu, lihat. Autentikasi basis data IAM

## Autentikasi Kerberos

Amazon Aurora Amazon mendukung otentikasi eksternal pengguna database menggunakan Kerberos dan Microsoft Active Directory. Kerberos adalah protokol autentikasi jaringan yang menggunakan tiket dan kriptografi kunci simetris untuk menghilangkan kebutuhan mengirim kata sandi melalui jaringan. Kerberos telah tertanam ke dalam Active Directory dan dirancang untuk mengautentikasi pengguna ke sumber daya jaringan, seperti basis data.

Dukungan Amazon Aurora Amazon untuk Kerberos dan Active Directory memberikan manfaat sistem masuk tunggal dan otentikasi terpusat dari pengguna database. Anda dapat menyimpan kredensial pengguna Anda di Active Directory.

Untuk menggunakan kredensyal dari Active Directory yang dikelola sendiri, Anda perlu menyiapkan hubungan kepercayaan ke for AWS Directory Service Microsoft Active Directory tempat cluster DB digabungkan.

satu arah dan dua arah dengan otentikasi seluruh hutan atau otentikasi selektif.

Dalam beberapa skenario, Anda dapat mengonfigurasi otentikasi Kerberos melalui hubungan kepercayaan eksternal. Ini memerlukan Active Directory yang dikelola sendiri untuk memiliki pengaturan tambahan. Ini termasuk tetapi tidak terbatas pada Perintah Pencarian Hutan Kerberos.

Aurora mendukung autentikasi Kerberos untuk klaster basis data Aurora MySQL dan Aurora PostgreSQL. Untuk informasi lebih lanjut, lihat Menggunakan otentikasi Kerberos untuk Aurora My SQL danMenggunakan autentikasi Kerberos dengan Aurora PostgreSQL.

# Manajemen kata sandi dengan dan AWS Secrets Manager

Amazon Aurora terintegrasi dengan Secrets Manager untuk mengelola kata sandi pengguna utama untuk klaster .

Topik


- Ketersediaan wilayah dan versi
- Batasan untuk integrasi Secrets Manager dengan Amazon Aurora
- Ikhtisar mengelola kata sandi pengguna master dengan AWS Secrets Manager
- Manfaat mengelola kata sandi pengguna utama dengan Secrets Manager
- Izin yang diperlukan untuk integrasi Secrets Manager
- Menegakkan manajemen dari kata sandi pengguna utama di AWS Secrets Manager
- Mengelola kata sandi pengguna utama untuk klaster DB dengan Secrets Manager
- Merotasi rahasia kata sandi pengguna utama untuk klaster DB
- Melihat detail tentang rahasia untuk klaster DB


## Ketersediaan wilayah dan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin basis data dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang ketersediaan versi dan Wilayah dengan integrasi Secrets Manager dengan Amazon Aurora, lihat Daerah yang Didukung dan engine Aurora DB untuk integrasi Secrets Manager.

## Batasan untuk integrasi Secrets Manager dengan Amazon Aurora

Mengelola kata sandi pengguna utama dengan Secrets Manager tidak didukung untuk fitur berikut:

- Penerapan Amazon RDS Blue/Green
- Klaster DB yang merupakan bagian dari basis data global Aurora
- Klaster DB Aurora Serverless v1
- Aurora baca replika

# Ikhtisar mengelola kata sandi pengguna master dengan AWS Secrets Manager

Dengan AWS Secrets Manager, Anda dapat mengganti kredensi hard-code dalam kode Anda, termasuk kata sandi database, dengan panggilan API ke Secrets Manager untuk mengambil rahasia secara terprogram. Untuk informasi selengkapnya tentang Secrets Manager, lihat Panduan Pengguna AWS Secrets Manager.

Ketika Anda menyimpan rahasia database di Secrets Manager, Anda akan Akun AWS dikenakan biaya. Untuk informasi tentang harga, lihat AWS Secrets Manager Harga.

Anda dapat menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager untuk klaster DB Amazon Aurora saat Anda melakukan salah satu operasi berikut:

- Buat cluster DB
- Memodifikasi cluster DB
- Kembalikan cluster DB dari Amazon S3 (hanya Aurora MySQL)

Saat Anda menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager, Aurora menghasilkan kata sandi dan menyimpannya dalam Secrets Manager. Anda dapat berinteraksi langsung dengan rahasia untuk mengambil kredensial untuk pengguna utama. Anda juga dapat menentukan kunci yang dikelola pelanggan untuk mengenkripsi rahasia, atau menggunakan kunci KMS default yang disediakan oleh Secrets Manager.

Secara default, Aurora mengelola pengaturan untuk rahasia dan merotasi rahasia setiap tujuh hari. Anda dapat mengubah beberapa pengaturan, seperti jadwal rotasi. Jika Anda menghapus klaster DB yang mengelola rahasia di Secrets Manager, rahasia dan metadata terkaitnya juga akan dihapus.

Untuk terhubung ke klaster DB dengan kredensial dalam rahasia, Anda dapat mengambil rahasia dari Secrets Manager. Untuk informasi selengkapnya, lihat Mengambil rahasia dari AWS Secrets Manager dan Connect ke database SQL dengan kredensi dalam AWS Secrets Manager rahasia di Panduan Pengguna.AWS Secrets Manager

## Manfaat mengelola kata sandi pengguna utama dengan Secrets Manager

Mengelola kata sandi pengguna utama Aurora dengan Secrets Manager memberikan manfaat berikut:

- Aurora secara otomatis menghasilkan kredensial basis data.

- Aurora secara otomatis menyimpan dan mengelola kredensil database di. AWS Secrets Manager
- Aurora merotasi kredensial basis data secara teratur, tanpa mewajibkan perubahan aplikasi.
- Secrets Manager mengamankan kredensial basis data dari akses manusia dan tampilan teks biasa.
- Secrets Manager memungkinkan pengambilan kredensial basis data rahasia untuk koneksi basis data.
- Secrets Manager memungkinkan kontrol akses terperinci ke kredensial basis data dalam rahasia menggunakan IAM.
- Secara opsional, Anda dapat memisahkan enkripsi basis data dari enkripsi kredensial dengan kunci KMS lainnya.
- Anda dapat menghilangkan rotasi dan manajemen manual kredensial basis data.
- Anda dapat memantau kredensi database dengan mudah dengan dan AWS CloudTrail Amazon. CloudWatch

Untuk informasi selengkapnya tentang manfaat Secrets Manager, lihat Panduan Pengguna AWS Secrets Manager.

## Izin yang diperlukan untuk integrasi Secrets Manager

Pengguna harus memiliki izin yang diperlukan untuk melakukan operasi yang terkait dengan integrasi Secrets Manager. Anda dapat membuat kebijakan IAM yang memberikan izin untuk melakukan operasi API tertentu pada sumber daya spesifik yang diperlukan. Kemudian, Anda dapat melampirkan kebijakan tersebut ke peran atau kumpulan izin IAM yang memerlukan izin tersebut. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Aurora.

Untuk membuat, memodifikasi, atau memulihkan operasi, pengguna yang menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager harus memiliki izin untuk melakukan operasi berikut:

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

`kms:DescribeKey`Izin diperlukan untuk mengakses kunci yang dikelola pelanggan Anda untuk `MasterUserSecretKmsKeyId` dan untuk menjelaskan. `aws/secretsmanager`

Untuk membuat, memodifikasi, atau memulihkan operasi, pengguna yang menentukan kunci yang dikelola pelanggan untuk mengenkripsi rahasia dalam Secrets Manager harus memiliki izin untuk melakukan operasi berikut:

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

Untuk mengubah operasi, pengguna yang merotasi kata sandi pengguna utama dalam Secrets Manager harus memiliki izin untuk melakukan operasi berikut:

- `secretsmanager:RotateSecret`

## Menegakkan manajemen dari kata sandi pengguna utama di AWS Secrets Manager

Anda dapat menggunakan kunci kondisi IAM untuk menerapkan manajemen Aurora kata sandi pengguna utama di AWS Secrets Manager. Kebijakan berikut tidak mengizinkan pengguna untuk membuat atau memulihkan instans DB atau kluster DB atau kecuali kata sandi pengguna utama dikelola oleh RDS di Secrets Manager.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
```

```
}
```

> **Note**
>
> Kebijakan ini memberlakukan manajemen kata sandi pada AWS Secrets Manager saat pembuatan. Namun, Anda masih dapat menonaktifkan integrasi Secrets Manager dan mengatur kata sandi utama secara manual dengan mengubah klaster.
> Untuk mencegahnya, sertakan `rds:ModifyDBInstance`, `rds:ModifyDBCluster` dalam blok tindakan kebijakan. Perhatikan bahwa tindakan ini akan mencegah pengguna menerapkan perubahan lebih lanjut pada klaster yang ada yang Secrets Manager-nya tidak diaktifkan.

Untuk informasi lebih lanjut tentang penggunaan kunci kondisi dalam kebijakan IAM, lihat Kunci kondisi kebijakan untuk Aurora dan Contoh Kebijakan: Menggunakan kunci kondisi.

## Mengelola kata sandi pengguna utama untuk klaster DB dengan Secrets Manager

Anda dapat mengonfigurasi manajemen Aurora kata sandi pengguna utama di Secrets Manager saat Anda melakukan tindakan berikut:

- Membuat klaster DB Amazon Aurora
- Memodifikasi klaster DB Amazon Aurora
- Memigrasikan data dari basis data MySQL eksternal ke klaster DB Amazon Aurora MySQL

Anda dapat menggunakan konsol RDS, API AWS CLI, atau RDS untuk melakukan tindakan ini.

Konsol

Ikuti petunjuk untuk membuat atau mengubah klaster DB dengan konsol RDS:

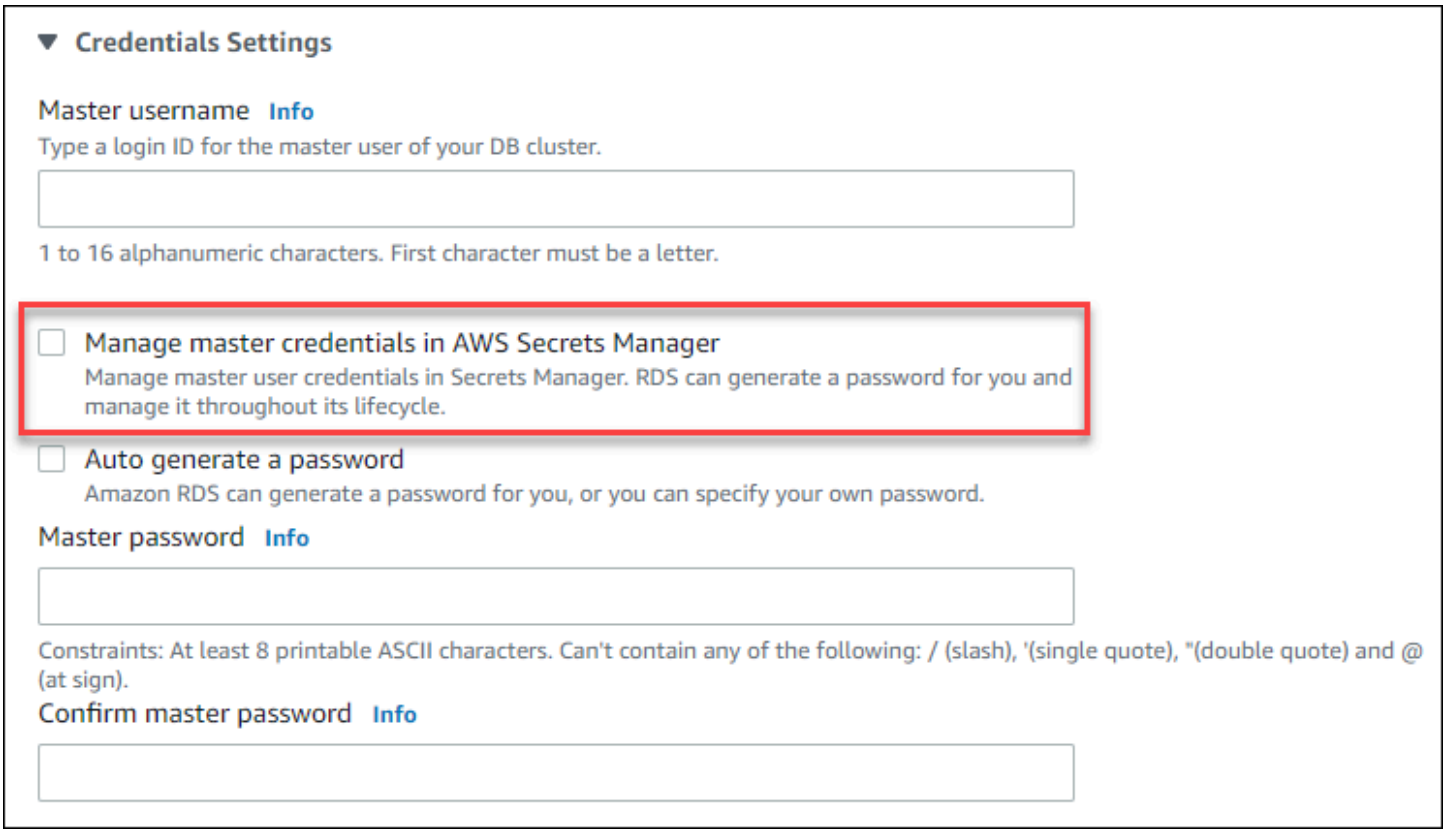
- Membuat klaster DB
- Memodifikasi instans DB dalam klaster DB

  Di konsol RDS, Anda dapat mengubah instans DB apa pun untuk menentukan pengaturan manajemen kata sandi pengguna utama untuk seluruh klaster DB.

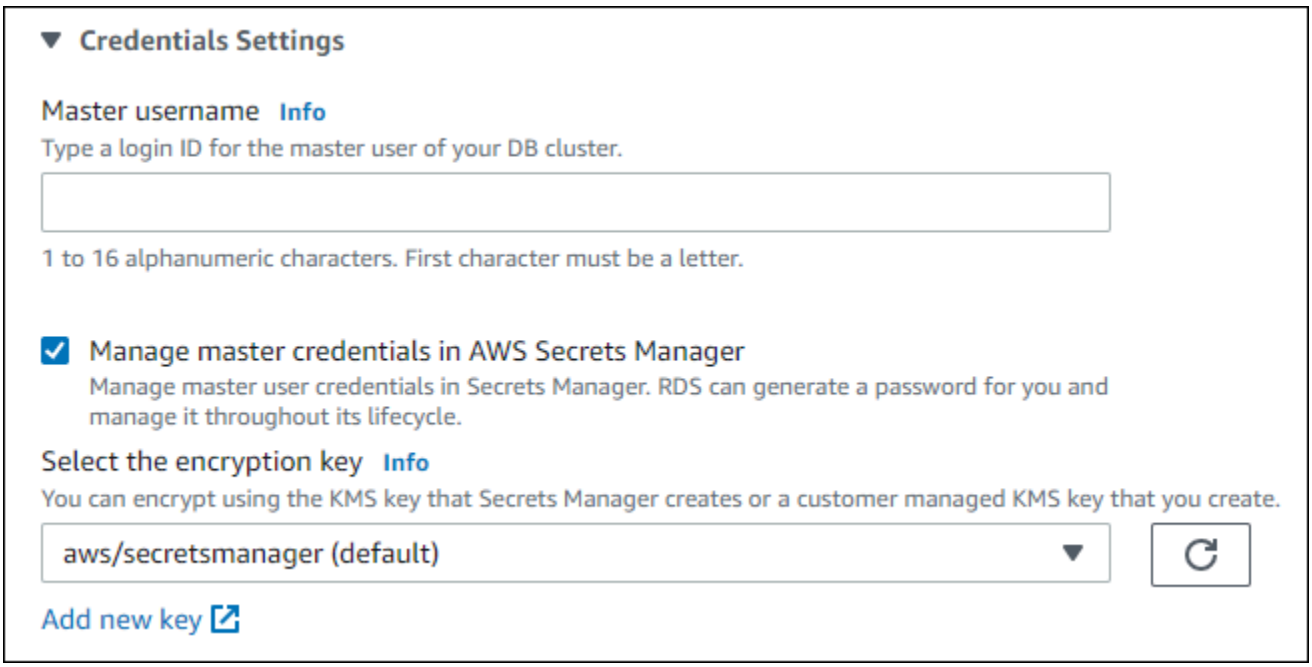
- Memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3

Saat menggunakan konsol RDS untuk melakukan salah satu operasi ini, Anda dapat menentukan bahwa kata sandi pengguna utama dikelola oleh Aurora di Secrets Manager. Untuk melakukannya saat membuat atau memulihkan klaster DB, pilih Kelola kredensial utama di AWS Secrets Manager dalam Pengaturan kredensial. Saat Anda mengubah klaster DB, pilih Kelola kredensial utama di AWS Secrets Manager dalam Pengaturan.

Gambar berikut adalah contoh pengaturan Kelola kredensial utama di AWS Secrets Manager saat Anda membuat atau memulihkan klaster DB.




Jika Anda memilih opsi ini, Aurora akan menghasilkan kata sandi pengguna utama dan mengelolanya sepanjang siklus pemakaiannya di Secrets Manager.




Anda dapat memilih untuk mengenkripsi rahasia dengan kunci KMS yang disediakan Secrets Manager atau dengan kunci yang dikelola pelanggan yang Anda buat. Setelah Aurora mengelola kredensial basis data untuk klaster DB, Anda tidak dapat mengubah kunci KMS yang digunakan untuk mengenkripsi rahasia.

Anda dapat memilih pengaturan lain untuk memenuhi kebutuhan Anda.

Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda membuat klaster DB, lihat Pengaturan untuk klaster Aurora DB. Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda mengubah klaster DB, lihat Pengaturan untuk Amazon Aurora.

AWS CLI

Untuk menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager, tentukan opsi `--manage-master-user-password` di salah satu perintah berikut:

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

Jika Anda menentukan opsi `--manage-master-user-password` dalam perintah ini, Aurora akan menghasilkan kata sandi pengguna utama dan mengelolanya sepanjang siklus pemakaiannya di Secrets Manager.

Untuk mengenkripsi rahasia, Anda dapat menentukan kunci yang dikelola pelanggan atau menggunakan kunci KMS default yang disediakan oleh Secrets Manager. Gunakan opsi `--master-user-secret-kms-key-id` untuk menentukan kunci yang dikelola pelanggan. Pengidentifikasi kunci AWS KMS adalah kunci ARN, ID kunci, alias ARN, atau nama alias untuk kunci KMS. Untuk menggunakan kunci KMS yang berbeda Akun AWS, tentukan kunci ARN atau alias ARN. Setelah Aurora mengelola kredensial basis data untuk klaster DB, Anda tidak dapat mengubah kunci KMS yang digunakan untuk mengenkripsi rahasia.

Anda dapat memilih pengaturan lain untuk memenuhi kebutuhan Anda.

Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda membuat klaster DB, lihat Pengaturan untuk klaster Aurora DB. Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda mengubah klaster DB, lihat Pengaturan untuk Amazon Aurora.

Contoh ini membuat klaster DB dan menentukan bahwa Aurora mengelola kata sandi di Secrets Manager. Rahasianya dienkripsi menggunakan kunci KMS yang disediakan oleh Secrets Manager.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
     --db-cluster-identifier sample-cluster \
     --engine aurora-mysql \
     --engine-version 8.0 \
     --master-username admin \
     --manage-master-user-password
```

Untuk Windows:

```
aws rds create-db-cluster ^
     --db-cluster-identifier sample-cluster ^
     --engine aurora-mysql ^
     --engine-version 8.0 ^
     --master-username admin ^
     --manage-master-user-password
```

RDS API

Untuk menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager, atur parameter `ManageMasterUserPassword` ke `true` di salah satu operasi berikut:

- Buat DBCluster
- Memodifikasi DBCluster
- Kembalikan DBCluster dari3

Jika Anda mengatur parameter `ManageMasterUserPassword` ke `true` di salah satu operasi ini, Aurora akan menghasilkan kata sandi pengguna utama dan mengelolanya sepanjang siklus pemakaiannya di Secrets Manager.

Untuk mengenkripsi rahasia, Anda dapat menentukan kunci yang dikelola pelanggan atau menggunakan kunci KMS default yang disediakan oleh Secrets Manager. Gunakan parameter `MasterUserSecretKmsKeyId` untuk menentukan kunci yang dikelola pelanggan. Pengidentifikasi kunci AWS KMS adalah kunci ARN, ID kunci, alias ARN, atau nama alias untuk kunci KMS. Untuk menggunakan kunci KMS di Akun AWS yang berbeda, tentukan ARN kunci atau ARN alias. Setelah Aurora mengelola kredensial basis data untuk klaster DB, Anda tidak dapat mengubah kunci KMS yang digunakan untuk mengenkripsi rahasia.

## Merotasi rahasia kata sandi pengguna utama untuk klaster DB

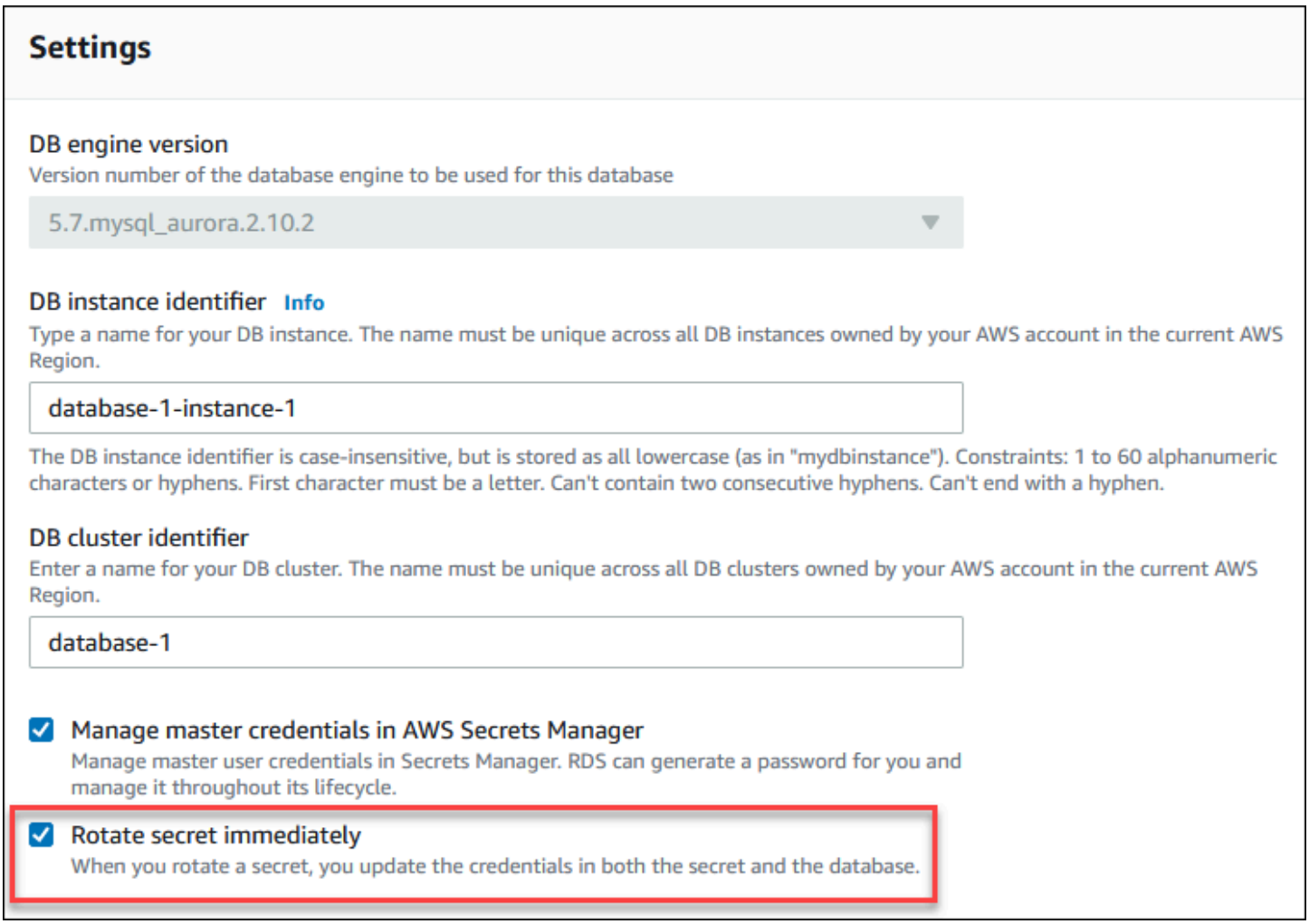
Ketika Aurora merotasi rahasia kata sandi pengguna utama, Secrets Manager akan menghasilkan versi rahasia baru untuk rahasia yang sudah ada. Rahasia versi baru berisi kata sandi pengguna utama baru. Aurora mengubah kata sandi pengguna utama untuk klaster DB agar sesuai dengan kata sandi versi rahasia baru.

Anda dapat segera merotasi rahasia, alih-alih menunggu rotasi yang dijadwalkan. Untuk merotasi rahasia kata sandi pengguna utama di Secrets Manager, ubah klaster DB . Untuk informasi tentang cara mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Anda dapat memutar rahasia kata sandi pengguna master segera dengan konsol RDS, API AWS CLI, atau RDS. Kata sandi baru selalu sepanjang 28 karakter dan berisi setidaknya satu karakter huruf besar dan kecil, satu angka, dan satu tanda baca.

Konsol

Untuk merotasi rahasia kata sandi pengguna utama menggunakan konsol RDS, ubah klaster DB dan pilih Rotasi rahasia secara langsung di Pengaturan.




Ikuti petunjuk untuk mengubah klaster DB dengan konsol RDS di Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Anda harus memilih Terapkan langsung di halaman konfirmasi.

AWS CLI

Untuk memutar rahasia kata sandi pengguna master menggunakan AWS CLI, gunakan modify-db-clusterperintah dan tentukan `--rotate-master-user-password` opsi. Anda harus menentukan opsi `--apply-immediately` saat merotasi kata sandi utama.

Contoh ini merotasi rahasia kata sandi pengguna utama.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
```

```
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

API RDS

Anda dapat memutar rahasia kata sandi pengguna master menggunakan DBCluster operasi Modify dan mengatur `RotateMasterUserPassword` parameternya`true`. Anda harus mengatur parameter `ApplyImmediately` ke `true` saat merotasi kata sandi utama.

## Melihat detail tentang rahasia untuk klaster DB

Anda dapat mengambil rahasia Anda menggunakan console (https://console.aws.amazon.com/secretsmanager/) atau perintah AWS CLI (get-secret-valueSecrets Manager).

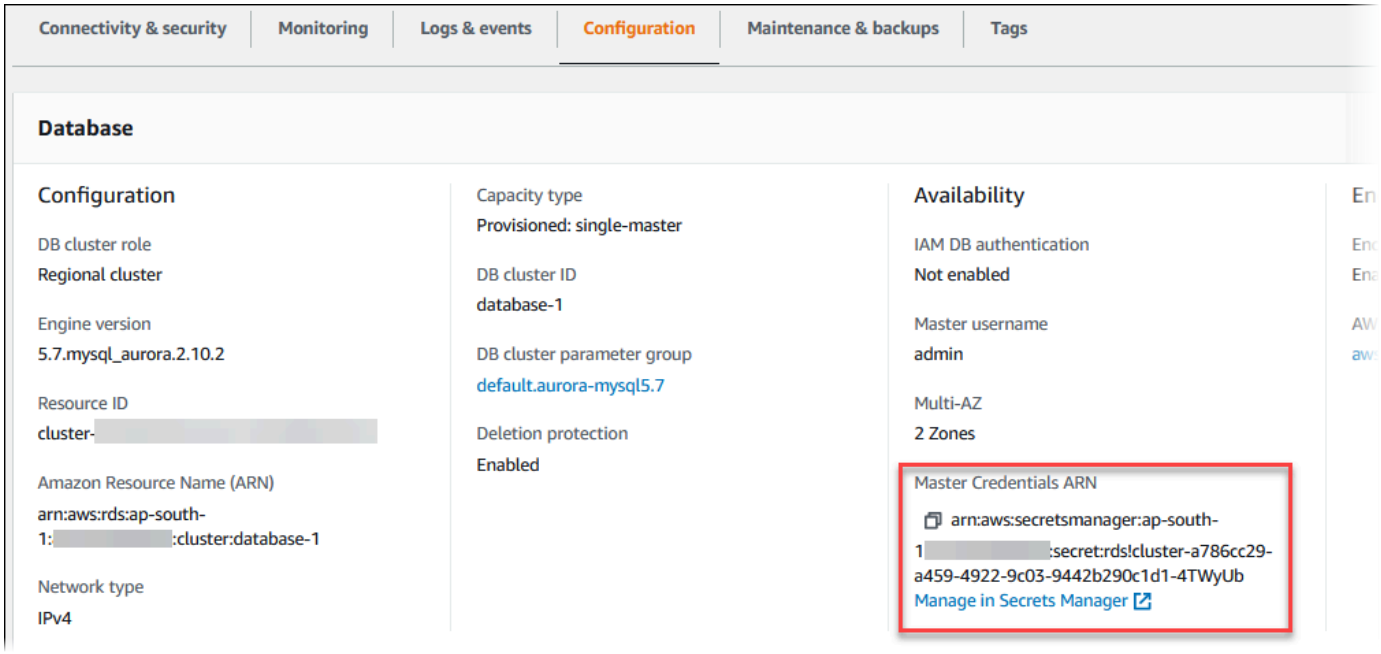
Anda dapat menemukan Amazon Resource Name (ARN) dari rahasia yang dikelola oleh Aurora di Secrets Manager dengan konsol RDS AWS CLI,, atau RDS API.

Konsol

Untuk melihat detail tentang rahasia yang dikelola oleh Aurora di Secrets Manager

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama klaster DB untuk menampilkan detailnya.
4. Pilih tab Konfigurasi.

   Di ARN Kredensial Utama, Anda dapat melihat ARN rahasia.




Anda dapat mengikuti tautan Mengelola di Secrets Manager untuk melihat dan mengelola rahasia di konsol Secrets Manager.

AWS CLI

Anda dapat menggunakan AWS CLI describe-db-clustersperintah RDS untuk menemukan informasi berikut tentang rahasia yang dikelola oleh di Secrets Manager:

- `SecretArn` – ARN rahasia
- `SecretStatus` – Status rahasia

  Kemungkinan nilai statusnya meliputi:

  - `creating` – Rahasia sedang dibuat.
  - `active` – Rahasia tersedia untuk penggunaan normal dan rotasi.
  - `rotating` – Rahasia sedang dirotasi.
  - `impaired` – Rahasia dapat digunakan untuk mengakses kredensial basis data, tetapi tidak dapat dirotasi. Rahasia mungkin memiliki status ini jika, misalnya, izin diubah sehingga RDS tidak dapat lagi mengakses rahasia atau kunci KMS untuk rahasia tersebut.

    Ketika rahasia memiliki status ini, Anda dapat memperbaiki kondisi yang menyebabkan status tersebut. Jika Anda memperbaiki kondisi yang menyebabkan status, status tersebut tetap `impaired` hingga rotasi berikutnya. Sebagai alternatif, Anda dapat mengubah klaster DB untuk

menonaktifkan manajemen otomatis kredensial basis data, dan kemudian mengubah klaster DB lagi untuk mengaktifkan manajemen otomatis kredensial basis data. Untuk memodifikasi cluster DB, gunakan `--manage-master-user-password` opsi dalam modify-db-clusterperintah.

- `KmsKeyId` – ARN kunci KMS yang digunakan untuk mengenkripsi rahasia

Tentukan opsi `--db-cluster-identifier` untuk menampilkan output untuk klaster DB tertentu. Contoh ini menunjukkan output untuk rahasia yang digunakan oleh klaster DB.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

Contoh berikut menunjukkan output untuk rahasia:

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

Ketika Anda memiliki ARN rahasia, Anda dapat melihat detail tentang rahasia menggunakan perintah get-secret-valueSecrets Manager CLI.

Contoh ini menunjukkan detail untuk rahasia dalam output sampel sebelumnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

Untuk Windows:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

API RDS

Anda dapat melihat kunci ARN, status, dan KMS untuk rahasia yang dikelola oleh RDS di Secrets Manager menggunakan operasi DBClusters Deskripsikan RDS dan mengatur `DBClusterIdentifier` parameter ke pengidentifikasi cluster DB. Detail tentang rahasia disertakan dalam output.

Ketika Anda memiliki ARN rahasia, Anda dapat melihat detail tentang rahasia menggunakan operasi GetSecretValueSecrets Manager.

# Perlindungan data di Amazon RDS

Model tanggung jawab AWS bersama model berlaku untuk perlindungan data di Amazon Relational Database Service. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Lihat informasi yang lebih lengkap tentang privasi data dalam Pertanyaan Umum Privasi Data. Lihat informasi tentang perlindungan data di Eropa di pos blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensyal dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan sumber daya. AWS Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat Bekerja dengan CloudTrail jejak di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.

- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-3.

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan Amazon RDS atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDKs. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Topik


- Melindungi data dengan menggunakan enkripsi
- Privasi lalu lintas antarjaringan


## Melindungi data dengan menggunakan enkripsi

Anda dapat mengaktifkan enkripsi untuk sumber daya basis data. Anda juga dapat mengenkripsi koneksi ke cluster DB.

Topik


- Mengenkripsi sumber daya Amazon Aurora
- AWS KMS key manajemen
- Menggunakan SSL/TLS untuk mengenkripsi koneksi ke
- Merotasi sertifikat SSL/TLS


### Mengenkripsi sumber daya Amazon Aurora

Amazon Aurora dapat mengenkripsi klaster DB Amazon Aurora Anda. Data yang dienkripsi saat diam termasuk penyimpanan dasar untuk klaster DB, pencadangan otomatisnya, replika baca, dan snapshot.

Klaster DB terenkripsi Amazon Aurora menggunakan algoritma AES-256 standar industri untuk mengenkripsi data Anda di server yang menjadi host klaster DB Amazon Aurora Anda. Setelah

data Anda dienkripsi, Amazon Aurora menangani autentikasi akses dan dekripsi data Anda secara transparan dengan dampak minimal terhadap performa. Anda tidak perlu memodifikasi aplikasi klien basis data untuk menggunakan enkripsi.

> **Note**
>
> Untuk cluster DB terenkripsi dan tidak terenkripsi, data yang sedang transit antara sumber dan replika baca dienkripsi, bahkan saat mereplikasi di seluruh Wilayah. AWS

Topik

- Ikhtisar enkripsi sumber daya Amazon Aurora
- Mengenkripsi klaster DB Amazon Aurora
- Menentukan apakah enkripsi untuk klaster DB diaktifkan
- Ketersediaan enkripsi Amazon Aurora
- Enkripsi dalam transit
- Batasan klaster DB terenkripsi Amazon Aurora

## Ikhtisar enkripsi sumber daya Amazon Aurora

Klaster DB terenkripsi Amazon Aurora menyediakan lapisan perlindungan data tambahan dengan mengamankan data Anda dari akses tidak sah ke penyimpanan dasar. Anda dapat menggunakan enkripsi Amazon Aurora untuk meningkatkan perlindungan data aplikasi Anda yang di-deploy di cloud, dan untuk memenuhi persyaratan kepatuhan enkripsi diam. Untuk klaster DB terenkripsi Amazon Aurora, semua instans DB, log, cadangan, dan snapshot dienkripsi. Untuk informasi selengkapnya tentang ketersediaan dan keterbatasan enkripsi, lihat Ketersediaan enkripsi Amazon Aurora danBatasan klaster DB terenkripsi Amazon Aurora.

Amazon Aurora menggunakan AWS Key Management Service kunci untuk mengenkripsi sumber daya ini. AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan sangat tersedia untuk menyediakan sistem manajemen kunci yang diskalakan untuk cloud. Anda dapat menggunakan Kunci yang dikelola AWS, atau Anda dapat membuat kunci yang dikelola pelanggan.

Saat membuat klaster DB terenkripsi, Anda dapat memilih kunci yang dikelola pelanggan atau Kunci yang dikelola AWS untuk Amazon Aurora untuk mengenkripsi klaster DB. Jika Anda tidak menentukan pengenal kunci untuk kunci yang dikelola pelanggan, Amazon Aurora menggunakan Kunci yang dikelola AWS untuk klaster DB baru Anda. Amazon Aurora membuat untuk Amazon

Aurora Kunci yang dikelola AWS untuk akun Anda. AWS AWS Akun Anda memiliki perbedaan Kunci yang dikelola AWS untuk Amazon Aurora untuk setiap AWS Wilayah.

Untuk mengelola kunci terkelola pelanggan yang digunakan untuk mengenkripsi dan mendekripsi sumber daya Amazon Aurora, Anda menggunakan ().AWS Key Management ServiceAWS KMS

Dengan menggunakan AWS KMS, Anda dapat membuat kunci terkelola pelanggan dan menentukan kebijakan untuk mengontrol penggunaan kunci yang dikelola pelanggan ini. AWS KMS mendukung CloudTrail, sehingga Anda dapat mengaudit penggunaan kunci KMS untuk memverifikasi bahwa kunci yang dikelola pelanggan digunakan dengan tepat. Anda dapat menggunakan kunci yang dikelola pelanggan dengan Amazon Aurora dan AWS layanan yang didukung seperti Amazon S3, Amazon EBS, dan Amazon Redshift. Untuk daftar layanan yang terintegrasi dengannya AWS KMS, lihat Integrasi AWS Layanan. Beberapa pertimbangan tentang penggunaan kunci KMS:

- Setelah membuat instans DB terenkripsi, Anda tidak dapat mengubah kunci KMS yang digunakan oleh instans DB tersebut. Oleh karena itu, tentukan persyaratan kunci KMS sebelum membuat instans DB terenkripsi.

  Jika Anda harus mengubah kunci enkripsi untuk cluster DB Anda, buat snapshot manual cluster Anda dan aktifkan enkripsi saat menyalin snapshot. Untuk informasi lebih lanjut, lihat artikel Re:Post Knowledge.
- Jika Anda menyalin snapshot terenkripsi, Anda dapat menggunakan kunci KMS yang berbeda untuk mengenkripsi snapshot target, bukan yang digunakan untuk mengenkripsi snapshot sumber.
- Anda tidak dapat membagikan snapshot yang telah dienkripsi menggunakan AWS akun Kunci yang dikelola AWS yang membagikan snapshot.
- Setiap instans DB dalam klaster DB dienkripsi menggunakan kunci KMS yang sama dengan klaster DB.
- Anda juga dapat mengenkripsi replika baca klaster terenkripsi Amazon Aurora.

**Important**

Amazon Aurora dapat kehilangan akses ke kunci KMS untuk cluster DB saat Anda menonaktifkan kunci KMS. Dalam kasus ini, klaster DB terenkripsi akan berstatus `inaccessible-encryption-credentials-recoverable`. Cluster DB tetap dalam keadaan ini selama tujuh hari, di mana instance dihentikan. Panggilan API yang dilakukan ke cluster DB selama waktu ini mungkin tidak berhasil. Untuk memulihkan cluster DB, aktifkan kunci KMS dan restart cluster DB ini. Aktifkan kunci KMS dari AWS Management Console,

AWS CLI, atau RDS API. Mulai ulang cluster DB menggunakan AWS CLI perintah start-db-clusteratau AWS Management Console.
`inaccessible-encryption-credentials-recoverable`Status hanya berlaku untuk cluster DB yang dapat berhenti. Status yang dapat dipulihkan ini tidak berlaku untuk instance yang tidak dapat dihentikan, seperti cluster dengan replika baca lintas wilayah. Untuk informasi selengkapnya, lihat the section called “Pembatasan”.
Jika cluster DB tidak dipulihkan dalam waktu tujuh hari, ia masuk ke `inaccessible-encryption-credentials` status terminal. Dalam keadaan ini, cluster DB tidak dapat digunakan lagi dan Anda hanya dapat memulihkan cluster DB dari cadangan. Kami sangat menyarankan agar Anda selalu mengaktifkan cadangan untuk cluster DB terenkripsi untuk mencegah hilangnya data terenkripsi dalam database Anda.
Selama pembuatan cluster DB, Aurora memeriksa apakah prinsipal panggilan memiliki akses ke kunci KMS dan menghasilkan hibah dari kunci KMS yang digunakan untuk seluruh masa pakai cluster DB. Mencabut akses prinsipal panggilan ke kunci KMS tidak memengaruhi database yang sedang berjalan. Saat menggunakan kunci KMS dalam skenario lintas akun, seperti menyalin snapshot ke akun lain, kunci KMS perlu dibagikan dengan akun lain. Jika Anda membuat cluster DB dari snapshot tanpa menentukan kunci KMS yang berbeda, cluster baru menggunakan kunci KMS dari akun sumber. Mencabut akses ke kunci setelah Anda membuat cluster DB tidak mempengaruhi cluster. Namun, menonaktifkan kunci berdampak pada semua cluster DB yang dienkripsi dengan kunci itu. Untuk mencegah hal ini, tentukan kunci yang berbeda selama operasi penyalinan snapshot.

Untuk informasi selengkapnya tentang kunci KMS, lihat AWS KMS keys di Panduan Developer AWS Key Management Service dan AWS KMS key manajemen.

Mengenkripsi klaster DB Amazon Aurora

Untuk mengenkripsi klaster DB baru, pilih Aktifkan enkripsi di konsol. Untuk informasi tentang pembuatan klaster DB, lihat Membuat klaster DB Amazon Aurora.

Jika Anda menggunakan create-db-cluster AWS CLI perintah untuk membuat cluster DB terenkripsi, atur parameternya. `--storage-encrypted` Jika Anda menggunakan operasi Create DBCluster API, setel `StorageEncrypted` parameter ke true.

Setelah membuat klaster DB terenkripsi, Anda tidak dapat mengubah kunci KMS yang digunakan oleh klaster DB tersebut. Oleh karena itu, tentukan persyaratan kunci KMS sebelum membuat klaster DB terenkripsi.

Jika Anda menggunakan AWS CLI `create-db-cluster` perintah untuk membuat cluster DB terenkripsi dengan kunci yang dikelola pelanggan, setel `--kms-key-id` parameter ke pengidentifikasi kunci apa pun untuk kunci KMS. Jika Anda menggunakan operasi `CreateDBInstance` Amazon RDS API, atur parameter `KmsKeyId` ke pengidentifikasi kunci mana pun untuk kunci KMS. Untuk menggunakan kunci yang dikelola pelanggan di AWS akun yang berbeda, tentukan kunci ARN atau alias ARN.

### Menentukan apakah enkripsi untuk klaster DB diaktifkan

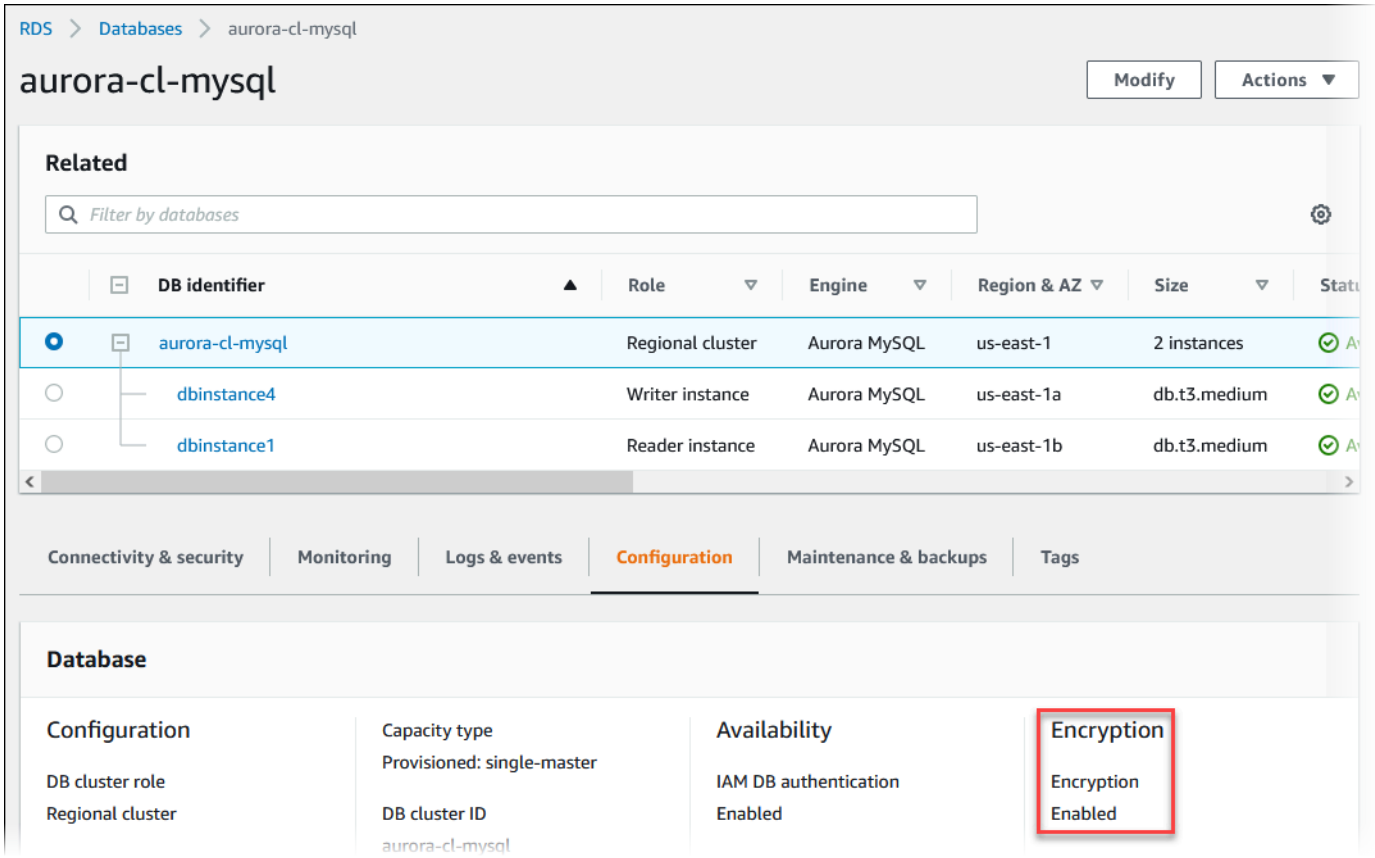
Anda dapat menggunakan AWS Management Console, AWS CLI, atau RDS API untuk menentukan apakah enkripsi saat istirahat diaktifkan untuk cluster DB.

#### Konsol

Untuk menentukan apakah enkripsi diam untuk klaster DB diaktifkan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih nama klaster DB yang ingin Anda periksa untuk melihat detailnya.
4. Pilih tab Konfigurasi dan periksa nilai Enkripsi.

   Akan muncul Diaktifkan atau Tidak diaktifkan.




AWS CLI

Untuk menentukan apakah enkripsi saat istirahat diaktifkan untuk cluster DB dengan menggunakan AWS CLI, panggil describe-db-clustersperintah dengan opsi berikut:

- `--db-cluster-identifier` – Nama klaster DB.

Contoh berikut ini menggunakan kueri untuk mengembalikan `TRUE` atau `FALSE` mengenai enkripsi diam untuk klaster DB `mydb`.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

API RDS

Untuk menentukan apakah enkripsi saat istirahat diaktifkan untuk cluster DB dengan menggunakan Amazon RDS API, panggil DBClusters operasi Deskripsikan dengan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB.

### Ketersediaan enkripsi Amazon Aurora

Enkripsi Amazon Aurora saat ini tersedia untuk semua jenis mesin basis data dan penyimpanan.

> **Note**
>
> Enkripsi Amazon Aurora tidak tersedia untuk kelas instans DB db.t2.micro.

### Enkripsi dalam transit

#### Enkripsi pada lapisan fisik

Semua data yang mengalir Wilayah AWS melalui jaringan AWS global secara otomatis dienkripsi pada lapisan fisik sebelum meninggalkan fasilitas yang aman. AWS Semua lalu lintas di antaranya AZs dienkripsi. Lapisan enkripsi tambahan, termasuk yang tercantum dalam bagian ini dapat memberikan perlindungan tambahan.

#### Enkripsi disediakan oleh peering VPC Amazon dan pengintipan lintas wilayah Gateway Transit

Semua lalu lintas lintas Wilayah yang menggunakan Amazon VPC dan Transit Gateway peering secara otomatis dienkripsi secara massal saat keluar dari suatu Wilayah. Lapisan enkripsi tambahan secara otomatis disediakan di lapisan fisik untuk semua lalu lintas sebelum meninggalkan fasilitas yang AWS aman.

#### Enkripsi antar instance

AWS menyediakan konektivitas aman dan pribadi antara instans DB dari semua jenis. Selain itu, beberapa tipe instans menggunakan kemampuan offload dari perangkat keras Nitro System yang mendasarinya untuk secara otomatis mengenkripsi lalu lintas dalam transit antar instans. Enkripsi ini menggunakan algoritma Authenticated Encryption with Associated Data (AEAD), dengan enkripsi 256-bit. Tidak ada dampak terhadap performa jaringan. Untuk mendukung enkripsi lalu lintas dalam transit tambahan ini antara instans, persyaratan-persyaratan berikut harus dipenuhi:

- Instans-instans tersebut menggunakan tipe instans berikut:
  - Tujuan umum: M6i, M6iD, M6in, M6idn, M7g
  - Memori yang dioptimalkan: R6i, R6id, R6in, R6idn, R7g, X2idn, X2IEDN, X2IEZN
- Contohnya sama Wilayah AWS.

- Instans berada dalam VPC atau VPCs peered yang sama, dan lalu lintas tidak melewati perangkat atau layanan jaringan virtual, seperti penyeimbang beban atau gateway transit.

Batasan klaster DB terenkripsi Amazon Aurora

Berikut batasan pada klaster DB terenkripsi Amazon Aurora:

- Anda tidak dapat menonaktifkan enkripsi pada klaster DB terenkripsi.
- Anda tidak dapat membuat snapshot terenkripsi dari klaster DB yang tidak terenkripsi.
- Snapshot klaster DB terenkripsi harus dienkripsi menggunakan kunci KMS yang sama dengan klaster DB.
- Anda tidak dapat mengonversi klaster DB tidak terenkripsi ke klaster terenkripsi. Namun, Anda dapat memulihkan snapshot tidak terenkripsi ke klaster DB Aurora terenkripsi. Untuk melakukannya, tentukan kunci KMS saat Anda memulihkan dari snapshot tidak terenkripsi.
- Anda tidak dapat membuat Aurora Replica terenkripsi dari klaster DB Aurora tidak terenkripsi. Anda tidak dapat membuat Aurora Replica tidak terenkripsi dari klaster DB Aurora terenkripsi.
- Untuk menyalin snapshot terenkripsi dari satu AWS Wilayah ke wilayah lain, Anda harus menentukan kunci KMS di Wilayah tujuan. AWS Ini karena kunci KMS khusus untuk AWS Wilayah tempat mereka dibuat.

  Tangkapan layar sumber tetap terenkripsi selama proses penyalinan. Amazon Aurora menggunakan enkripsi amplop untuk melindungi data selama proses penyalinan. Untuk informasi selengkapnya tentang enkripsi amplop, lihat Enkripsi amplop di Panduan Developer AWS Key Management Service .
- Anda tidak dapat membatalkan enkripsi klaster DB terenkripsi. Namun, Anda dapat mengekspor data dari klaster DB terenkripsi dan mengimpor data ke klaster DB tidak terenkripsi.

## AWS KMS key manajemen

Amazon Aurora secara otomatis terintegrasi AWS Key Management Service dengan AWS KMS() untuk manajemen kunci. Amazon Aurora menggunakan enkripsi amplop. Untuk informasi selengkapnya tentang enkripsi amplop, lihat Enkripsi amplop di Panduan Developer AWS Key Management Service .

Anda dapat menggunakan dua jenis AWS KMS kunci untuk mengenkripsi cluster DB Anda.

- Jika Anda menginginkan kontrol penuh atas kunci KMS, Anda harus membuat kunci yang dikelola pelanggan. Untuk informasi selengkapnya tentang kunci yang dikelola pelanggan, lihat Kunci yang dikelola pelanggan di Panduan Developer AWS Key Management Service .
- Kunci yang dikelola AWSadalah kunci KMS di akun Anda yang dibuat, dikelola, dan digunakan atas nama Anda oleh AWS layanan yang terintegrasi dengannya AWS KMS. Secara default, RDS Kunci yang dikelola AWS (`aws/rds`) digunakan untuk enkripsi. Anda tidak dapat mengelola, memutar, atau menghapus RDS. Kunci yang dikelola AWS Untuk informasi selengkapnya Kunci yang dikelola AWS, lihat Kunci yang dikelola AWSdi Panduan AWS Key Management Service Pengembang.

Untuk mengelola kunci KMS yang digunakan untuk cluster instans terenkripsi Amazon Aurora, gunakan AWS KMS() di AWS Key Management ServiceAWS KMS konsol, atau API. AWS CLI AWS KMS Untuk melihat log Audit setiap tindakan yang diambil dengan kunci yang dikelola pelanggan atau AWS , gunakan AWS CloudTrail. Untuk informasi selengkapnya tentang rotasi kunci, lihat Merotasi kunci AWS KMS.

Mengizinkan penggunaan kunci yang dikelola pelanggan

Ketika Aurora menggunakan kunci yang dikelola pelanggan dalam operasi kriptografi, ia bertindak atas nama pengguna yang membuat atau mengubah sumber daya Aurora.

Untuk membuat sumber daya Aurora menggunakan kunci yang dikelola pelanggan, pengguna harus memiliki izin untuk memanggil operasi berikut pada kunci yang dikelola pelanggan:

- `kms:CreateGrant`
- `kms:DescribeKey`

Anda dapat menentukan izin yang diperlukan ini dalam kebijakan kunci, atau dalam kebijakan IAM jika kebijakan kunci memungkinkan hal tersebut.

> **Important**
>
> Bila Anda menggunakan pernyataan penolakan eksplisit untuk semua sumber daya (*) dalam kebijakan AWS KMS utama dengan layanan terkelola seperti Amazon RDS, Anda harus menentukan kondisi untuk mengizinkan akun pemilik sumber daya. Operasi mungkin gagal tanpa kondisi ini, bahkan jika aturan penolakan menyertakan pengecualian untuk pengguna IAM Anda.

> **Tip**
>
> Untuk mengikuti prinsip hak akses paling rendah, jangan biarkan akses penuh ke `kms:CreateGrant`. Sebagai gantinya, gunakan kunci ViaService kondisi kms: untuk memungkinkan pengguna membuat hibah pada kunci KMS hanya ketika hibah dibuat atas nama pengguna oleh layanan. AWS

Anda dapat membuat kebijakan IAM lebih ketat dalam berbagai cara. Misalnya, jika Anda ingin mengizinkan kunci yang dikelola pelanggan hanya digunakan untuk permintaan yang berasal dari , gunakan kunci kms: ViaService kondisi dengan nilainya. `rds.<region>.amazonaws.com` Selain itu, Anda dapat menggunakan kunci atau nilai dalam Konteks enkripsi Amazon RDS sebagai syarat untuk menggunakan kunci yang dikelola pelanggan untuk enkripsi.

Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS di Panduan Developer AWS Key Management Service dan Kebijakan kunci di AWS KMS.

Konteks enkripsi Amazon RDS

Ketika Aurora menggunakan kunci KMS Anda, atau ketika Amazon EBS menggunakan kunci KMS atas nama Aurora, layanan menentukan konteks enkripsi. Konteks enkripsi adalah data otentikasi tambahan (AAD) yang AWS KMS digunakan untuk memastikan integritas data. Ketika konteks enkripsi ditentukan untuk operasi enkripsi, layanan harus menentukan konteks enkripsi yang sama untuk operasi dekripsi. Jika tidak, dekripsi akan gagal. Konteks enkripsi juga ditulis ke log AWS CloudTrail untuk membantu Anda memahami mengapa kunci KMS tertentu digunakan. CloudTrail Log Anda mungkin berisi banyak entri yang menjelaskan penggunaan kunci KMS, tetapi konteks enkripsi di setiap entri log dapat membantu Anda menentukan alasan penggunaan tertentu.

Minimal, Aurora selalu menggunakan ID cluster instans untuk konteks enkripsi, seperti pada contoh berformat JSON berikut:

```
{ "aws:rds:dbc-id": "cluster-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

Konteks enkripsi ini dapat membantu Anda mengidentifikasi cluster DB tempat kunci KMS Anda digunakan.

Bila kunci KMS Anda digunakan untuk cluster DB tertentu dan volume Amazon EBS tertentu, ID cluster DB dan ID volume Amazon EBS digunakan untuk konteks enkripsi, seperti pada contoh berformat JSON berikut:

```
{
  "aws:rds:dbc-id": "cluster-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

## Menggunakan SSL/TLS untuk mengenkripsi koneksi ke

Anda dapat menggunakan Lapisan Soket Aman (SSL) atau Keamanan Lapisan Pengangkutan (TLS) dari aplikasi Anda untuk mengenkripsi koneksi ke klaster DB yang menjalankan Aurora MySQL atau Aurora PostgreSQL.

Secara opsional, SSL/TLS koneksi Anda dapat melakukan verifikasi identitas server dengan memvalidasi sertifikat server yang diinstal pada database Anda. Untuk meminta verifikasi identitas server, ikuti proses umum ini:

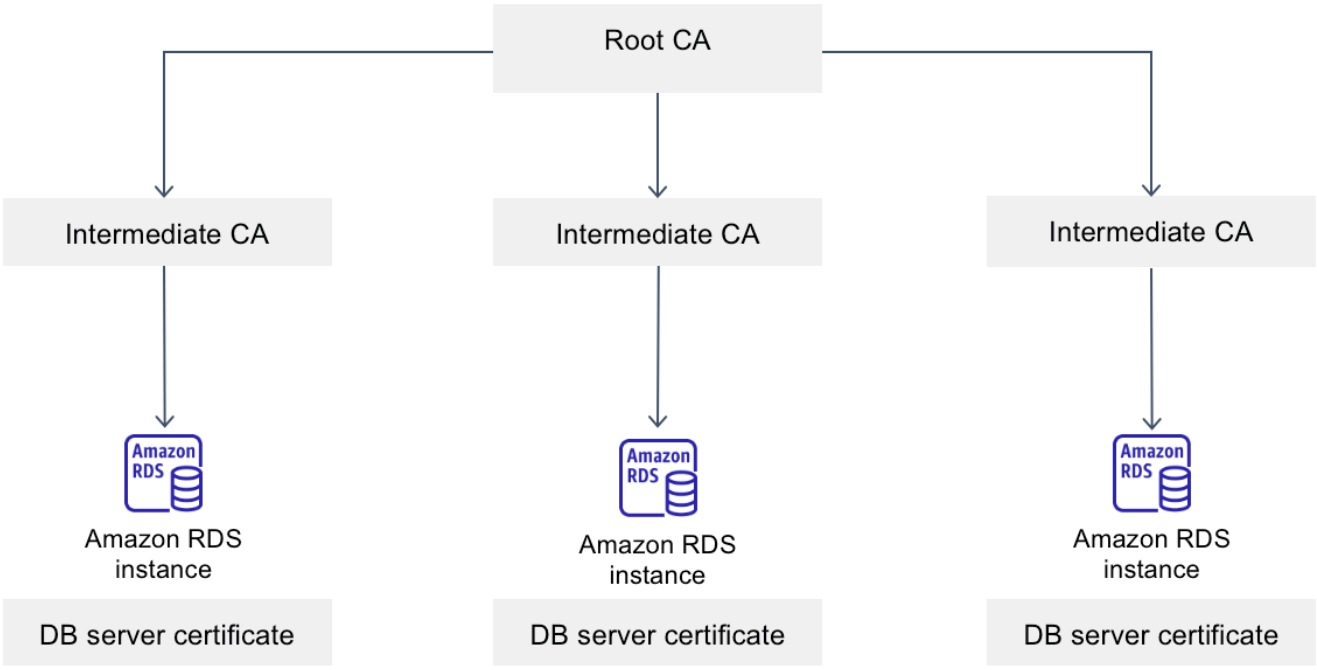
1. Pilih Otoritas Sertifikat (CA) yang menandatangani sertifikat server DB, untuk basis data Anda. Untuk informasi selengkapnya tentang otoritas sertifikat, lihatOtoritas sertifikat.
2. Unduh paket sertifikat yang akan digunakan saat Anda terhubung ke basis data. Untuk mengunduh bundel sertifikat, lihat Bundel sertifikat oleh Wilayah AWS.

   > **Note**
   >
   > Semua sertifikat hanya tersedia untuk diunduh menggunakan koneksi SSL/TLS.

3. Connect ke database menggunakan proses mesin DB Anda untuk mengimplementasikan SSL/TLS koneksi. Setiap mesin DB memiliki proses sendiri SSL/TLS. To learn how to implement SSL/TLS untuk mengimplementasikan database Anda, ikuti tautan yang sesuai dengan mesin DB Anda:
   - Keamanan dengan Amazon Aurora MySQL
   - Keamanan dengan Amazon Aurora PostgreSQL

### Otoritas sertifikat

Otoritas Sertifikat (CA) adalah sertifikat yang mengidentifikasi CA root di bagian atas rantai sertifikat. CA menandatangani sertifikat server DB, yang diinstal pada setiap instans DB. Sertifikat server DB mengidentifikasi instans DB sebagai server tepercaya.




Amazon RDS menyediakan yang berikut ini CAs untuk menandatangani sertifikat server DB untuk database.

| Otoritas sertifikat (CA) | Deskripsi | Nama umum (CN) |
|---|---|---|
| rds-ca-rsa2048-g1 | Menggunakan otoritas sertifikat dengan algoritma kunci pribadi RSA 2048 dan algoritma SHA256 penandatanganan di sebagian besar. Wilayah AWS<br><br>Dalam AWS GovCloud (US) Regions, CA ini menggunakan otoritas sertifikat dengan algoritma kunci pribadi RSA 2048 dan algoritma SHA384 penandatanganan.<br><br>CA ini mendukung rotasi sertifikat server otomatis. | Amazon RDS *region-id entifier* RSA2 048 G1 |
| rds-ca-rsa4096-g1 | Menggunakan otoritas sertifikat dengan algoritma kunci pribadi RSA 4096 dan algoritma SHA384 penandatanganan. | Amazon RDS *region-id* |

| Otoritas sertifikat (CA) | Deskripsi | Nama umum (CN) |
| --- | --- | --- |
| | CA ini mendukung rotasi sertifikat server otomatis. | *entifier* RSA4 096 G1 |
| rds-ca-ecc384-g1 | Menggunakan otoritas sertifikat dengan algoritma kunci pribadi ECC 384 dan algoritma SHA384 penandatanganan. CA ini mendukung rotasi sertifikat server otomatis. | Amazon RDS G1 *region-identifier* ECC384 |

> **Note**
>
> Jika Anda menggunakan AWS CLI, Anda dapat melihat validitas otoritas sertifikat yang tercantum di atas dengan menggunakan deskripsi-sertifikat.

Sertifikat CA ini termasuk dalam paket sertifikat regional dan global. Bila Anda menggunakan rds-ca-rsa 2048-g1, rds-ca-rsa 4096-g1, atau rds-ca-ecc 384-g1 CA dengan database, RDS mengelola sertifikat server DB pada database. RDS merotasi sertifikat server DB secara otomatis sebelum sertifikat ini kedaluwarsa.

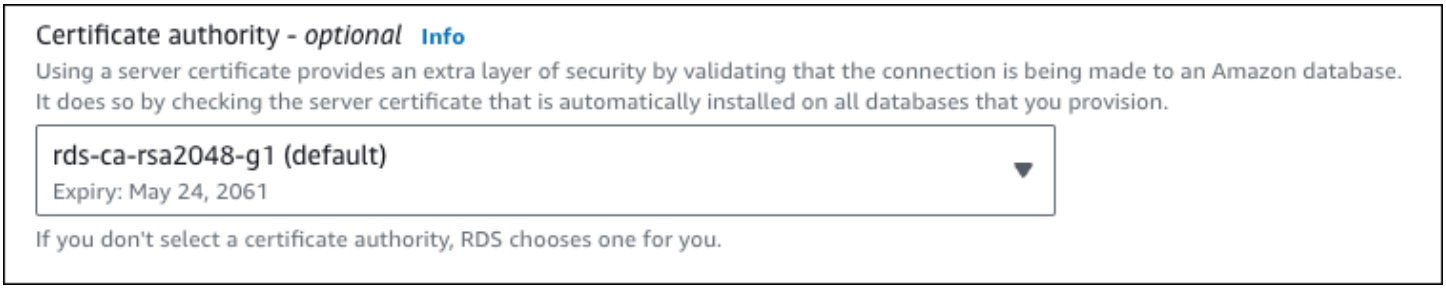
Mengatur CA untuk basis data Anda

Anda dapat mengatur CA untuk basis data saat Anda melakukan tugas berikut:

- Buat cluster Aurora DB — Anda dapat mengatur CA untuk instans DB di cluster Aurora saat Anda membuat instans DB pertama di cluster DB menggunakan atau RDS API. AWS CLI Saat ini, Anda tidak dapat mengatur CA untuk instans DB di klaster DB saat Anda membuat klaster DB menggunakan konsol RDS. Untuk instruksi, lihatMembuat klaster DB Amazon Aurora.
- Memodifikasi instans DB – Anda dapat mengatur CA untuk instans DB di klaster DB dengan memodifikasinya. Untuk instruksi, lihatMemodifikasi instans DB dalam klaster DB.

> **Note**
>
> CA default diatur ke rds-ca-rsa 2048-g1. Anda dapat mengganti CA default untuk Anda Akun AWS dengan menggunakan perintah modify-certificate.

Yang tersedia CAs tergantung pada mesin DB dan versi mesin DB. Bila Anda menggunakan AWS Management Console, Anda dapat memilih CA menggunakan pengaturan otoritas Sertifikat, seperti yang ditunjukkan pada gambar berikut.




Konsol hanya menampilkan CAs yang tersedia untuk mesin DB dan versi mesin DB. Jika Anda menggunakan AWS CLI, Anda dapat mengatur CA untuk instans DB menggunakan modify-db-instanceperintah create-db-instanceor.

Jika Anda menggunakan AWS CLI, Anda dapat melihat yang tersedia CAs untuk akun Anda dengan menggunakan perintah describe-certificate. Perintah ini juga menunjukkan tanggal kedaluwarsa untuk setiap CA di `ValidTill` dalam output. Anda dapat menemukan CAs yang tersedia untuk mesin DB tertentu dan versi mesin DB menggunakan describe-db-engine-versionsperintah.

Contoh berikut menunjukkan CAs tersedia untuk RDS default untuk versi mesin PostgreSQL DB.

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

Output Anda akan seperti yang berikut ini. Yang CAs tersedia tercantum dalam`SupportedCACertificateIdentifiers`. Output ini juga menunjukkan apakah versi mesin DB mendukung rotasi sertifikat tanpa pengaktifan ulang di `SupportsCertificateRotationWithoutRestart`.

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
```

```
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

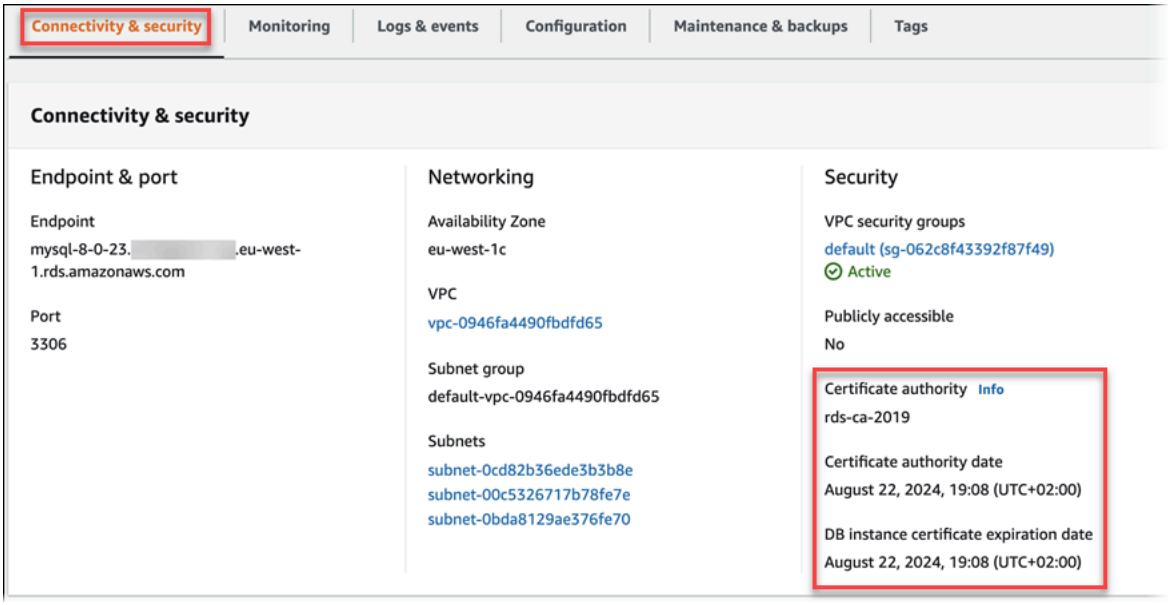
Validitas sertifikat server DB

Validitas sertifikat server DB bergantung pada mesin DB dan versi mesin DB. Jika versi mesin DB mendukung rotasi sertifikat tanpa pengaktifan ulang, validitas sertifikat server DB adalah 1 tahun. Jika tidak, validitasnya adalah 3 tahun.

Untuk informasi selengkapnya tentang rotasi sertifikat server DB, lihatRotasi sertifikat server otomatis.

Melihat CA untuk instans DB Anda

Anda dapat melihat detail tentang CA untuk database dengan melihat tab Konektivitas & keamanan di konsol, seperti pada gambar berikut.




Jika Anda menggunakan AWS CLI, Anda dapat melihat detail tentang CA untuk instans DB dengan menggunakan describe-db-instancesperintah.

Unduh bundel sertifikat untuk Aurora

Saat Anda terhubung ke database dengan SSL atau TLS, instance database memerlukan sertifikat kepercayaan dari Amazon RDS. Pilih tautan yang sesuai di tabel berikut untuk mengunduh bundel yang sesuai dengan Wilayah AWS tempat Anda meng-host database Anda.

Bundel sertifikat oleh Wilayah AWS

Bundel sertifikat untuk semua Wilayah AWS dan Wilayah GovCloud (AS) berisi sertifikat CA root berikut:

- `rds-ca-rsa2048-g1`
- `rds-ca-rsa4096-g1`
- `rds-ca-ecc384-g1`

`rds-ca-ecc384-g1`Sertifikat `rds-ca-rsa4096-g1` dan sertifikat tidak tersedia di Wilayah berikut:

- Asia Pasifik (Mumbai)
- Asia Pasifik (Melbourne)
- Kanada Barat (Calgary)
- Eropa (Zürich)

- Eropa (Spanyol)
- Israel (Tel Aviv)

Toko kepercayaan aplikasi Anda hanya perlu mendaftarkan sertifikat CA root.

> **Note**
>
> Amazon RDS Proxy dan Aurora Serverless v1 penggunaan sertifikat dari AWS Certificate Manager (ACM). Jika Anda menggunakan RDS Proxy, Anda tidak perlu mengunduh sertifikat Amazon RDS atau memperbarui aplikasi yang menggunakan koneksi Proxy RDS. Untuk informasi selengkapnya, lihat MenggunakanTLS/SSLdengan RDS Proxy.
> Jika Anda menggunakanAurora Serverless v1, mengunduh sertifikat Amazon RDS tidak diperlukan. Untuk informasi selengkapnya, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1.

Untuk mengunduh bundel sertifikat Wilayah AWS, pilih tautan untuk Wilayah AWS yang menghosting database Anda di tabel berikut.

| AWS Wilayah | Paket sertifikat (PEM) | Bundel sertifikat (PKCS7) |
|---|---|---|
| Komersil apa pun Wilayah AWS | global-bundle.pem | global-bundle.p7b |
| AS Timur (Virginia Utara) | us-east-1-bundle.pem | us-east-1-bundle.p7b |
| AS Timur (Ohio) | us-east-2-bundle.pem | us-east-2-bundle.p7b |
| AS Barat (California Utara) | us-west-1-bundle.pem | us-west-1-bundle.p7b |
| AS Barat (Oregon) | us-west-2-bundle.pem | us-west-2-bundle.p7b |
| Afrika (Cape Town) | af-south-1-bundle.pem | af-south-1-bundle.p7b |
| Asia Pasifik (Hong Kong) | ap-east-1-bundle.pem | ap-east-1-bundle.p7b |
| Asia Pasifik (Hyderabad) | ap-south-2-bundle.pem | ap-south-2-bundle.p7b |
| Asia Pasifik (Jakarta) | ap-southeast-3-bundle.pem | ap-southeast-3-bundle.p7b |

| AWS Wilayah | Paket sertifikat (PEM) | Bundel sertifikat (PKCS7) |
|---|---|---|
| Asia Pasifik (Malaysia) | ap-tenggara 5-bundel.pem | ap-tenggara 5-bundel.p7b |
| Asia Pasifik (Melbourne) | ap-southeast-4-bundle.pem | ap-southeast-4-bundle.p7b |
| Asia Pasifik (Mumbai) | ap-south-1-bundle.pem | ap-south-1-bundle.p7b |
| Asia Pasifik (Osaka) | ap-northeast-3-bundle.pem | ap-northeast-3-bundle.p7b |
| Asia Pasifik (Thailand) | ap-tenggara 7-bundel.pem | ap-tenggara 7-bundel.p7b |
| Asia Pasifik (Tokyo) | ap-northeast-1-bundle.pem | ap-northeast-1-bundle.p7b |
| Asia Pasifik (Seoul) | ap-northeast-2-bundle.pem | ap-northeast-2-bundle.p7b |
| Asia Pasifik (Singapura) | ap-southeast-1-bundle.pem | ap-southeast-1-bundle.p7b |
| Asia Pasifik (Sydney) | ap-southeast-2-bundle.pem | ap-southeast-2-bundle.p7b |
| Kanada (Pusat) | ca-central-1-bundle.pem | ca-central-1-bundle.p7b |
| Kanada Barat (Calgary) | ca-barat-1-bundle.pem | ca-barat-1-bundle.p7b |
| Eropa (Frankfurt) | eu-central-1-bundle.pem | eu-central-1-bundle.p7b |
| Eropa (Irlandia) | eu-west-1-bundle.pem | eu-west-1-bundle.p7b |
| Eropa (London) | eu-west-2-bundle.pem | eu-west-2-bundle.p7b |
| Eropa (Milan) | eu-south-1-bundle.pem | eu-south-1-bundle.p7b |
| Eropa (Paris) | eu-west-3-bundle.pem | eu-west-3-bundle.p7b |
| Eropa (Spanyol) | eu-south-2-bundle.pem | eu-south-2-bundle.p7b |
| Eropa (Stockholm) | eu-north-1-bundle.pem | eu-north-1-bundle.p7b |
| Eropa (Zürich) | eu-central-2-bundle.pem | eu-central-2-bundle.p7b |
| Israel (Tel Aviv) | il-central-1-bundle.pem | il-central-1-bundle.p7b |

| AWS Wilayah | Paket sertifikat (PEM) | Bundel sertifikat (PKCS7) |
|---|---|---|
| Meksiko (Tengah) | mx-sentral-1-bundel.pem | mx-sentral-1-bundel.p7b |
| Timur Tengah (Bahrain) | me-south-1-bundle.pem | me-south-1-bundle.p7b |
| Timur Tengah (UEA) | me-central-1-bundle.pem | me-central-1-bundle.p7b |
| Amerika Selatan (Sao Paulo) | sa-east-1-bundle.pem | sa-east-1-bundle.p7b |
| Setiap AWS GovCloud (US) Region s | global-bundle.pem | global-bundle.p7b |
| AWS GovCloud (AS-Timur) | us-gov-east-1-bundel.pem | us-gov-east-1-bundel.p7b |
| AWS GovCloud (AS-Barat) | us-gov-west-1-bundel.pem | us-gov-west-1-bundel.p7b |

Melihat isi sertifikat CA Anda

Untuk memeriksa konten paket sertifikat CA Anda, gunakan perintah berikut:

```
keytool -printcert -v -file global-bundle.pem
```

## Merotasi sertifikat SSL/TLS

Sertifikat Otoritas Sertifikat Amazon RDS rds-ca-2019 berakhir pada Agustus 2024. Jika Anda menggunakan atau berencana untuk menggunakan Secure Sockets Layer (SSL) atau Transport Layer Security (TLS) dengan verifikasi sertifikat untuk terhubung ke instans RDS DB , pertimbangkan untuk menggunakan salah satu sertifikat rds-ca-rsa CA baru 2048-g1, 4096-g1 atau 384-g1. rds-ca-rsa rds-ca-ecc Jika saat ini Anda tidak menggunakan SSL/TLS with certificate verification, you might still have an expired CA certificate and must update them to a new CA certificate if you plan to use SSL/TLS dengan verifikasi sertifikat untuk terhubung ke database RDS Anda.

Amazon RDS menyediakan sertifikat CA baru sebagai praktik terbaik AWS keamanan. Untuk informasi tentang sertifikat baru dan AWS Wilayah yang didukung, lihatMenggunakan SSL/TLS untuk mengenkripsi koneksi ke .

Untuk memperbarui sertifikat CA untuk database Anda, gunakan metode berikut:

- Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB Anda

- Memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan

Sebelum memperbarui instans DB untuk menggunakan sertifikat CA baru, pastikan Anda memperbarui klien atau aplikasi yang terhubung ke database RDS Anda.

Pertimbangan untuk memutar sertifikat

Pertimbangkan situasi berikut sebelum memutar sertifikat Anda:

- Amazon RDS Proxy dan Aurora Serverless v1 penggunaan sertifikat dari AWS Certificate Manager (ACM). Jika Anda menggunakan RDS Proxy, saat memutar sertifikat SSL/TLS, Anda tidak perlu memperbarui aplikasi yang menggunakan koneksi Proxy RDS. Untuk informasi selengkapnya, lihat MenggunakanTLS/SSLdengan RDS Proxy.
- Jika Anda menggunakanAurora Serverless v1, mengunduh sertifikat Amazon RDS tidak diperlukan. Untuk informasi selengkapnya, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1.
- Jika Anda menggunakan aplikasi Go versi 1.15 dengan instans DB yang dibuat atau diperbarui ke sertifikat rds-ca-2019 sebelum 28 Juli 2020, Anda harus memperbarui sertifikat lagi.

  Gunakan `modify-db-instance` perintah , menggunakan pengidentifikasi sertifikat CA baru. Anda dapat menemukan CAs yang tersedia untuk mesin DB tertentu dan versi mesin DB menggunakan `describe-db-engine-versions` perintah.

  Jika Anda membuat database atau memperbarui sertifikatnya setelah 28 Juli 2020, tidak ada tindakan yang diperlukan. Untuk informasi selengkapnya, lihat Go GitHub issue #39568.

Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB Anda

Contoh berikut memperbarui sertifikat CA Anda dari rds-ca-2019 ke 2048-g1. rds-ca-rsa Anda dapat memilih sertifikat yang berbeda. Untuk informasi lebih lanjut, lihatOtoritas sertifikat.

Perbarui toko kepercayaan aplikasi Anda untuk mengurangi waktu henti yang terkait dengan memperbarui sertifikat CA Anda. Untuk informasi selengkapnya tentang restart yang terkait dengan rotasi sertifikat CA, lihatRotasi sertifikat server otomatis.

Untuk memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB Anda

1. Unduh sertifikat SSL/TLS baru seperti yang dijelaskan dalam. Menggunakan SSL/TLS untuk mengenkripsi koneksi ke

2. Perbarui aplikasi Anda untuk menggunakan sertifikat SSL/TLS baru.

   Metode untuk memperbarui aplikasi untuk SSL/TLS certificates depend on your specific applications. Work with your application developers to update the SSL/TLS sertifikat baru untuk aplikasi Anda.

   Untuk informasi tentang pemeriksaan koneksi SSL/TLS dan pembaruan aplikasi untuk setiap mesin DB, lihat topik berikut:

   - Memperbarui aplikasi untuk terhubung ke klaster DB Aurora MySQL menggunakan sertifikat TLS baru
   - Memperbarui aplikasi untuk terhubung ke klaster DB Aurora PostgreSQL menggunakan sertifikat SSL/TLS baru

   Untuk contoh skrip yang memperbarui toko kepercayaan untuk sistem operasi Linux, lihatContoh skrip untuk mengimpor sertifikat ke trust store Anda.

   > **Note**
   >
   > Paket sertifikat berisi sertifikat untuk CA lama dan baru, sehingga Anda dapat meningkatkan aplikasi Anda dengan aman dan mempertahankan konektivitas selama periode transisi. Jika Anda menggunakan AWS Database Migration Service untuk memigrasikan database ke cluster, sebaiknya gunakan bundel sertifikat untuk memastikan konektivitas selama migrasi.

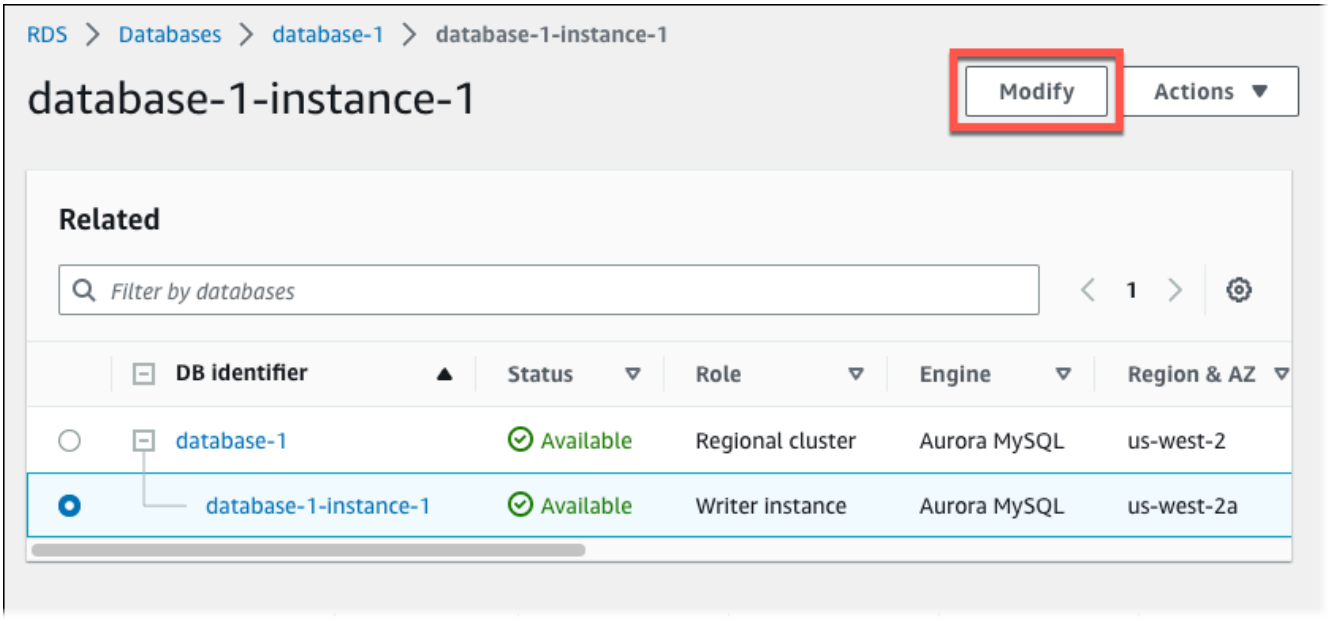
3. Ubah instans DB untuk mengubah CA dari rds-ca-2019 menjadi 2048-g1. rds-ca-rsa Untuk memeriksa apakah database Anda memerlukan restart untuk memperbarui sertifikat CA, gunakan describe-db-engine-versionsperintah dan periksa `SupportsCertificateRotationWithoutRestart` bendera.

   > **Note**
   >
   > Boot ulang klaster Babelfish Anda setelah memodifikasi untuk memperbarui sertifikat CA.

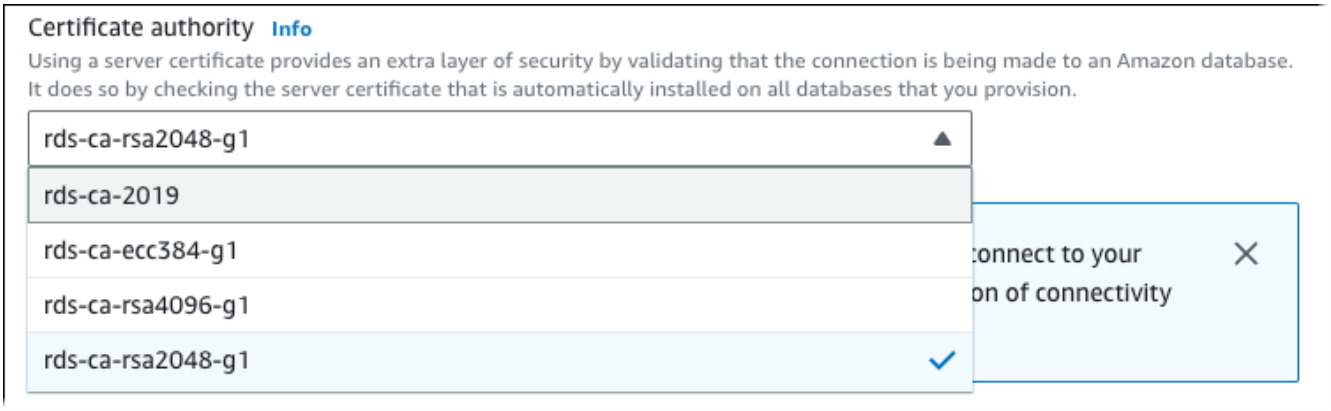
> **Important**
>
> Jika Anda mengalami masalah konektivitas setelah masa berlaku sertifikat berakhir, gunakan opsi terapkan segera dengan menentukan Terapkan segera di konsol atau dengan menentukan opsi `--apply-immediately` menggunakan AWS CLI. Secara default, operasi ini dijadwalkan untuk berjalan selama jendela pemeliharaan berikutnya. Untuk menyetel penggantian CA cluster Anda yang berbeda dari CA RDS default, gunakan perintah CLI modify-certificate.

Console

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Databases, lalu pilih instans DB yang ingin Anda modifikasi.
3. Pilih Ubah.




4. Di bagian Konektivitas, pilih rds-ca-rsa2048-g1.




5. Pilih Lanjutkan dan periksa ringkasan modifikasi.
6. Untuk segera menerapkan perubahan, pilih Terapkan segera.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika benar, pilih Modify DB Instance untuk menyimpan perubahan Anda.

> **Important**
>
> Saat Anda menjadwalkan operasi ini, pastikan bahwa Anda telah memperbarui trust store sisi klien sebelumnya.

Atau pilih Kembali untuk mengedit perubahan atau Batalkan untuk membatalkan perubahan Anda.

AWS CLI

Untuk menggunakan AWS CLI untuk mengubah CA dari rds-ca-2019 ke rds-ca-rsa2048-g1 untuk instans DB atau cluster DB multi-AZ, panggil perintah . [modify-db-instancemodify-db-cluster](#) Tentukan instans DB dan `--ca-certificate-identifier` opsi.

Gunakan `--apply-immediately` parameter untuk segera menerapkan pembaruan. Secara default, operasi ini dijadwalkan untuk berjalan selama periode pemeliharaan berikutnya.

> **Important**
>
> Saat Anda menjadwalkan operasi ini, pastikan bahwa Anda telah memperbarui trust store sisi klien sebelumnya.

Example

Contoh berikut memodifikasi `mydbinstance` dengan menyetel sertifikat CA ke `rds-ca-rsa2048-g1`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

> **Note**
>
> Jika instance Anda memerlukan reboot, Anda dapat menggunakan perintah modify-db-instanceCLI dan menentukan opsi. `--no-certificate-rotation-restart`

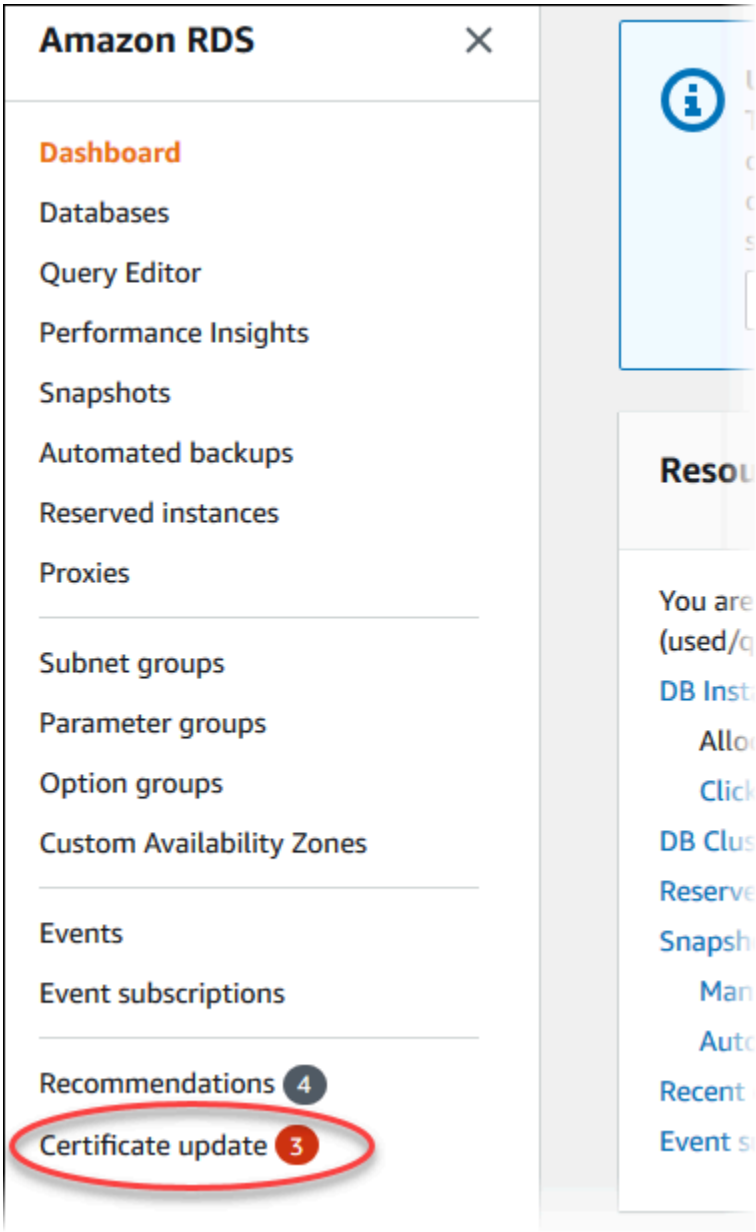
Memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan

Lakukan langkah-langkah berikut untuk memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan.

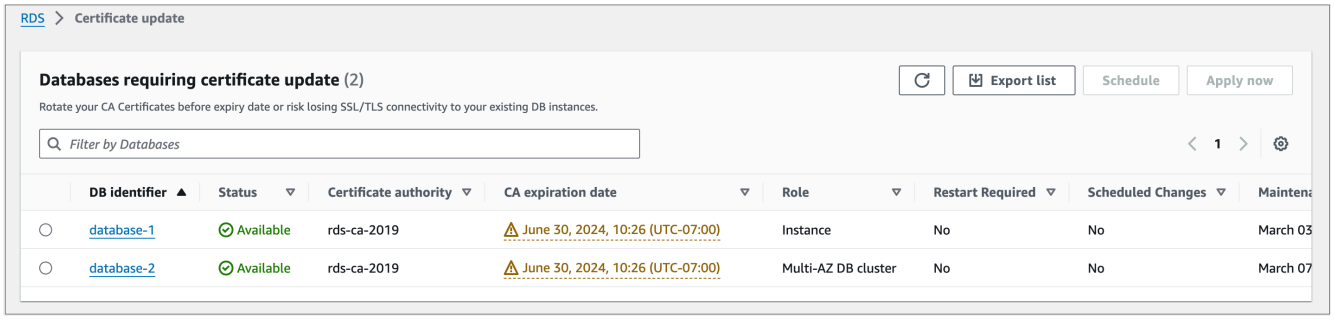
Console

Untuk memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pembaruan sertifikat.




Halaman Basis data yang memerlukan pembaruan sertifikat akan muncul.



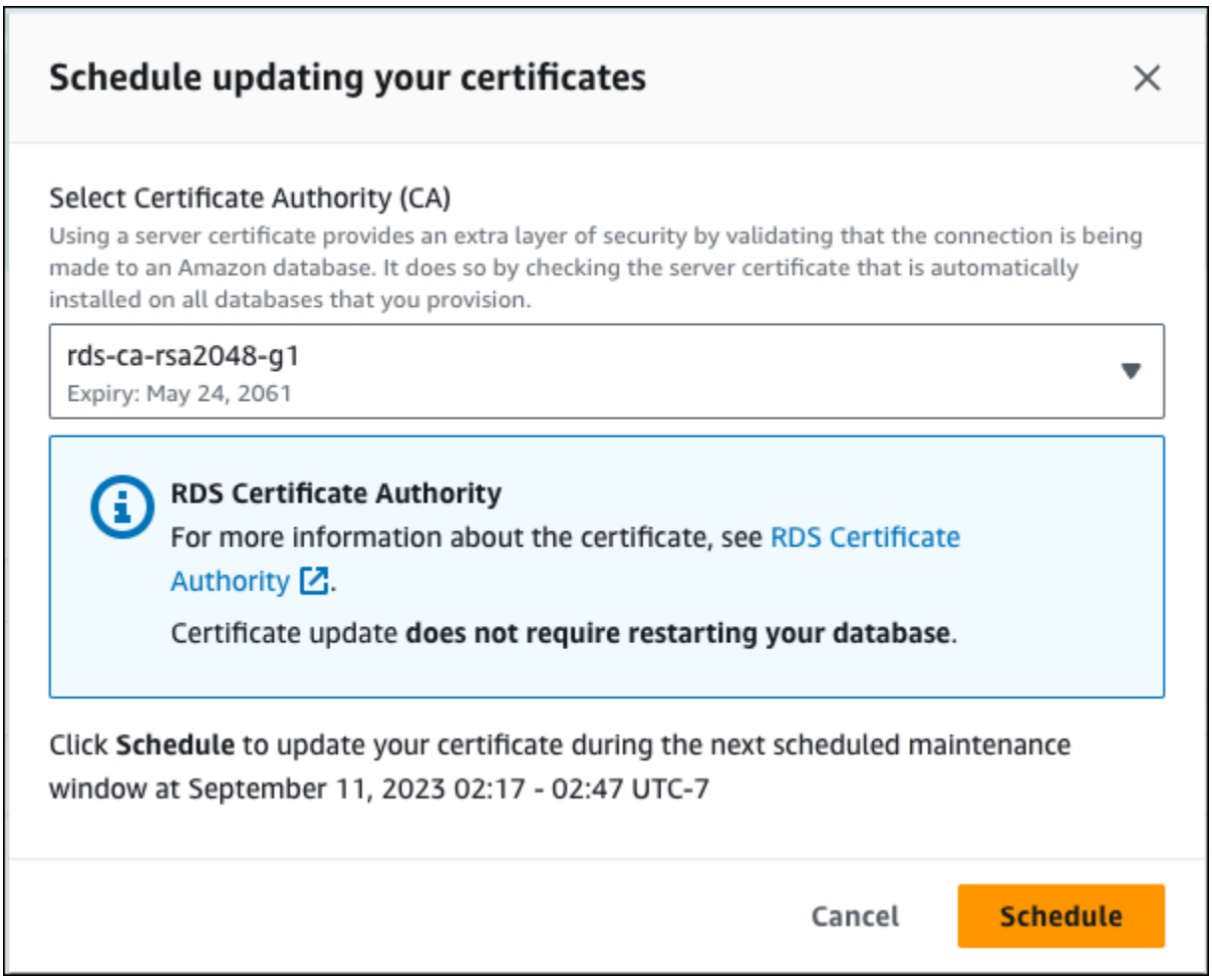
> **Note**
>
> Halaman ini hanya menampilkan instans DB untuk saat ini. Wilayah AWS Jika Anda memiliki database di lebih dari satu Wilayah AWS, periksa halaman ini di masing-masing Wilayah AWS untuk melihat semua instance DB dengan sertifikat SSL/TLS lama.

3. Pilih instans DB yang ingin Anda perbarui.

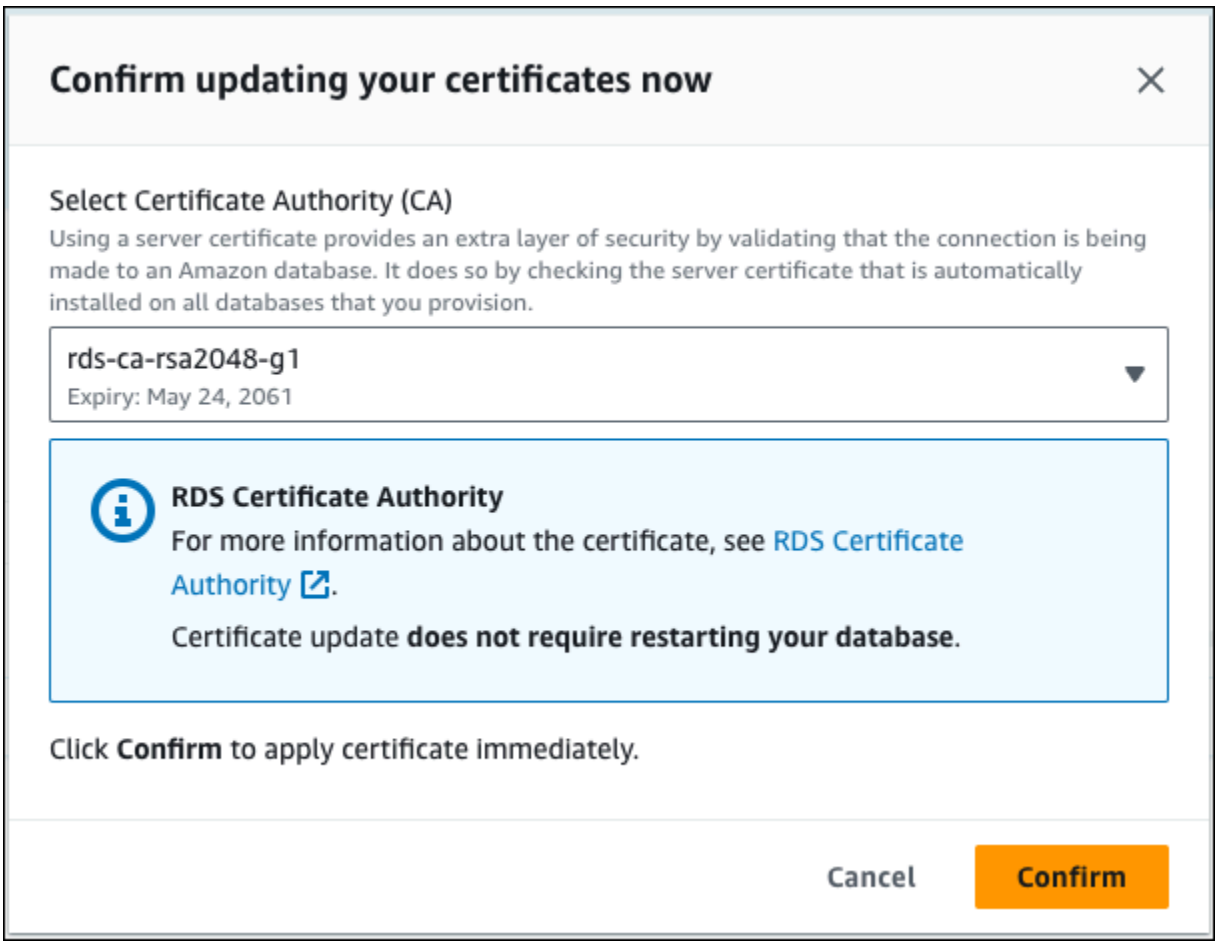
   Anda dapat menjadwalkan rotasi sertifikat untuk periode pemeliharaan berikutnya dengan memilih Jadwal. Segera terapkan rotasi dengan memilih Terapkan sekarang.

   > **Important**
   >
   > Jika Anda mengalami masalah konektivitas setelah sertifikat kedaluwarsa, gunakan opsi Terapkan sekarang.

4. a. Jika Anda memilih Jadwal, Anda akan diminta untuk mengonfirmasi rotasi sertifikat CA. Prompt ini juga menyatakan periode terjadwal untuk pembaruan Anda.




b. Jika Anda memilih Terapkan sekarang, Anda akan diminta untuk mengonfirmasi rotasi sertifikat CA.




> **Important**
>
> Sebelum menjadwalkan rotasi sertifikat CA di basis data Anda, perbarui aplikasi klien yang menggunakan SSL/TLS dan sertifikat server untuk terhubung. Pembaruan ini khusus untuk mesin DB Anda. Setelah Anda memperbarui aplikasi klien ini, Anda dapat mengonfirmasi rotasi sertifikat CA.

Untuk melanjutkan, pilih kotak centang, lalu pilih Konfirmasi.

5. Ulangi langkah 3 dan 4 untuk setiap instans DB yang ingin Anda perbarui.

Rotasi sertifikat server otomatis

Jika CA root Anda mendukung rotasi sertifikat server otomatis, RDS secara otomatis menangani rotasi sertifikat server DB. RDS menggunakan CA root yang sama untuk rotasi otomatis ini, jadi Anda tidak perlu mengunduh paket CA baru. LihatOtoritas sertifikat.

Rotasi dan validitas sertifikat server DB Anda bergantung pada mesin DB Anda:

- Jika mesin DB Anda mendukung rotasi tanpa pengaktifan ulang, RDS secara otomatis merotasi sertifikat server DB tanpa memerlukan tindakan apa pun dari Anda. RDS mencoba merotasi sertifikat server DB Anda dalam periode pemeliharaan yang Anda pilih di waktu paruh sertifikat server DB. Sertifikat server DB baru berlaku selama 12 bulan.
- Jika mesin DB Anda tidak mendukung rotasi tanpa pengaktifan ulang, RDS memberi tahu Anda tentang peristiwa pemeliharaan setidaknya 6 bulan sebelum sertifikat server DB kedaluwarsa. Sertifikat server DB baru berlaku selama 36 bulan.

Gunakan describe-db-engine-versionsperintah dan periksa `SupportsCertificateRotationWithoutRestart` bendera untuk mengidentifikasi apakah versi mesin DB mendukung memutar sertifikat tanpa memulai ulang. Untuk informasi selengkapnya, lihat Mengatur CA untuk basis data Anda.

Contoh skrip untuk mengimpor sertifikat ke trust store Anda

Berikut adalah contoh skrip shell yang mengimpor paket sertifikat ke trust store.

Setiap skrip shell menggunakan keytool, yang merupakan bagian dari Java Development Kit (JDK). Untuk informasi tentang cara menginstal JDK, lihat JDK Installation Guide.

Linux

Berikut adalah contoh skrip shell yang mengimpor paket sertifikat ke trust store di sistem operasi Linux.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit
```

```
curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS

Berikut adalah contoh skrip shell yang mengimpor paket sertifikat ke trust store di sistem operasi macOS.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
```

```
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

Untuk mempelajari lebih lanjut praktik terbaik tentang penggunaan SSL dengan Amazon RDS, lihat Praktik terbaik untuk koneksi SSL yang berhasil ke Amazon RDS for Oracle.

## Privasi lalu lintas antarjaringan

Koneksi dilindungi baik antara Aurora dan aplikasi lokal dan antara Aurora AWS dan sumber daya lain dalam Wilayah yang sama. AWS

### Lalu lintas antara layanan dan aplikasi serta klien on-premise

Anda memiliki dua opsi konektivitas antara jaringan pribadi Anda dan AWS:

- Koneksi AWS Site-to-Site VPN. Untuk informasi selengkapnya, lihat Apa itu AWS Site-to-Site VPN?
- AWS Direct Connect Koneksi. Untuk informasi lebih lanjut, lihat Apa itu AWS Direct Connect?

Anda mendapatkan akses ke Amazon Aurora melalui jaringan dengan AWS menggunakan operasi API yang diterbitkan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan prinsipal IAM. Atau Anda bisa menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara guna menandatangani permintaan.

# Manajemen identitas dan akses untuk Aurora

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. IAMadministrator mengontrol siapa yang dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya Amazon. RDS IAMadalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik


- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Bagaimana Aurora bekerja dengan IAM
- Contoh kebijakan berbasis identitas untuk Amazon Aurora
- AWS kebijakan terkelola untuk Amazon RDS
- Amazon RDS memperbarui kebijakan AWS terkelola
- Pencegahan masalah confused deputy lintas layanan
- Autentikasi basis data IAM
- Memecahkan masalah identitas dan akses Amazon Aurora


## Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan di Aurora.

Pengguna layanan — Jika Anda menggunakan layanan Aurora untuk melakukan pekerjaan Anda, administrator Anda memberi Anda kredensi dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak fitur Aurora untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur di Aurora, lihat. Memecahkan masalah identitas dan akses Amazon Aurora

Administrator layanan - Jika Anda bertanggung jawab atas sumber daya Aurora di perusahaan Anda, Anda mungkin memiliki akses penuh ke Amazon . Tugas Anda adalah menentukan fitur dan

sumber daya Aurora mana yang harus diakses karyawan Anda. Anda kemudian harus mengirimkan permintaan ke administrator Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep basic IAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM Aurora, lihat. Bagaimana Aurora bekerja dengan IAM

Administrator - Jika Anda seorang administrator, Anda mungkin ingin mempelajari detail tentang cara menulis kebijakan untuk mengelola akses ke Aurora. Untuk melihat contoh kebijakan berbasis identitas Aurora yang dapat Anda gunakan, lihat. IAM Contoh kebijakan berbasis identitas untuk Amazon Aurora

## Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai IAM pengguna, atau dengan mengambil peranIAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensi yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (Pusat IAM Identitas), autentikasi masuk tunggal perusahaan Anda, dan kredensi Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas federasi, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan IAM peran. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang menggunakan metode yang disarankan untuk menandatangani permintaan sendiri, lihat Versi AWS Tanda Tangan 4 untuk API permintaan di Panduan IAM Pengguna.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor di Panduan AWS IAM Identity Center Pengguna dan Autentikasi AWS multi-faktor IAM di Panduan Pengguna. IAM

## AWS pengguna root akun

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensi pengguna root di IAMPanduan Pengguna.

## Identitas gabungan

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat IAM Identitas, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat IAM Identitas, lihat Apa itu Pusat IAM Identitas? dalam AWS IAM Identity Center User Guide.

## Pengguna dan grup IAM

IAMPengguna adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya mengandalkan kredensi sementara daripada membuat IAM pengguna yang memiliki kredensi jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensi jangka panjang dengan IAM pengguna, kami sarankan Anda memutar kunci akses. Untuk informasi selengkapnya, lihat Memutar kunci akses secara teratur untuk kasus penggunaan yang memerlukan kredensi jangka panjang di IAMPanduan Pengguna.

IAMGrup adalah identitas yang menentukan kumpulan IAM pengguna. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin bagi beberapa pengguna sekaligus. Grup mempermudah manajemen izin untuk sejumlah besar pengguna sekaligus. Misalnya, Anda dapat memiliki grup bernama IAMAdminsdan memberikan izin grup tersebut untuk mengelola sumber dayaIAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk IAM pengguna di Panduan IAM Pengguna.

Anda dapat mengautentikasi ke cluster DB Anda menggunakan otentikasi IAM database.

IAMotentikasi database bekerja dengan Aurora.Untuk informasi selengkapnya tentang autentikasi ke cluster DB Anda menggunakanIAM, lihatAutentikasi basis data IAM.

## Peran IAM

IAMPeran adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil IAM peran sementara AWS Management Console dengan beralih peran. Anda dapat mengambil peran dengan memanggil AWS CLI atau AWS API operasi atau dengan menggunakan kustomURL. Untuk informasi selengkapnya tentang metode penggunaan peran, lihat Menggunakan IAM peran di Panduan IAM Pengguna.

IAMperan dengan kredensi sementara berguna dalam situasi berikut:

- Izin pengguna sementara — Pengguna dapat mengambil IAM peran untuk sementara mengambil izin yang berbeda untuk tugas tertentu.
- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Ketika identitas terfederasi mengautentikasi, identitas tersebut terhubung dengan peran dan diberi izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat Membuat peran untuk penyedia identitas pihak ketiga (federasi) di Panduan IAM Pengguna. Jika Anda menggunakan Pusat IAM Identitas, Anda mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah diautentikasi, Pusat IAM Identitas menghubungkan izin yang disetel ke peran. IAM Untuk informasi tentang set izin, lihat Set izin dalam Panduan Pengguna AWS IAM Identity Center .

- Akses lintas akun — Anda dapat menggunakan IAM peran untuk memungkinkan seseorang (prinsipal tepercaya) di akun lain mengakses sumber daya di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Perbedaan IAM peran dari kebijakan berbasis sumber daya di Panduan Pengguna. IAM
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Misalnya, saat Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.
  - Sesi akses teruskan — Saat Anda menggunakan IAM pengguna atau peran untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FASmenggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. FASPermintaan hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat FAS permintaan, lihat Meneruskan sesi akses.
  - Peran layanan — Peran layanan adalah IAMperan yang diasumsikan layanan untuk melakukan tindakan atas nama Anda. IAMAdministrator dapat membuat, memodifikasi, dan menghapus peran layanan dari dalamIAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS dalam IAMPanduan Pengguna.
  - Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. IAMAdministrator dapat melihat, tetapi tidak mengedit izin untuk peran terkait layanan.
- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan IAM peran untuk mengelola kredenal sementara untuk aplikasi yang berjalan pada EC2 instance dan membuat AWS CLI atau AWS API meminta. Ini lebih baik untuk menyimpan kunci akses dalam EC2 instance. Untuk menetapkan AWS peran ke EC2 instance dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instance berisi peran dan memungkinkan program yang berjalan pada EC2 instance untuk mendapatkan kredensi

sementara. Untuk informasi selengkapnya, lihat Menggunakan IAM peran untuk memberikan izin ke aplikasi yang berjalan di EC2 instans Amazon di IAMPanduan Pengguna.

Untuk mempelajari apakah akan menggunakan IAM peran, lihat Kapan membuat IAM peran (bukan pengguna) di Panduan IAM Pengguna.

## Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke IAM identitas atau AWS sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika entitas (pengguna root, pengguna, atau IAM peran) membuat permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai JSON dokumen. Untuk informasi selengkapnya tentang struktur dan isi dokumen JSON kebijakan, lihat Ringkasan JSON kebijakan di Panduan IAM Pengguna.

Administrator dapat menggunakan kebijakan untuk menentukan siapa yang memiliki akses ke AWS sumber daya, dan tindakan apa yang dapat mereka lakukan pada sumber daya tersebut. Setiap IAM entitas (set izin atau peran) dimulai tanpa izin. Dengan kata lain, secara default, pengguna tidak dapat melakukan apa pun, termasuk mengubah kata sandinya sendiri. Untuk memberikan izin kepada pengguna untuk melakukan sesuatu, administrator harus melampirkan kebijakan izin kepada pengguna. Atau administrator dapat menambahkan pengguna ke grup yang memiliki izin yang dimaksudkan. Ketika administrator memberikan izin untuk grup, semua pengguna dalam grup tersebut akan diberi izin tersebut.

Kebijakan IAM mendefinisikan izin untuk tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasi. Misalnya, anggaplah Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan itu bisa mendapatkan informasi peran dari AWS Management Console, AWS CLI, atau AWS API.

### Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan JSON izin yang dapat dilampirkan ke identitas, seperti set izin atau peran. Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat IAM kebijakan di Panduan Pengguna. IAM

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan yang dikelola. Kebijakan inline disematkan secara langsung ke satu set izin atau peran. Kebijakan

terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa set izin dan peran di AWS akun Anda. Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan terkelola atau kebijakan sebaris, lihat Memilih antara kebijakan terkelola dan kebijakan sebaris di IAMPanduan Pengguna.

Untuk informasi tentang kebijakan AWS terkelola yang khusus untuk Aurora, lihat. AWS kebijakan terkelola untuk Amazon RDS

## Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis-jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batas izin — Batas izin adalah fitur lanjutan tempat Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas (set izin atau peranIAM). Anda dapat menetapkan batas izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izin tersebut. Kebijakan berbasis sumber daya yang menentukan set izin atau peran di bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya tentang batas izin, lihat Batas izin untuk IAM entitas di IAMPanduan Pengguna.
- Kebijakan kontrol layanan (SCPs) — SCPs adalah JSON kebijakan yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di AWS Organizations. AWS Organizations adalah layanan untuk mengelompokkan dan mengelola beberapa AWS akun secara terpusat yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam organisasi, Anda dapat menerapkan kebijakan kontrol layanan (SCPs) ke salah satu atau semua akun Anda. SCPMembatasi izin untuk entitas di akun anggota, termasuk masing-masing Pengguna root akun AWS. Untuk informasi selengkapnya tentang Organizations danSCPs, lihat Cara SCPs kerja di Panduan AWS Organizations Pengguna.
- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter saat Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara set izin atau kebijakan berbasis identitas peran dan kebijakan sesi tersebut. Izin juga dapat berasal dari kebijakan berbasis sumber daya. Penolakan secara tegas dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya, lihat Kebijakan sesi di Panduan IAM Pengguna.

### Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan IAM Pengguna.

## Bagaimana Aurora bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Aurora, Anda harus memahami IAM fitur apa yang tersedia untuk digunakan dengan Amazon .

Tabel berikut mencantumkan IAM fitur yang dapat Anda gunakan dengan Aurora:

| IAMfitur | Amazon Aurora |
|---|---|
| Kebijakan berbasis identitas | Ya |
| Kebijakan berbasis sumber daya | Tidak |
| Tindakan kebijakan | Ya |
| Sumber daya kebijakan | Ya |
| kunci-kunci persyaratan kebijakan (spesifik layanan) | Ya |
| ACLs | Tidak |
| Kontrol akses berbasis atribut (ABAC) (tag dalam kebijakan) | Ya |
| Kredensial sementara | Ya |
| Teruskan sesi akses | Ya |
| Peran layanan | Ya |
| Peran terkait layanan | Ya |

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja Amazon Aurora dan layanan AWS lainnya, AWS lihat layanan yang berfungsi di IAM Panduan PenggunaIAM.

Topik

- Aurora
- 
- Tindakan kebijakan untuk Aurora
- Sumber daya kebijakan untuk Aurora
- Kunci kondisi kebijakan untuk Aurora
- Daftar kontrol akses (ACLs) di Aurora
- Kontrol akses berbasis atribut (ABAC) dalam kebijakan dengan tag Amazon
- 
- Teruskan sesi akses untuk Aurora
- Peran layanan untuk Aurora
- 

## Aurora

Mendukung kebijakan berbasis identitas: Ya.

Kebijakan berbasis identitas adalah dokumen kebijakan JSON izin yang dapat dilampirkan ke identitas, seperti pengguna, grup IAM pengguna, atau peran. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Menentukan IAM izin khusus dengan kebijakan yang dikelola pelanggan di Panduan Pengguna. IAM

Dengan kebijakan IAM berbasis identitas, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak serta kondisi di mana tindakan diizinkan atau ditolak. Anda tidak dapat menentukan secara spesifik prinsipal dalam sebuah kebijakan berbasis identitas karena prinsipal berlaku bagi pengguna atau peran yang melekat kepadanya. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam JSON kebijakan, lihat referensi elemen IAM JSON kebijakan di Panduan IAM Pengguna.

Untuk melihat contoh kebijakan berbasis identitas Aurora, lihat. Contoh kebijakan berbasis identitas untuk  Amazon Aurora

Mendukung kebijakan berbasis sumber daya: No.

Kebijakan berbasis sumber daya adalah dokumen JSON kebijakan yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan IAM peran dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau entitas IAM di akun lain sebagai penanggung jawab kebijakan berbasis sumber daya. Menambahkan prinsipal akun silang ke kebijakan berbasis sumber daya hanya setengah dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, IAM administrator di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Mereka memberikan izin dengan melampirkan kebijakan berbasis identitas kepada entitas. Namun, jika kebijakan berbasis sumber daya memberikan akses ke principal dalam akun yang sama, tidak diperlukan kebijakan berbasis identitas tambahan. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun IAM di Panduan IAM Pengguna.

## Tindakan kebijakan untuk Aurora

Mendukung tindakan kebijakan: Ya.

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

`Action`Elemen JSON kebijakan menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan AWS API operasi terkait. Ada beberapa pengecualian, seperti tindakan khusus izin yang tidak memiliki operasi yang cocok. API Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Tindakan kebijakan di Aurora menggunakan awalan berikut sebelum tindakan:. `rds:` Misalnya, untuk memberikan izin kepada seseorang untuk mendeskripsikan instans DB dengan RDS

`DescribeDBInstances` API operasi Amazon, Anda menyertakan `rds:DescribeDBInstances` tindakan tersebut dalam kebijakan mereka. Pernyataan kebijakan harus memuat elemen `Action` atau `NotAction`. Aurora mendefinisikan serangkaian tindakannya sendiri yang menggambarkan tugas yang dapat Anda lakukan dengan layanan ini.

Untuk menentukan beberapa tindakan dalam satu pernyataan, pisahkan tindakan dengan koma seperti berikut:

```
"Action": [
      "rds:action1",
      "rds:action2"
```

Anda juga dapat menentukan beberapa tindakan menggunakan wildcard (*). Misalnya, untuk menentukan semua tindakan yang dimulai dengan kata `Describe`, sertakan tindakan berikut.

```
"Action": "rds:Describe*"
```

Untuk melihat daftar tindakan Aurora, lihat Tindakan yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

## Sumber daya kebijakan untuk Aurora

Mendukung sumber daya kebijakan: Ya.

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Resource` JSON kebijakan menentukan objek atau objek yang tindakan tersebut berlaku. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Sebagai praktik terbaik, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Sumber daya instans DB memiliki Amazon Resource Name (ARN) berikut.

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

Untuk informasi selengkapnya tentang formatARNs, lihat Amazon Resource Names (ARNs) dan ruang nama AWS layanan.

Misalnya, untuk menentukan instans `dbtest` DB dalam pernyataan Anda, gunakan yang berikut iniARN.

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

Untuk menentukan semua instans DB milik akun tertentu, gunakan wildcard (*).

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

Beberapa RDS API operasi, seperti untuk membuat sumber daya, tidak dapat dilakukan pada sumber daya tertentu. Jika demikian, gunakan wildcard (*).

```
"Resource": "*"
```

Banyak RDS API operasi Amazon melibatkan banyak sumber daya. Misalnya, `CreateDBInstance` membuat instans DB. Anda dapat menentukan bahwa seorang pengguna harus menggunakan grup keamanan dan grup parameter spesifik saat membuat instans DB. Untuk menentukan beberapa sumber daya dalam pernyataan tunggal, pisahkan ARNs dengan koma.

```
"Resource": [
      "resource1",
      "resource2"
```

Untuk melihat daftar jenis sumber daya Aurora beserta jenisnyaARNs, lihat Sumber Daya yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan. Untuk mempelajari tindakan yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan yang Ditentukan oleh Amazon RDS.

## Kunci kondisi kebijakan untuk Aurora

Mendukung kunci kondisi kebijakan khusus layanan: Ya.

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Condition` (atau blok `Condition`) akan memungkinkan Anda menentukan kondisi yang menjadi dasar suatu pernyataan berlaku. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam sebuah pernyataan, atau beberapa kunci dalam elemen `Condition` tunggal, maka AWS akan mengevaluasinya menggunakan operasi `AND` logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan `OR` operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Misalnya, Anda dapat memberikan izin IAM pengguna untuk mengakses sumber daya hanya jika ditandai dengan nama IAM pengguna mereka. Untuk informasi selengkapnya, lihat elemen IAM kebijakan: variabel dan tag di Panduan IAM Pengguna.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan IAM Pengguna.

Aurora mendefinisikan kumpulan kunci kondisinya sendiri dan juga mendukung penggunaan beberapa kunci kondisi global. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan IAM Pengguna.

Semua RDS API operasi mendukung kunci `aws:RequestedRegion` kondisi.

Untuk melihat daftar kunci kondisi Aurora, lihat Kunci Kondisi untuk Amazon RDS di Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang Ditentukan oleh Amazon RDS.

## Daftar kontrol akses (ACLs) di Aurora

Mendukung daftar kontrol akses (ACLs): Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLsmirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan. JSON

## Kontrol akses berbasis atribut (ABAC) dalam kebijakan dengan tag Amazon

Mendukung tag kontrol akses berbasis atribut (ABAC) dalam kebijakan: Ya

Attribute-based access control (ABAC) adalah strategi otorisasi yang mendefinisikan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke IAM entitas (pengguna atau peran) dan ke banyak AWS sumber daya. Menandai entitas dan sumber daya adalah langkah pertama dari. ABAC Kemudian Anda merancang ABAC kebijakan untuk mengizinkan operasi ketika tag prinsipal cocok dengan tag pada sumber daya yang mereka coba akses.

ABACmembantu dalam lingkungan yang berkembang pesat dan membantu dengan situasi di mana manajemen kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnyaABAC, lihat Menentukan izin dengan ABAC otorisasi di IAMPanduan Pengguna. Untuk melihat tutorial dengan langkah-langkah penyiapanABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) di IAMPanduan Pengguna.

Untuk informasi selengkapnya tentang menandai sumber daya Aurora, lihat. Menentukan kondisi: Menggunakan tag kustom Untuk melihat contoh kebijakan berbasis identitas untuk membatasi akses ke sumber daya berdasarkan tag pada sumber daya tersebut, lihat Berikan izin untuk tindakan atas suatu sumber daya dengan tag tertentu dengan dua nilai yang berbeda.

Mendukung kredensi sementara: Ya.

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensi sementara. Untuk informasi tambahan, termasuk yang Layanan AWS bekerja dengan kredensi sementara, lihat Layanan AWS yang berfungsi IAM di IAMPanduan Pengguna.

Anda menggunakan kredensi sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan link sign-on (SSO) tunggal perusahaan Anda, proses tersebut secara otomatis membuat kredensi sementara. Anda juga akan secara otomatis membuat kredensial sementara

ketika Anda masuk ke konsol sebagai seorang pengguna lalu beralih peran. Untuk informasi selengkapnya tentang beralih peran, lihat Beralih dari pengguna ke IAM peran (konsol) di Panduan IAM Pengguna.

Anda dapat secara manual membuat kredensi sementara menggunakan atau. AWS CLI AWS API Anda kemudian dapat menggunakan kredensi sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensi keamanan sementara di. IAM

## Teruskan sesi akses untuk Aurora

Mendukung sesi akses maju: Ya.

Saat Anda menggunakan IAM pengguna atau peran untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FASmenggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. FASPermintaan hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat FAS permintaan, lihat Meneruskan sesi akses.

## Peran layanan untuk Aurora

Mendukung peran layanan: Ya.

Peran layanan adalah IAMperan yang diasumsikan layanan untuk melakukan tindakan atas nama Anda. IAMAdministrator dapat membuat, memodifikasi, dan menghapus peran layanan dari dalamIAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS dalam IAMPanduan Pengguna.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat merusak fungsionalitas Aurora. Edit peran layanan hanya jika Aurora memberikan panduan untuk melakukannya.

Mendukung peran terkait layanan: Ya.

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. IAMAdministrator dapat melihat, tetapi tidak mengedit izin untuk peran terkait layanan.

Untuk detail tentang menggunakan peran terkait layanan Aurora, lihat. Menggunakan peran terkait layanan untuk Amazon Aurora

## Contoh kebijakan berbasis identitas untuk  Amazon Aurora

Secara default, peran dan kumpulan izin tidak memiliki izin untuk membuat atau mengubah sumber daya Aurora. Mereka juga tidak dapat melakukan tugas menggunakan AWS Management Console, AWS CLI, atau AWS API. Administrator harus membuat kebijakan IAM yang memberikan izin kepada peran atau kumpulan izin untuk menjalankan operasi API tertentu pada sumber daya tertentu yang diperlukan. Administrator kemudian dapat melampirkan kebijakan tersebut ke peran atau kumpulan izin yang memerlukan izin tersebut.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan pada tab JSON dalam Panduan Pengguna IAM.

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol Aurora
- Izin yang diperlukan untuk menggunakan konsol
- Mengizinkan pengguna melihat izin mereka sendiri
- 
- Contoh Kebijakan: Menggunakan kunci kondisi
- Menentukan kondisi: Menggunakan tag kustom
- Berikan izin untuk menandai sumber daya Aurora selama pembuatan

### Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Amazon RDS yang ada di akun Anda. Tindakan ini dikenai biaya untuk Akun AWS Anda. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat Kebijakan yang dikelola AWS atau Kebijakan yang dikelola AWS untuk fungsi tugas dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Kondisi dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat Validasi kebijakan dengan IAM Access Analyzer dalam Panduan Pengguna IAM.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

## Menggunakan konsol Aurora

Untuk mengakses konsol Amazon Aurora, Anda harus memiliki kumpulan izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya Amazon Aurora di Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebagai alternatif, hanya izinkan akses ke tindakan yang cocok dengan operasi API yang sedang Anda coba lakukan.

Untuk memastikan bahwa entitas tersebut masih dapat menggunakan konsol Aurora, lampirkan juga kebijakan terkelola AWS berikut ke entitas.

```
AmazonRDSReadOnlyAccess
```

Untuk informasi selengkapnya, lihat Menambah izin untuk pengguna dalam Panduan Pengguna IAM.

## Izin yang diperlukan untuk menggunakan konsol

Agar pengguna dapat bekerja dengan konsol, pengguna tersebut harus memiliki kumpulan izin minimum. Izin ini memungkinkan pengguna untuk mendeskripsikan sumber daya Amazon Aurora untuk akun AWS mereka dan untuk memberikan informasi terkait lainnya, termasuk keamanan EC2 Amazon dan informasi jaringan.

Jika Anda membuat kebijakan IAM yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk pengguna dengan kebijakan IAM. Untuk memastikan bahwa pengguna masih dapat menggunakan konsol, lampirkan juga kebijakan yang dikelola `AmazonRDSReadOnlyAccess` kepada pengguna, sebagaimana dijelaskan dalam Mengelola akses menggunakan kebijakan.

Anda tidak perlu memperbolehkan izin konsol minimum bagi pengguna yang hanya melakukan panggilan ke AWS CLI atau Amazon RDS API.

Kebijakan berikut memberikan akses penuh ke semua sumber daya Amazon Aurora untuk akun root: AWS

```
AmazonRDSFullAccess
```

## Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Bagian berikut menyajikan contoh kebijakan izin yang memberikan dan membatasi akses ke sumber daya:

Izinkan pengguna untuk membuat instans DB di akun AWS

Berikut ini adalah contoh kebijakan yang memungkinkan akun dengan ID `123456789012` untuk membuat instans DB untuk AWS akun Anda. Kebijakan ini mewajibkan nama instans DB baru dimulai dengan `test`. Instans DB yang baru juga harus menggunakan mesin basis data MySQL dan kelas instans DB `db.t2.micro`. Selain itu, instans DB baru harus menggunakan grup opsi dan grup parameter DB yang dimulai dengan `default`, dan harus menggunakan grup subnet `default`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:db:test*",
            "arn:aws:rds:*:123456789012:og:default*",
            "arn:aws:rds:*:123456789012:pg:default*",
            "arn:aws:rds:*:123456789012:subgrp:default"
         ],
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql",
               "rds:DatabaseClass": "db.t2.micro"
            }
         }
      }
   ]
}
```

Kebijakan ini mencakup pernyataan tunggal yang menentukan izin berikut untuk pengguna:

- Kebijakan ini memungkinkan akun untuk membuat instans DB menggunakan operasi Create DBInstance API (ini juga berlaku untuk create-db-instance AWS CLI perintah dan AWS Management Console).
- Elemen `Resource` menentukan bahwa pengguna dapat melakukan tindakan pada atau dengan sumber daya. Anda menentukan sumber daya menggunakan Amazon Resource Name (ARN). ARN ini mencakup nama layanan yang dimiliki sumber daya (`rds`), AWS Wilayah (*menunjukkan wilayah mana pun dalam contoh ini), nomor AWS akun (`123456789012`adalah nomor akun dalam contoh ini), dan jenis sumber daya. Untuk informasi selengkapnya tentang membuat ARNs, lihatNama Sumber Daya Amazon (ARNs) di Amazon RDS.

  Elemen `Resource` dalam contoh menentukan batasan kebijakan berikut pada sumber daya untuk pengguna:

  - ID instans DB untuk instans DB baru harus dimulai dengan `test` (misalnya, `testCustomerData1`, `test-region2-data`).
  - Grup opsi untuk instans DB baru harus dimulai dengan `default`.
  - Grup parameter DB opsi untuk instans DB baru harus dimulai dengan `default`.
  - Grup subnet untuk instans DB baru harus berupa grup subnet `default`.
- Elemen `Condition` menentukan bahwa mesin DB harus berupa MySQL dan kelas instans DB harus berupa `db.t2.micro`. Elemen `Condition` menentukan kondisi ketika kebijakan harus diberlakukan. Anda dapat menambahkan izin atau batasan tambahan dengan menggunakan elemen `Condition`. Untuk informasi selengkapnya tentang cara menentukan kondisi, lihat Kunci kondisi kebijakan untuk Aurora. Contoh ini menetapkan kondisi `rds:DatabaseEngine` dan `rds:DatabaseClass`. Untuk informasi tentang nilai kondisi yang valid untuk`rds:DatabaseEngine`, lihat daftar di bawah `Engine` parameter di Buat DBInstance. Untuk informasi tentang nilai kondisi yang valid untuk `rds:DatabaseClass`, Lihat Mesin DB yang didukung untuk kelas instans DB.

Kebijakan ini tidak menentukan elemen `Principal` karena dalam kebijakan berbasis identitas, Anda tidak menentukan pengguna utama yang mendapatkan izin. Saat Anda menyematkan kebijakan kepada pengguna, pengguna ini menjadi pengguna utama implisit. Saat Anda menyematkan kebijakan izin pada peran IAM, pengguna utama yang diidentifikasi dalam kebijakan kepercayaan peran tersebut akan mendapatkan izin.

Untuk melihat daftar tindakan Aurora, lihat Tindakan yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

Mengizinkan pengguna melakukan setiap tindakan yang dijelaskan pada sumber daya RDS

Kebijakan izin berikut memberikan izin kepada pengguna untuk menjalankan semua tindakan yang dimulai dengan `Describe`. Tindakan ini menunjukkan informasi tentang sumber daya RDS, seperti instans DB. Karakter wildcard (*) dalam elemen `Resource` menunjukkan bahwa tindakan diperbolehkan untuk semua sumber daya Amazon Aurora yang dimiliki akun tersebut.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

Mengizinkan pengguna membuat instans DB yang menggunakan grup parameter DB dan grup subnet yang telah ditentukan

Kebijakan izin berikut memberikan izin untuk hanya memperbolehkan pengguna membuat instans DB yang harus menggunakan grup parameter DB `mydbpg` dan grup subnet DB `mydbsubnetgroup`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": [
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
```

```
      }
   ]
}
```

Berikan izin untuk tindakan atas suatu sumber daya dengan tag tertentu dengan dua nilai yang berbeda

Anda dapat menggunakan kondisi dalam kebijakan berbasis identitas untuk mengontrol akses ke sumber daya Aurora berdasarkan tag. Kebijakan berikut memungkinkan izin untuk melakukan operasi `CreateDBSnapshot` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
```

```
   ]
}
```

Kebijakan berikut memungkinkan izin untuk melakukan operasi `ModifyDBInstance` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Mencegah pengguna menghapus instans DB

Kebijakan izin berikut memberikan izin untuk mencegah pengguna menghapus instans DB tertentu. Misalnya, Anda mungkin ingin menolak kemampuan untuk menghapus instans DB produksi Anda kepada setiap pengguna yang bukan administrator.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

Menolak semua akses ke sumber daya

Anda juga dapat secara eksplisit menolak akses ke sumber daya. Kebijakan penolakan lebih diutamakan daripada kebijakan yang diizinkan. Kebijakan berikut secara eksplisit menolak kemampuan pengguna untuk mengelola sumber daya:

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
```

```
      }
   ]
}
```

## Contoh Kebijakan: Menggunakan kunci kondisi

Berikut ini adalah contoh cara menggunakan kunci kondisi dalam kebijakan izin IAM Amazon Aurora.

Contoh 1: Memberikan izin untuk membuat instans DB yang menggunakan mesin DB spesifik dan tidak berupa Multi-AZ

Kebijakan berikut menggunakan kunci kondisi RDS dan memungkinkan pengguna membuat instans DB yang menggunakan mesin basis data MySQL saja dan tidak menggunakan MultiAZ. Elemen `Condition` menunjukkan persyaratan bahwa mesin basis data adalah MySQL.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

Contoh 2: Secara eksplisit menolak izin untuk membuat instans DB untuk kelas instans DB tertentu dan membuat instans DB yang menggunakan IOPS yang Tersedia

Kebijakan berikut secara eksplisit menolak izin untuk membuat instans DB yang menggunakan kelas instans DB `r3.8xlarge` dan `m4.10xlarge`, yang merupakan kelas instans DB terbesar dan termahal. Kebijakan ini juga mencegah pengguna membuat instans DB yang menggunakan IOPS yang Tersedia, yang menimbulkan biaya tambahan.

Izin yang secara tegas menolak lebih diprioritaskan daripada izin lain yang diberikan. Ini memastikan bahwa identitas tidak akan secara kebetulan mendapatkan izin yang tidak pernah ingin Anda berikan.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
```

```
}
```

Contoh 3: Membatasi kumpulan kunci dan nilai tag yang dapat digunakan untuk menandai sumber daya

Kebijakan berikut menggunakan kunci kondisi RDS dan memungkinkan penambahan tag dengan kunci `stage` untuk ditambahkan ke sumber daya dengan nilai `test`, `qa`, dan `production`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource": "*",
         "Condition": {
            "streq": {
               "rds:req-tag/stage": [
                  "test",
                  "qa",
                  "production"
               ]
            }
         }
      }
   ]
}
```

## Menentukan kondisi: Menggunakan tag kustom

Amazon Aurora mendukung penentuan kondisi dalam kebijakan IAM menggunakan tag kustom.

Sebagai contoh, misalkan Anda menambahkan tag bernama `environment` ke instans DB Anda dengan nilai seperti `beta`, `staging`, `production`, dan sebagainya. Jika melakukannya, Anda

dapat membuat kebijakan yang membatasi pengguna tertentu pada instans DB berdasarkan nilai tag `environment`.

> **Note**
>
> ID tag kustom bersifat peka huruf besar-kecil.

Tabel berikut mencantumkan ID tag RDS yang dapat digunakan pada elemen `Condition`.

| ID tag RDS | Berlaku untuk |
|---|---|
| `db-tag` | Instans DB, termasuk replika baca |
| `snapshot-tag` | Snapshot DB |
| `ri-tag` | Instans DB terpesan |
| `og-tag` | Grup opsi DB |
| `pg-tag` | Grup parameter DB |
| `subgrp-tag` | Grup subnet DB |
| `es-tag` | Langganan peristiwa |
| `cluster-tag` | Klaster DB |
| `cluster-pg-tag` | Grup parameter klaster DB |
| `cluster-snapshot-tag` | Snapshot klaster DB |

Sintaks untuk kondisi tag kustom adalah sebagai berikut:

```
"Condition":{"StringEquals":{"rds:rds-tag-identifier/tag-name":
["value"]} }
```

Misalnya, elemen `Condition` berikut berlaku untuk instans DB dengan tag bernama `environment` dan nilai tag `production`.

```
"Condition":{"StringEquals":{"rds:db-tag/environment": ["production"]} }
```

Untuk informasi tentang membuat tag, lihat Menandai sumber daya Amazon Aurora dan Amazon RDS.

**Important**

Jika Anda mengelola akses ke sumber daya RDS Anda menggunakan pemberian tag, sebaiknya Anda mengamankan akses ke tag untuk sumber daya RDS Anda. Anda dapat mengelola akses ke tag dengan membuat kebijakan untuk tindakan `AddTagsToResource` dan `RemoveTagsFromResource`. Misalnya, kebijakan berikut menolak kemampuan pengguna untuk menambahkan atau menghapus tag untuk semua sumber daya. Anda kemudian dapat membuat kebijakan untuk mengizinkan pengguna tertentu menambahkan atau menghapus tag.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
         "Action":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"*"
      }
   ]
}
```

Untuk melihat daftar tindakan Aurora, lihat Tindakan yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

Contoh kebijakan: Menggunakan tag kustom

Contoh berikut menunjukkan cara menggunakan tag kustom dalam kebijakan izin IAM Amazon Aurora. Untuk informasi lebih lanjut tentang cara menambahkan tag ke sumber daya Amazon Aurora, lihat Nama Sumber Daya Amazon (ARNs) di Amazon RDS.

> **Note**
>
> Semua contoh menggunakan wilayah us-west-2 dan berisi akun fiktif. IDs

Contoh 1: Memberikan izin untuk tindakan pada sumber daya dengan tag tertentu dengan dua nilai yang berbeda

Kebijakan berikut memungkinkan izin untuk melakukan operasi `CreateDBSnapshot` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
```

```
}
```

Kebijakan berikut memungkinkan izin untuk melakukan operasi `ModifyDBInstance` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":" [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Contoh 2: Secara eksplisit menolak izin untuk membuat instans DB yang menggunakan grup parameter DB yang ditentukan

Kebijakan berikut secara eksplisit menolak izin untuk membuat instans DB yang menggunakan grup parameter DB dengan nilai tag spesifik. Anda dapat menerapkan kebijakan ini jika Anda mengharuskan grup parameter DB yang dibuat pengguna tertentu selalu digunakan saat membuat instans DB. Kebijakan yang menggunakan `Deny` paling sering digunakan untuk membatasi akses yang diberikan oleh kebijakan yang lebih luas.

Izin yang secara tegas menolak lebih diprioritaskan daripada izin lain yang diberikan. Ini memastikan bahwa identitas tidak akan secara kebetulan mendapatkan izin yang tidak pernah ingin Anda berikan.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

Contoh 3: Memberikan izin untuk tindakan pada instans DB dengan nama instans yang diawali dengan nama pengguna

Kebijakan berikut memungkinkan izin untuk memanggil API apa pun (kecuali untuk `AddTagsToResource` atau `RemoveTagsFromResource`) pada instans DB yang memiliki nama instans DB yang diawali dengan nama pengguna dan memiliki tag bernama `stage` yang sama dengan `devo` atau yang tidak memiliki tag bernama `stage`.

Baris `Resource` dalam kebijakan mengidentifikasi sumber daya berdasarkan Amazon Resource Name (ARN). Untuk informasi selengkapnya tentang penggunaan ARNs dengan sumber daya Amazon Aurora, lihat. Nama Sumber Daya Amazon (ARNs) di Amazon RDS

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
               "rds:db-tag/stage":"devo"
            }
         }
      }
   ]
}
```

## Berikan izin untuk menandai sumber daya Aurora selama pembuatan

Beberapa operasi API RDS memungkinkan Anda menentukan tag saat membuat sumber daya. Anda dapat menggunakan tanda sumber daya untuk menerapkan pengendalian berbasis atribut (ABAC). Untuk informasi lebih lanjut, lihat Untuk apa ABAC? AWS dan Mengontrol akses ke AWS sumber daya menggunakan tag.

Untuk memungkinkan pengguna menandai sumber daya pada pembuatan, mereka harus memiliki izin untuk menggunakan tindakan yang membuat sumber daya, seperti . Jika tag ditentukan dalam tindakan buat, RDS melakukan otorisasi tambahan pada `rds:AddTagsToResource` tindakan untuk memverifikasi apakah pengguna memiliki izin untuk membuat tag. Oleh karena itu, para pengguna juga harus memiliki izin eksplisit untuk menggunakan tindakan `rds:AddTagsToResource`.

Dalam definisi kebijakan IAM untuk `rds:AddTagsToResource` tindakan, Anda dapat menggunakan kunci `aws:RequestTag` kondisi untuk mewajibkan tag dalam permintaan untuk menandai sumber daya.

Misalnya, kebijakan berikut memungkinkan pengguna untuk membuat instance DB dan menerapkan tag selama pembuatan instans DB, tetapi hanya dengan kunci tag tertentu (`environment`atau`project`):

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBInstance"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
           "Action": [
               "rds:AddTagsToResource"
           ],
           "Resource": "*",
           "Condition": {
               "StringEquals": {
                   "aws:RequestTag/environment": ["production", "development"],
                   "aws:RequestTag/project": ["dataanalytics", "webapp"]
               },
               "ForAllValues:StringEquals": {
                   "aws:TagKeys": ["environment", "project"]
               }
           }
       }
   ]
}
```

Kebijakan ini menolak permintaan instans create DB yang menyertakan tag selain `project` tag `environment` atau, atau yang tidak menentukan salah satu tag ini. Selain itu, pengguna harus menentukan nilai untuk tag yang cocok dengan nilai yang diizinkan dalam kebijakan.

Kebijakan berikut memungkinkan pengguna untuk membuat cluster DB dan menerapkan tag apa pun selama pembuatan kecuali `environment=prod` tag:

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBCluster"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
           "Action": [
               "rds:AddTagsToResource"
           ],
           "Resource": "*",
           "Condition": {
               "StringNotEquals": {
                   "aws:RequestTag/environment": "prod"
               }
           }
       }
   ]
}
```

Tindakan API RDS yang didukung untuk menandai pada pembuatan

Tindakan API RDS berikut mendukung penandaan saat Anda membuat sumber daya. Untuk tindakan ini, Anda dapat menentukan tag saat membuat sumber daya:

- `CreateBlueGreenDeployment`
- `CreateCustomDBEngineVersion`
- `CreateDBCluster`
- `CreateDBClusterEndpoint`
- `CreateDBClusterParameterGroup`
- `CreateDBClusterSnapshot`
- `CreateDBInstance`

- CreateDBInstanceReadReplica
- CreateDBParameterGroup
- CreateDBProxy
- CreateDBProxyEndpoint
- CreateDBSecurityGroup
- CreateDBShardGroup
- CreateDBSnapshot
- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- CreateIntegration
- CreateOptionGroup
- CreateTenantDatabase
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CopyDBParameterGroup
- CopyDBSnapshot
- CopyOptionGroup
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime
- PurchaseReservedDBInstancesOffering

Jika Anda menggunakan API AWS CLI atau untuk membuat sumber daya dengan tag, `Tags` parameter tersebut digunakan untuk menerapkan tag ke sumber daya selama pembuatan.

Untuk tindakan API ini, jika penandaan gagal, sumber daya tidak dibuat, dan permintaan gagal dengan kesalahan. Ini memastikan bahwa sumber daya dibuat dengan tag atau tidak dibuat sama sekali, mencegah sumber daya dibuat tanpa tag yang dimaksudkan.

# AWS kebijakan terkelola untuk Amazon RDS

Untuk menambahkan izin ke set dan peran izin, lebih mudah menggunakan kebijakan AWS terkelola daripada menulis kebijakan sendiri. Butuh waktu dan keahlian untuk membuat kebijakan terkelola IAM pelanggan yang hanya memberi tim Anda izin yang mereka butuhkan. Untuk memulai dengan cepat, Anda dapat menggunakan kebijakan AWS terkelola kami. Kebijakan ini mencakup kasus penggunaan umum dan tersedia di Akun AWS Anda. Untuk informasi selengkapnya tentang kebijakan AWS AWS terkelola, lihat kebijakan terkelola di Panduan IAM Pengguna.

Layanan AWS memelihara dan memperbarui kebijakan AWS terkelola. Anda tidak dapat mengubah izin dalam kebijakan AWS terkelola. Layanan terkadang menambahkan izin tambahan ke kebijakan AWS terkelola untuk mendukung fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Layanan kemungkinan besar akan memperbarui kebijakan AWS terkelola saat fitur baru diluncurkan atau saat operasi baru tersedia. Layanan tidak menghapus izin dari kebijakan AWS terkelola, sehingga pembaruan kebijakan tidak merusak izin yang ada.

Selain itu, AWS mendukung kebijakan terkelola untuk fungsi pekerjaan yang mencakup beberapa layanan. Misalnya, kebijakan `ReadOnlyAccess` AWS terkelola menyediakan akses hanya-baca ke semua Layanan AWS dan sumber daya. Saat layanan meluncurkan fitur baru, AWS tambahkan izin hanya-baca untuk operasi dan sumber daya baru. Untuk daftar dan deskripsi kebijakan fungsi pekerjaan, lihat kebijakan AWS terkelola untuk fungsi pekerjaan di Panduan IAM Pengguna.

Topik

- AWS kebijakan terkelola: A mazonRDSRead OnlyAccess
- AWS kebijakan terkelola: mazonRDSFull Akses
- AWS kebijakan terkelola: A mazonRDSData FullAccess
- AWS kebijakan terkelola: A mazonRDSEnhanced MonitoringRole
- AWS kebijakan terkelola: A mazonRDSPerformance InsightsReadOnly
- AWS kebijakan terkelola: A mazonRDSPerformance InsightsFullAccess
- AWS kebijakan terkelola: A mazonRDSDirectory ServiceAccess
- AWS kebijakan terkelola: A mazonRDSService RolePolicy
- AWS kebijakan terkelola: A mazonRDSPreview ServiceRolePolicy
- AWS kebijakan terkelola: A mazonRDSBeta ServiceRolePolicy

## AWS kebijakan terkelola: A mazonRDSRead OnlyAccess

Kebijakan ini memungkinkan akses hanya-baca ke Amazon RDS melalui. AWS Management Console

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds`— Memungkinkan kepala sekolah untuk mendeskripsikan RDS sumber daya Amazon dan mencantumkan tag untuk sumber daya Amazon. RDS
- `cloudwatch`— Memungkinkan kepala sekolah untuk mendapatkan statistik metrik Amazon CloudWatch .
- `ec2` – Mengizinkan pengguna utama mendeskripsikan Zona Ketersediaan dan sumber daya jaringan.
- `logs`— Memungkinkan prinsipal untuk menggambarkan aliran CloudWatch log Log dari grup log, dan mendapatkan CloudWatch peristiwa log Log.
- `devops-guru`— Memungkinkan prinsipal untuk mendeskripsikan sumber daya yang memiliki cakupan Amazon DevOps Guru, yang ditentukan baik oleh nama CloudFormation tumpukan atau tag sumber daya.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat A mazonRDSRead OnlyAccess di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: mazonRDSFull Akses

Kebijakan ini menyediakan akses penuh ke Amazon RDS melalui AWS Management Console.

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds`— Memungkinkan kepala sekolah akses penuh ke Amazon. RDS
- `application-autoscaling` – Mengizinkan pengguna utama mendeskripsikan dan mengelola target dan kebijakan penskalaan Application Auto Scaling.
- `cloudwatch`— Memungkinkan kepala sekolah mendapatkan statika CloudWatch metrik dan mengelola alarm. CloudWatch

- `ec2` – Mengizinkan pengguna utama mendeskripsikan Zona Ketersediaan dan sumber daya jaringan.
- `logs`— Memungkinkan prinsipal untuk menggambarkan aliran CloudWatch log Log dari grup log, dan mendapatkan CloudWatch peristiwa log Log.
- `outposts`— Memungkinkan prinsipal untuk mendapatkan AWS Outposts jenis instance.
- `pi` – Mengizinkan pengguna utama untuk mendapatkan metrik Wawasan Performa.
- `sns`— Memungkinkan prinsipal untuk langganan dan topik Amazon Simple Notification Service (AmazonSNS), dan untuk mempublikasikan pesan Amazon. SNS
- `devops-guru`— Memungkinkan prinsipal untuk mendeskripsikan sumber daya yang memiliki cakupan Amazon DevOps Guru, yang ditentukan baik oleh nama CloudFormation tumpukan atau tag sumber daya.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat mazonRDSFullAkses di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSData FullAccess

Kebijakan ini memungkinkan akses penuh untuk menggunakan Data API dan editor kueri pada Aurora Serverless cluster tertentu Akun AWS. Kebijakan ini memungkinkan Akun AWS untuk mendapatkan nilai rahasia dari AWS Secrets Manager.

Anda dapat melampirkan `AmazonRDSDataFullAccess` kebijakan ke IAM identitas Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `dbqms` – Mengizinkan pengguna utama mengakses, menghapus, mendeskripsikan, dan memperbarui kueri. Layanan Metadata Kueri Basis Data (`dbqms`) adalah layanan khusus internal. Ini memberikan kueri terbaru dan tersimpan Anda untuk editor kueri di AWS Management Console untuk beberapa Layanan AWS, termasuk AmazonRDS.
- `rds-data`— Memungkinkan prinsipal untuk menjalankan SQL pernyataan pada database. Aurora Serverless
- `secretsmanager`— Memungkinkan kepala sekolah untuk mendapatkan nilai rahasia dari. AWS Secrets Manager

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat AmazonRDSData FullAccess di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSEnhanced MonitoringRole

Kebijakan ini menyediakan akses ke Amazon CloudWatch Logs untuk Amazon RDS Enhanced Monitoring.

Detail izin

Kebijakan ini mencakup izin berikut:

- `logs`— Memungkinkan prinsipal untuk membuat grup CloudWatch log Log dan kebijakan retensi, dan untuk membuat dan mendeskripsikan aliran CloudWatch log log dari grup log. Hal ini juga memungkinkan prinsipal untuk menempatkan dan mendapatkan peristiwa CloudWatch log Log.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat AmazonRDSEnhanced MonitoringRole di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSPerformance InsightsReadOnly

Kebijakan ini menyediakan akses hanya-baca ke Amazon Performance RDS Insights untuk instans Amazon DB dan kluster RDS Amazon Aurora DB.

Kebijakan ini kini mencakup `Sid` (ID pernyataan) sebagai pengidentifikasi pernyataan kebijakan.

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds`— Memungkinkan kepala sekolah untuk menggambarkan instans Amazon RDS DB dan cluster Amazon Aurora DB.
- `pi`— Memungkinkan kepala sekolah melakukan panggilan ke Amazon Performance Insights dan mengakses metrik RDS Performance InsightsAPI.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat AmazonRDSPerformance InsightsReadOnly di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSPerformance InsightsFullAccess

Kebijakan ini menyediakan akses penuh ke Amazon RDS Performance Insights untuk instans Amazon RDS DB dan kluster Amazon Aurora DB.

Kebijakan ini kini mencakup `Sid` (ID pernyataan) sebagai pengidentifikasi pernyataan kebijakan.

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds`— Memungkinkan kepala sekolah untuk menggambarkan instans Amazon RDS DB dan cluster Amazon Aurora DB.
- `pi`— Memungkinkan prinsipal melakukan panggilan ke Amazon Performance Insights, serta membuatAPI, melihat, dan menghapus laporan analisis RDS kinerja.
- `cloudwatch`— Memungkinkan kepala sekolah untuk membuat daftar semua metrik Amazon, dan mendapatkan CloudWatch data metrik dan statistik.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat A mazonRDSPerformance InsightsFullAccess di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSDirectory ServiceAccess

Kebijakan ini memungkinkan Amazon RDS untuk melakukan panggilan ke AWS Directory Service.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ds`— Memungkinkan kepala sekolah untuk mendeskripsikan AWS Directory Service direktori dan mengontrol otorisasi ke direktori. AWS Directory Service

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat A mazonRDSDirectory ServiceAccess di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSService RolePolicy

Anda tidak dapat melampirkan `AmazonRDSServiceRolePolicy` kebijakan ke IAM entitas Anda. Kebijakan ini dilampirkan pada peran terkait layanan yang memungkinkan Amazon RDS melakukan

tindakan atas nama Anda. Untuk informasi selengkapnya, lihat Izin peran terkait layanan untuk Amazon Aurora.

## AWS kebijakan terkelola: A mazonRDSPreview ServiceRolePolicy

Anda tidak harus melampirkan `AmazonRDSPreviewServiceRolePolicy` ke IAM entitas Anda. Kebijakan ini dilampirkan ke peran terkait layanan yang memungkinkan Amazon RDS memanggil AWS layanan atas nama sumber daya RDS DB Anda. Untuk informasi selengkapnya, lihat Peran terkait layanan untuk Amazon RDS Preview.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ec2`- Memungkinkan kepala sekolah untuk menggambarkan Availability Zone dan sumber daya jaringan.
- `secretsmanager`— Memungkinkan kepala sekolah untuk mendapatkan nilai rahasia dari. AWS Secrets Manager
- `cloudwatch`, `logs` - Memungkinkan Amazon RDS mengunggah metrik dan log instans DB CloudWatch melalui CloudWatch agen.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat A mazonRDSData FullAccess di Panduan Referensi Kebijakan AWS Terkelola.

## AWS kebijakan terkelola: A mazonRDSBeta ServiceRolePolicy

Anda tidak harus melampirkan `AmazonRDSBetaServiceRolePolicy` ke IAM entitas Anda. Kebijakan ini dilampirkan ke peran terkait layanan yang memungkinkan Amazon RDS memanggil AWS layanan atas nama sumber daya RDS DB Anda. Untuk informasi selengkapnya, lihat Izin peran terkait layanan untuk Amazon RDS Beta.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ec2`- Memungkinkan Amazon RDS melakukan operasi pencadangan pada instans DB yang menyediakan kemampuan point-in-time pemulihan.
- `secretsmanager`- Memungkinkan Amazon RDS mengelola rahasia khusus instans DB yang dibuat oleh AmazonRDS.

- `cloudwatch`, `logs` - Memungkinkan Amazon RDS mengunggah metrik dan log instans DB CloudWatch melalui CloudWatch agen.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen JSON kebijakan, lihat AmazonRDSBeta ServiceRolePolicy di Panduan Referensi Kebijakan AWS Terkelola.

# Amazon RDS memperbarui kebijakan AWS terkelola

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk Amazon RDS sejak layanan ini mulai melacak perubahan ini. Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan di umpan RSS di halaman Riwayat dokumen Amazon RDS.

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola: AmazonRDSPreview ServiceRolePolicy – Pembaruan ke kebijakan yang sudah ada | Amazon RDS menghapus `sns:Publish` izin `AmazonRDSPreviewServiceRolePolicy` dari peran `AWSServiceRoleForRDSPreview` terkait layanan. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: AmazonRDSPreview ServiceRolePolicy. | Agustus 7, 2024 |
| AWS kebijakan terkelola: AmazonRDSBeta ServiceRolePolicy – Pembaruan ke kebijakan yang sudah ada | Amazon RDS menghapus `sns:Publish` izin `AmazonRDSBetaServiceRolePolicy` dari peran `AWSServiceRoleForRDSBeta` terkait layanan. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: AmazonRDSBeta ServiceRolePolicy. | Agustus 7, 2024 |
| AWS kebijakan terkelola: AmazonRDSService RolePolicy – Pembaruan ke kebijakan yang sudah ada | Amazon RDS menghapus `sns:Publish` izin `AmazonRDSServiceRolePolicy` dari peran `AWSServiceRoleForRDS` terkait layanan. Untuk | Juli 2, 2024 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| | informasi selengkapnya, lihat AWS kebijakan terkelola: AmazonRDSService RolePolicy. | |
| AWS kebijakan terkelola untuk Amazon RDS – Pembaruan ke kebijakan yang sudah ada | Amazon RDS menambahkan izin baru ke peran `AWSServiceRoleForRDSCustom` terkait layanan untuk memungkinkan RDS Custom for SQL Server memodifikasi jenis instans host database yang mendasarinya. `AmazonRDSCustomServiceRolePolicy` RDS juga menambahkan `ec2:DescribeInstanceTypes` izin untuk mendapatkan informasi tipe instance untuk host database. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola untuk Amazon RDS. | April 8, 2024 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola untuk Amazon RDS – Kebijakan baru | Amazon RDS menambahkan kebijakan terkelola baru yang diberi nama `AmazonRDSCustomInstanceProfileRolePolicy` untuk memungkinkan RDS Custom melakukan tindakan otomatisasi dan tugas manajemen database melalui profil EC2 instans. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola untuk Amazon RDS. | Februari 27, 2024 |
| Izin peran terkait layanan untuk Amazon Aurora – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan pernyataan baru IDs ke `AmazonRDSServiceRolePolicy` peran `AWSServiceRoleForRDS` terkait layanan.<br><br>Untuk informasi selengkapnya, lihat Izin peran terkait layanan untuk Amazon Aurora. | Januari 19, 2024 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola untuk Amazon RDS – Pembaruan ke kebijakan yang ada | Kebijakan yang dikelola `AmazonRDSPerformanceInsightsReadOnly` dan `AmazonRDSPerformanceInsightsFullAccess` kini menyertakan `Sid` (ID pernyataan) sebagai ID dalam pernyataan kebijakan.<br><br>Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: A mazonRDSPerformance InsightsReadOnly dan AWS kebijakan terkelola: A mazonRDSPerformance InsightsFullAccess | 23 Oktober 2023 |
| AWS kebijakan terkelola untuk Amazon RDS – Pembaruan ke kebijakan yang sudah ada | Amazon RDS menambahkan izin baru ke kebijakan yang dikelola `AmazonRDSFullAccess`. Kebijakan ini memungkinkan Anda membuat, melihat, dan menghapus laporan analisis performa selama periode tertentu.<br><br>Untuk informasi selengkapnya tentang cara mengonfigurasi kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa | 17 Agustus 2023 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola untuk Amazon RDS – Kebijakan baru dan pembaruan ke kebijakan yang ada | Amazon RDS menambahkan izin baru ke kebijakan yang dikelola `AmazonRDSPerformanceInsightsReadOnly` dan kebijakan terkelola baru bernama `AmazonRDSPerformanceInsightsFullAccess`. Izin ini memungkinkan Anda menganalisis Wawasan Performa selama periode tertentu, melihat hasil analisis beserta rekomendasinya, dan menghapus laporan.<br><br>Untuk informasi selengkapnya tentang cara mengonfigurasi kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa | 16 Agustus 2023 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola untuk Amazon RDS – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan CloudWatch namespace `ListMetrics` Amazon baru ke dan. `AmazonRDSFullAccess` `AmazonRDSReadOnlyAccess`<br><br>Nama ruang ini diperlukan untuk Amazon RDS untuk mencantumkan metrik penggunaan sumber daya tertentu.<br><br>Untuk informasi selengkapnya, lihat Ringkasan mengelola izin akses ke CloudWatch sumber daya Anda di Panduan CloudWatch Pengguna Amazon. | 4 April 2023 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| Izin peran terkait layanan untuk Amazon Aurora – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan izin baru ke `AmazonRDSServiceRolePolicy` peran `AWSServiceRoleForRDS` terkait layanan untuk integrasi . AWS Secrets Manager RDS memerlukan integrasi dengan Secrets Manager untuk mengelola kata sandi pengguna utama di Secrets Manager. Rahasianya menggunakan konvensi penamaan yang dicadangkan dan membatasi pembaruan pelanggan.<br><br>Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. | 22 Desember 2022 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola untuk Amazon RDS – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan izin baru ke `AmazonRDSFullAccess` dan kebijakan `AmazonRDSReadOnlyAccess` terkelola untuk memungkinkan Anda mengaktifkan Amazon DevOps Guru di konsol RDS. Izin ini diperlukan untuk memeriksa apakah DevOps Guru dihidupkan.<br><br>Untuk informasi selengkapnya, lihat Mengkonfigurasi kebijakan IAM akses untuk DevOps Guru for RDS. | 19 Desember 2022 |
| Izin peran terkait layanan untuk Amazon Aurora – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan CloudWatch namespace Amazon baru ke for. `AmazonRDSPreviewServiceRolePolicy` `PutMetricData`<br><br>Namespace ini mewajibkan Amazon RDS menerbitkan metrik penggunaan sumber daya.<br><br>Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon. | 7 Juni 2022 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| Izin peran terkait layanan untuk Amazon Aurora – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan CloudWatch namespace Amazon baru ke for. `AmazonRDSBetaServiceRolePolicy` `PutMetricData`<br><br>Namespace ini mewajibkan Amazon RDS menerbitkan metrik penggunaan sumber daya.<br><br>Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon. | 7 Juni 2022 |
| Izin peran terkait layanan untuk Amazon Aurora – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan CloudWatch namespace Amazon baru ke for. `AWSServiceRoleForRDS` `PutMetricData`<br><br>Namespace ini mewajibkan Amazon RDS menerbitkan metrik penggunaan sumber daya.<br><br>Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon. | 22 April 2022 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| AWS kebijakan terkelola untuk Amazon RDS – Kebijakan baru | Amazon RDS menambahkan kebijakan terkelola baru yang diberi nama `AmazonRDSPerformanceInsightsReadOnly` untuk mengizinkan Amazon RDS memanggil AWS layanan atas nama instans DB Anda.<br><br>Untuk informasi selengkapnya tentang cara mengonfigurasi kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa | 10 Maret 2022 |

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| Izin peran terkait layanan untuk Amazon Aurora – Pembaruan ke kebijakan yang ada | Amazon RDS menambahkan CloudWatch ruang nama Amazon baru ke for. `AWSServiceRoleForRDS` `PutMetricData`<br><br>Ruang nama ini diperlukan untuk Amazon DocumentDB (dengan kompatibilitas MongoDB) dan Amazon Neptune untuk menerbitkan metrik. CloudWatch<br><br>Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon. | 4 Maret 2022 |
| Amazon RDS memulai pelacakan perubahan | Amazon RDS mulai melacak perubahan untuk kebijakan yang AWS dikelola. | 26 Oktober 2021 |

# Pencegahan masalah confused deputy lintas layanan

Masalah confused deputy adalah masalah keamanan saat entitas yang tidak memiliki izin untuk melakukan suatu tindakan dapat memaksa entitas yang lebih berhak untuk melakukan tindakan tersebut. Pada tahun AWS, peniruan lintas layanan dapat mengakibatkan masalah wakil yang membingungkan.

Peniruan identitas lintas layanan dapat terjadi ketika satu layanan (layanan panggilan) memanggil layanan lain (layanan yang dipanggil). Layanan panggilan dapat dimanipulasi agar menggunakan izinnya untuk bertindak pada sumber daya pelanggan lain dengan cara yang seharusnya tidak dilakukannya kecuali bila memiliki izin untuk mengakses. Untuk mencegah hal ini, AWS sediakan alat yang dapat membantu Anda melindungi data Anda untuk semua layanan dengan prinsip layanan yang telah diberikan akses ke sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Masalah confused deputy di Panduan Pengguna IAM.

Untuk membatasi izin yang diberikan Amazon RDS ke layanan lain untuk sumber daya tertentu, sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` di kebijakan sumber daya.

Dalam beberapa kasus, nilai `aws:SourceArn` tidak berisi ID akun, misalnya saat Anda menggunakan Amazon Resource Name (ARN) untuk bucket Amazon S3. Dalam kasus ini, pastikan untuk menggunakan kedua kunci konteks kondisi global untuk membatasi izin. Dalam beberapa kasus, Anda menggunakan kunci konteks kondisi global dan nilai `aws:SourceArn` yang berisi ID akun. Dalam hal ini, pastikan bahwa nilai `aws:SourceAccount` dan akun dalam nilai `aws:SourceArn` menggunakan ID akun yang sama ketika digunakan dalam pernyataan yang sama. Jika Anda ingin hanya satu sumber daya yang akan dikaitkan dengan akses lintas layanan, gunakan `aws:SourceArn`. Jika Anda ingin mengizinkan sumber daya apa pun di AWS akun yang ditentukan dikaitkan dengan penggunaan lintas layanan, gunakan`aws:SourceAccount`.

Pastikan nilai `aws:SourceArn` adalah ARN untuk jenis sumber daya Amazon RDS. Untuk informasi selengkapnya, lihat Nama Sumber Daya Amazon (ARNs) di Amazon RDS.

Cara paling efektif untuk melindungi dari masalah "confused deputy" adalah dengan menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN lengkap sumber daya. Dalam beberapa kasus, Anda mungkin tidak mengetahui ARN lengkap sumber daya atau mungkin Anda menentukan beberapa sumber daya. Dalam hal ini, gunakan kunci kondisi konteks global `aws:SourceArn` dengan wildcard (*) untuk bagian ARN yang tidak diketahui. Contohnya adalah `arn:aws:rds:*:123456789012:*`.

Contoh berikut menunjukkan cara Anda dapat menggunakan `aws:SourceArn` dan kunci konteks kondisi global `aws:SourceAccount` di Amazon RDS untuk mencegah masalah confused deputy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Untuk contoh kebijakan lainnya yang menggunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount`, lihat bagian berikut:

- Memberikan izin untuk menerbitkan pemberitahuan ke topik Amazon SNS
- Menyiapkan akses ke bucket Amazon S3 (Impor PostgreSQL)
- Menyiapkan akses ke bucket Amazon S3 (Ekspor PostgreSQL)

# Autentikasi basis data IAM

Anda dapat mengautentikasi ke cluster DB Anda menggunakan otentikasi database AWS Identity and Access Management (IAM). Autentikasi basis data IAM berfungsi dengan Aurora MySQL, dan Aurora PostgreSQL. Dengan metode autentikasi ini, Anda tidak perlu menggunakan kata sandi saat menghubungkan dengan klaster basis data. Sebagai gantinya, gunakan token autentikasi.

Token autentikasi adalah string karakter unik yang dihasilkan Amazon Aurora sesuai permintaan. Token otentikasi dihasilkan menggunakan AWS Signature Version 4. Setiap token memiliki masa pakai 15 menit. Anda tidak perlu menyimpan kredensial pengguna di basis data karena autentikasi dikelola secara eksternal menggunakan IAM. Anda juga masih dapat menggunakan autentikasi basis data standar. Token hanya digunakan untuk autentikasi dan tidak memengaruhi sesi setelah dibuat.

Autentikasi basis data IAM memberikan manfaat berikut:

- Lalu lintas jaringan ke dan dari basis data dienkripsi menggunakan Lapisan Soket Aman (SSL) atau Keamanan Lapisan Pengangkutan (TLS). Untuk informasi selengkapnya tentang penggunaan SSL/TLS Amazon Aurora, lihat. Menggunakan SSL/TLS untuk mengenkripsi koneksi ke
- Anda dapat menggunakan IAM untuk mengelola akses ke sumber daya basis data Anda secara terpusat, bukan mengelola akses satu per satu pada setiap klaster DB.
- Untuk aplikasi yang berjalan di Amazon EC2, Anda dapat menggunakan kredensil profil khusus untuk EC2 instans Anda untuk mengakses database Anda, bukan kata sandi, untuk keamanan yang lebih baik.

Secara umum, pertimbangkan untuk menggunakan autentikasi basis data IAM ketika aplikasi Anda membuat kurang dari 200 koneksi per detik, dan Anda tidak ingin mengelola nama pengguna dan kata sandi secara langsung dalam kode aplikasi Anda.

Driver Amazon Web Services (AWS) JDBC mendukung autentikasi database IAM. Untuk informasi selengkapnya, lihat Plugin Autentikasi AWS IAM di repositori Driver Amazon Web Services (AWS) JDBC. GitHub

Driver Python Amazon Web Services (AWS) mendukung otentikasi database IAM. Untuk informasi selengkapnya, lihat Plugin Autentikasi AWS IAM di repositori Amazon Web Services ()AWS Python Driver. GitHub

Arahkan topik berikut untuk mempelajari proses menyetel IAM untuk otentikasi DB:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM

- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM
- Menghubungkan ke klaster DB menggunakan autentikasi IAM

## Ketersediaan wilayah dan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang ketersediaan versi dan Wilayah dengan autentikasi basis data IAM dan Aurora, lihat Daerah yang Didukung dan mesin Aurora DB untuk IAM otentikasi database.

Untuk Aurora MySQL, semua kelas instans DB yang didukung mendukung autentikasi basis data IAM, kecuali untuk db.t2.small dan db.t3.small. Untuk informasi tentang kelas instans DB yang didukung, lihat Mesin DB yang didukung untuk kelas instans DB.

## Dukungan CLI dan SDK

Autentikasi database IAM tersedia untuk AWS CLIdan untuk bahasa khusus berikut: AWS SDKs

- AWS SDK untuk .NET
- AWS SDK untuk C++
- AWS SDK untuk Go
- AWS SDK untuk Java
- AWS SDK untuk JavaScript
- AWS SDK untuk PHP
- AWS SDK untuk Python (Boto3)
- AWS SDK untuk Ruby

## Batasan untuk autentikasi basis data IAM

Saat menggunakan autentikasi basis data IAM, batasan berikut berlaku:

- Autentikasi database IAM membatasi koneksi pada 200 koneksi per detik.

  Koneksi yang menggunakan token otentikasi yang sama tidak dibatasi. Kami menyarankan Anda menggunakan kembali token otentikasi jika memungkinkan.

- Saat ini, autentikasi basis data IAM tidak mendukung kunci konteks kondisi global.

  Untuk informasi selengkapnya tentang kunci konteks kondisi global, lihat Kunci konteks kondisi global AWS dalam Panduan Pengguna IAM.
- Untuk PostgreSQL, jika peran IAM (`rds_iam`) ditambahkan ke pengguna (termasuk pengguna master RDS), autentikasi IAM akan lebih diprioritaskan daripada autentikasi kata sandi, sehingga pengguna harus login sebagai pengguna IAM.
- Untuk  Aurora PostgreSQL, Anda tidak dapat menggunakan autentikasi IAM untuk membuat koneksi replikasi.
- Anda tidak dapat menggunakan data DNS Route 53 kustom sebagai pengganti titik akhir klaster DB untuk menghasilkan token autentikasi.
- CloudWatch dan CloudTrail jangan mencatat otentikasi IAM. Layanan ini tidak melacak panggilan `generate-db-auth-token` API yang mengotorisasi peran IAM untuk mengaktifkan koneksi database.
- Anda harus memiliki antara 300 dan 1000 MiB memori tambahan pada database Anda untuk konektivitas yang andal. Untuk melihat memori yang dibutuhkan untuk beban kerja Anda, bandingkan kolom RES untuk proses RDS dalam daftar proses Enhanced Monitoring sebelum dan sesudah mengaktifkan autentikasi IAM DB. Lihat Melihat metrik OS di konsol RDS.

  Jika Anda menggunakan instance kelas burstable, hindari kehabisan memori dengan mengurangi memori yang digunakan oleh parameter lain seperti buffer dan cache dengan jumlah yang sama.
- Untuk , Anda tidak dapat menggunakan otentikasi berbasis kata sandi untuk pengguna database yang Anda konfigurasikan dengan otentikasi IAM.
- Autentikasi IAM DB tidak didukung untuk RDS di Outposts untuk mesin apa pun.

## Rekomendasi untuk autentikasi basis data IAM

Kami merekomendasikan hal berikut saat menggunakan autentikasi basis data IAM:

- Gunakan autentikasi basis data IAM ketika aplikasi Anda membutuhkan tidak lebih dari 200 koneksi autentikasi basis data IAM per detik.

  Mesin basis data yang berfungsi dengan Amazon Aurora tidak memberlakukan batasan apa pun untuk percobaan autentikasi per detik. Namun, ketika Anda menggunakan autentikasi basis data IAM, aplikasi Anda harus membuat token autentikasi. Aplikasi Anda kemudian menggunakan token tersebut untuk terhubung ke klaster DB. Jika Anda melebihi batas maksimum untuk koneksi baru

per detik, maka overhead tambahan dari autentikasi basis data IAM dapat menyebabkan throttling koneksi.

Pertimbangkan untuk menggunakan pooling koneksi di aplikasi Anda untuk mengurangi pembuatan koneksi yang konstan. Cara ini dapat mengurangi overhead dari autentikasi DB IAM dan memungkinkan aplikasi Anda menggunakan kembali koneksi yang ada. Alternatifnya, pertimbangkan untuk menggunakan Proksi RDS untuk kasus penggunaan ini. Proksi RDS memiliki biaya tambahan. Lihat Harga Proksi RDS.

- Ukuran token autentikasi basis data IAM tergantung pada banyak hal termasuk jumlah tag IAM, kebijakan layanan IAM, panjang ARN, serta properti IAM dan basis data lainnya. Ukuran minimum token ini umumnya sekitar 1 KB tetapi bisa lebih besar. Karena token ini digunakan sebagai kata sandi dalam string koneksi ke database menggunakan otentikasi IAM, Anda harus memastikan bahwa driver database Anda (misalnya, ODBC) alat and/or apa pun tidak membatasi atau memotong token ini karena ukurannya. Token yang terpotong akan menyebabkan kegagalan validasi autentikasi oleh basis data dan IAM.
- Jika Anda menggunakan kredensial sementara saat membuat token autentikasi basis data IAM, kredensial sementara masih harus valid saat menggunakan token autentikasi basis data IAM untuk membuat permintaan koneksi.

## Kunci konteks kondisi AWS global yang tidak didukung

Autentikasi database IAM tidak mendukung subset kunci konteks kondisi AWS global berikut.

- `aws:Referer`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

Untuk informasi selengkapnya, lihat Kunci konteks kondisi global AWS dalam Panduan Pengguna IAM.

## Mengaktifkan dan menonaktifkan autentikasi basis data IAM

Secara default, autentikasi basis data IAM dinonaktifkan di klaster DB. Anda dapat mengaktifkan atau menonaktifkan autentikasi database IAM menggunakan AWS Management Console, AWS CLI, atau API.

Anda dapat mengaktifkan autentikasi basis data IAM saat Anda melakukan salah satu tindakan berikut:

- Untuk membuat klaster DB baru dengan autentikasi IAM diaktifkan, lihat Membuat klaster DB Amazon Aurora.
- Untuk memodifikasi klaster DB untuk mengaktifkan autentikasi basis data IAM, lihat Memodifikasi klaster DB Amazon Aurora.
- Untuk memulihkan klaster DB dari snapshot dengan autentikasi basis data IAM diaktifkan, lihat Memulihkan dari snapshot klaster DB.
- Untuk memulihkan klaster DB ke titik waktu dengan autentikasi basis data IAM diaktifkan, lihat Memulihkan klaster DB ke waktu tertentu.

### Konsol

Setiap alur kerja pembuatan atau modifikasi memiliki bagian Autentikasi basis data, tempat Anda dapat mengaktifkan atau menonaktifkan autentikasi basis data IAM. Di bagian tersebut, pilih Kata sandi dan autentikasi basis data IAM untuk mengaktifkan autentikasi basis data IAM.

Untuk mengaktifkan atau menonaktifkan autentikasi basis data IAM untuk klaster DB yang ada

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB yang ingin Anda ubah.

> **Note**
>
> Anda hanya dapat mengaktifkan autentikasi IAM jika semua instans DB dalam klaster DB kompatibel dengan IAM. Periksa persyaratan kompatibilitas dalam Ketersediaan wilayah dan versi.

4. Pilih Ubah.

5. Di bagian Autentikasi basis data, pilih Kata sandi dan autentikasi basis data IAM untuk mengaktifkan autentikasi basis data IAM. Pilih Autentikasi kata sandi atau Kata sandi dan autentikasi Kerberos untuk menonaktifkan autentikasi IAM.
6. Anda juga dapat memilih untuk mengaktifkan penerbitan log autentikasi IAM DB ke CloudWatch Log. Di bawah Ekspor log, pilih opsi iam-db-auth-error log. Menerbitkan log Anda ke CloudWatch Log menghabiskan penyimpanan dan Anda dikenakan biaya untuk penyimpanan itu. Pastikan untuk menghapus CloudWatch Log apa pun yang tidak lagi Anda butuhkan.
7. Pilih Lanjutkan.
8. Untuk segera menerapkan perubahan, pilih Segera di bagian Penjadwalan modifikasi.
9. Pilih Ubah klaster.

AWS CLI

Untuk membuat cluster DB baru dengan otentikasi IAM dengan menggunakan AWS CLI, gunakan perintah. `create-db-cluster` Tentukan opsi `--enable-iam-database-authentication`.

Untuk memperbarui klaster DB yang ada agar memiliki atau tidak memiliki autentikasi IAM, gunakan perintah AWS CLI `modify-db-cluster`. Tentukan opsi `--enable-iam-database-authentication` atau `--no-enable-iam-database-authentication`, sesuai kebutuhan.

> **Note**
>
> Anda hanya dapat mengaktifkan autentikasi IAM jika semua instans DB dalam klaster DB kompatibel dengan IAM. Periksa persyaratan kompatibilitas dalam Ketersediaan wilayah dan versi.

Secara default, Aurora melakukan modifikasi selama periode pemeliharaan berikutnya. Jika Anda ingin menggantinya dan mengaktifkan autentikasi DB IAM sesegera mungkin, gunakan parameter `--apply-immediately`.

Jika Anda memulihkan cluster DB, gunakan salah satu AWS CLI perintah berikut:

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

Pengaturan autentikasi basis data IAM akan ditetapkan secara default ke pengaturan untuk snapshot sumber. Untuk mengubah pengaturan ini, atur opsi `--enable-iam-database-authentication` atau `--no-enable-iam-database-authentication` sebagaimana diperlukan.

API RDS

Untuk membuat instans DB baru dengan autentikasi IAM dengan menggunakan API, gunakan operasi API `CreateDBCluster`. Atur parameter `EnableIAMDatabaseAuthentication` ke `true`.

Untuk memperbarui klaster DB yang ada agar memiliki autentikasi IAM, gunakan operasi API `ModifyDBCluster`. Atur parameter `EnableIAMDatabaseAuthentication` ke `true` untuk mengaktifkan autentikasi IAM, atau `false` untuk menonaktifkannya.

> **Note**
>
> Anda hanya dapat mengaktifkan autentikasi IAM jika semua instans DB dalam klaster DB kompatibel dengan IAM. Periksa persyaratan kompatibilitas dalam Ketersediaan wilayah dan versi.

Jika Anda memulihkan klaster DB, gunakan salah satu operasi API berikut:

- `RestoreDBClusterFromSnapshot`
- `RestoreDBClusterToPointInTime`

Pengaturan autentikasi basis data IAM akan ditetapkan secara default ke pengaturan untuk snapshot sumber. Untuk mengubah pengaturan ini, atur parameter `EnableIAMDatabaseAuthentication` ke `true` untuk mengaktifkan autentikasi IAM, atau `false` untuk menonaktifkannya.

## Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM

Untuk memungkinkan pengguna atau peran terhubung ke klaster DB, Anda harus membuat kebijakan IAM. Setelah itu, lampirkan kebijakan tersebut ke set izin atau peran.

> **Note**
>
> Untuk mempelajari selengkapnya tentang kebijakan IAM, lihat Manajemen identitas dan akses untuk Aurora.

Contoh kebijakan berikut memungkinkan pengguna terhubung ke klaster DB menggunakan autentikasi basis data IAM.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:1234567890:dbuser:db-ABCDEFGHIJKL01234/
db_user"
         ]
      }
   ]
}
```

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-
ABCDEFGHIJKL01234/db_user"
         ]
      }
   ]
}
```

**Important**

Pengguna dengan izin administrator dapat mengakses klaster DB tanpa izin eksplisit dalam kebijakan IAM. Jika Anda ingin membatasi akses administrator ke klaster DB, Anda dapat membuat peran IAM yang dengan izin istimewa yang sesuai dan menetapkannya ke administrator.

**Note**

Jangan samakan awalan `rds-db:` dengan awalan operasi API RDS lain yang diawali dengan `rds:`. Anda menggunakan awalan `rds-db:` dan tindakan `rds-db:connect` hanya untuk autentikasi basis data IAM. Hal ini tidak berlaku dalam konteks lainnya.

Contoh kebijakan ini mencakup satu pernyataan dengan elemen berikut:

- `Effect` – Tentukan `Allow` untuk memberikan akses ke klaster DB. Jika Anda tidak secara eksplisit mengizinkan akses, maka akses ditolak secara default.
- `Action` – Tentukan `rds-db:connect` untuk mengizinkan koneksi ke klaster DB.
- `Resource` – Tentukan Amazon Resource Name (ARN) yang menjelaskan satu akun basis data dalam satu klaster DB. Format ARN adalah sebagai berikut.

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

Dalam format ini, ganti hal berikut:

- *`region`*adalah AWS Region untuk cluster DB. Dalam contoh kebijakan, AWS Wilayah adalah`us-east-2`.
- *`account-id`*adalah nomor AWS akun untuk cluster DB. Dalam contoh kebijakan, nomor akun adalah `1234567890`. Pengguna harus berada di akun yang sama dengan akun untuk klaster DB.

  Untuk melakukan akses lintas akun, buat peran IAM dengan kebijakan yang ditunjukkan di atas di akun untuk klaster DB dan izinkan akun Anda yang lain untuk mengambil peran tersebut.

- *DbClusterResourceId* adalah pengidentifikasi untuk klaster DB. Pengenal ini unik untuk AWS Wilayah dan tidak pernah berubah. Dalam contoh kebijakan, pengidentifikasi adalah `cluster-ABCDEFGHIJKL01234`.

  Untuk menemukan ID sumber daya cluster DB di Amazon Aurora Amazon, pilih cluster instans untuk melihat detailnya. AWS Management Console Kemudian, pilih tab Konfigurasi. ID Sumber Daya ditampilkan di bagian Konfigurasi.

  Atau, Anda dapat menggunakan AWS CLI perintah untuk mencantumkan pengenal dan sumber daya IDs untuk semua cluster DB Anda di AWS Wilayah saat ini, seperti yang ditunjukkan berikut.

  ```
  aws rds describe-db-clusters --query "DBClusters[*].
  [DBClusterIdentifier,DbClusterResourceId]"
  ```

  > **Note**
  >
  > Jika Anda terhubung ke basis data melalui Proksi RDS, tentukan ID sumber daya proksi, seperti `prx-ABCDEFGHIJKL01234`. Untuk informasi tentang menggunakan autentikasi basis data IAM dengan Proksi RDS, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

- *db-user-name* adalah nama akun basis data untuk dikaitkan dengan autentikasi IAM. Dalam contoh kebijakan, akun basis data adalah `db_user`.

Anda dapat membangun yang lain ARNs untuk mendukung berbagai pola akses. Kebijakan berikut memungkinkan akses ke dua akun basis data yang berbeda dalam klaster DB.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
```

```
            "rds-db:connect"
        ],
        "Resource": [
            "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
jane_doe",
            "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
mary_roe"
        ]
     }
  ]
}
```

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/mary_roe"
         ]
      }
   ]
}
```

Kebijakan berikut menggunakan karakter “*” untuk mencocokkan semua cluster DB dan akun database untuk AWS akun dan AWS Wilayah tertentu.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:*/*"
            ]
        }
    ]
}
```

Kebijakan berikut cocok dengan semua cluster DB untuk AWS akun dan AWS Wilayah tertentu. Namun, kebijakan ini hanya memberikan akses ke klaster DB yang memiliki akun basis data `jane_doe`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

Pengguna atau peran hanya memiliki akses ke basis data yang aksesnya dimiliki oleh pengguna basis data tersebut. Misalnya, anggaplah klaster DB Anda memiliki basis data bernama dev, dan basis data lain bernama uji. Jika pengguna basis data `jane_doe` hanya memiliki akses ke dev, setiap pengguna atau peran yang mengakses klaster DB tersebut dengan pengguna `jane_doe` juga hanya akan memiliki akses ke dev. Pembatasan akses ini juga berlaku untuk objek basis data lain, seperti tabel, tampilan, dan sebagainya.

Administrator harus membuat kebijakan IAM yang memberikan izin pada entitas untuk melakukan operasi API tertentu pada sumber daya yang diperlukan. Administrator kemudian harus melampirkan kebijakan tersebut ke set izin atau peran yang memerlukan izin tersebut. Untuk contoh kebijakan, lihat Contoh kebijakan berbasis identitas untuk  Amazon Aurora.

Melampirkan kebijakan IAM ke set izin atau peran

Setelah membuat kebijakan IAM untuk memungkinkan autentikasi basis data, Anda perlu melampirkan kebijakan tersebut ke set izin atau peran. Untuk tutorial tentang topik ini, lihat Buat dan lampirkan kebijakan yang dikelola pelanggan pertama Anda dalam Panduan Pengguna IAM.

Saat mengikuti tutorial ini, Anda dapat menggunakan salah satu contoh kebijakan yang ditunjukkan dalam bagian ini sebagai titik awal dan menyesuaikannya dengan kebutuhan Anda. Di akhir tutorial, Anda akan memiliki set izin dengan kebijakan terlampir yang dapat menggunakan tindakan `rds-db:connect`.

> **Note**
>
> Anda dapat memetakan beberapa set izin atau peran ke akun pengguna basis data yang sama. Misalnya, anggaplah kebijakan IAM Anda telah menentukan ARN sumber daya berikut.
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/
> jane_doe
> ```
>
> Jika Anda melampirkan kebijakan ke Jane, Bob, dan Diego, maka masing-masing pengguna tersebut dapat terhubung ke instans DB yang telah ditentukan menggunakan akun basis data `jane_doe`.

## Membuat akun basis data menggunakan autentikasi IAM

Dengan autentikasi basis data IAM, Anda tidak perlu menetapkan kata sandi basis data ke akun pengguna yang Anda buat. Jika Anda menghapus pengguna yang dipetakan ke akun basis data, Anda juga harus menghapus akun basis data dengan pernyataan `DROP USER`.

> **Note**
>
> Nama pengguna yang digunakan untuk autentikasi IAM harus sesuai dengan huruf besar/kecil nama pengguna dalam basis data.

Topik

- Menggunakan autentikasi IAM dengan Aurora MySQL
- Menggunakan autentikasi IAM dengan Aurora PostgreSQL

### Menggunakan autentikasi IAM dengan Aurora MySQL

Dengan dan Aurora MySQL, otentikasi `AWSAuthenticationPlugin` ditangani AWS oleh —sebuah plugin yang disediakan yang bekerja mulus dengan IAM untuk mengautentikasi pengguna Anda. Hubungkan ke klaster DB sebagai pengguna master atau pengguna lain yang dapat membuat pengguna dan memberikan hak akses. Setelah terhubung, berikan pernyataan `CREATE USER`, seperti yang ditunjukkan pada contoh berikut.

```
CREATE USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

Klausa `IDENTIFIED WITH` memungkinkan Aurora MySQL menggunakan `AWSAuthenticationPlugin` untuk mengautentikasi akun basis data (`jane_doe`). Klausa `AS 'RDS'` mengacu pada metode autentikasi. Pastikan nama pengguna basis data yang ditentukan sama dengan sumber daya dalam kebijakan IAM untuk akses basis data IAM. Untuk informasi selengkapnya, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM.

> **Note**
>
> Jika Anda melihat pesan berikut, itu berarti plugin AWS-provided tidak tersedia untuk cluster DB saat ini.

```
ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded
```

Untuk mengatasi kesalahan ini, verifikasikan bahwa Anda menggunakan konfigurasi yang didukung dan bahwa Anda telah mengaktifkan autentikasi basis data IAM di klaster DB Anda. Untuk informasi lebih lanjut, lihat Ketersediaan wilayah dan versi dan Mengaktifkan dan menonaktifkan autentikasi basis data IAM.

Setelah membuat akun menggunakan `AWSAuthenticationPlugin`, Anda mengelolanya dengan cara yang sama seperti akun basis data lainnya. Misalnya, Anda dapat memodifikasi hak akses akun dengan pernyataan `GRANT` dan `REVOKE`, atau memodifikasi berbagai atribut akun dengan pernyataan `ALTER USER`.

Lalu lintas jaringan basis data dienkripsi menggunakan SSL/TLS saat menggunakan IAM. Untuk mengizinkan koneksi SSL, ubah akun pengguna dengan perintah berikut.

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

Menggunakan autentikasi IAM dengan Aurora PostgreSQL

Untuk menggunakan autentikasi IAM dengan Aurora PostgreSQL, hubungkan ke klaster DB sebagai pengguna master atau pengguna lain yang dapat membuat pengguna dan memberikan hak istimewa. Setelah terhubung, buat pengguna basis data lalu berikan peran `rds_iam` kepada pengguna tersebut seperti yang ditunjukkan pada contoh berikut.

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

Pastikan nama pengguna basis data yang ditentukan sama dengan sumber daya dalam kebijakan IAM untuk akses basis data IAM. Untuk informasi selengkapnya, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Anda harus memberikan `rds_iam` peran untuk menggunakan otentikasi IAM. Anda juga dapat menggunakan keanggotaan bersarang atau hibah tidak langsung dari peran tersebut.

Perhatikan bahwa pengguna basis data PostgreSQL dapat menggunakan autentikasi IAM atau Kerberos tetapi tidak keduanya, sehingga pengguna ini tidak dapat memiliki juga peran `rds_ad`. Hal ini juga berlaku untuk keanggotaan bersarang. Untuk informasi selengkapnya, lihat Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda .

## Menghubungkan ke klaster DB menggunakan autentikasi IAM

Dengan autentikasi basis data IAM, Anda menggunakan token autentikasi ketika Anda terhubung ke klaster DB Anda. Token autentikasi adalah string karakter yang Anda gunakan sebagai pengganti kata sandi. Setelah Anda membuat token autentikasi, token tersebut berlaku selama 15 menit sebelum kedaluwarsa. Jika Anda mencoba terhubung menggunakan token yang kedaluwarsa, permintaan koneksi ditolak.

Setiap token autentikasi harus disertai dengan tanda tangan yang valid, menggunakan AWS Signature versi 4. (Untuk informasi selengkapnya, lihat Proses penandatanganan Versi Tanda Tangan 4 di  Referensi Umum AWS. ) SDK AWS CLI dan AWS SDK, seperti AWS SDK untuk Java or AWS SDK untuk Python (Boto3), dapat secara otomatis menandatangani setiap token yang Anda buat.

Anda dapat menggunakan token otentikasi saat Anda terhubung ke Aurora dari layanan AWS lain, seperti. AWS Lambda Dengan menggunakan token, Anda tidak perlu menempatkan kata sandi dalam kode Anda. Atau, Anda dapat menggunakan AWS SDK untuk membuat dan menandatangani token otentikasi secara terprogram secara terprogram.

Setelah Anda memiliki token autentikasi IAM yang telah ditandatangani, Anda dapat terhubung ke klaster DB Aurora. Berikut ini, Anda dapat mengetahui cara melakukan ini menggunakan alat baris perintah atau AWS SDK, seperti AWS SDK untuk Java atau AWS SDK untuk Python (Boto3).

Untuk informasi selengkapnya, lihat postingan blog berikut ini:

- Gunakan autentikasi IAM untuk terhubung dengan SQL Workbench/J ke Aurora MySQL atau Amazon RDS for MySQL
- Menggunakan autentikasi IAM untuk terhubung dengan pgAdmin Amazon Aurora PostgreSQL atau Amazon RDS for PostgreSQL

### Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke klaster DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

### Topik

- Menghubungkan ke cluster DB Anda menggunakan autentikasi IAM dengan driver AWS
- Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dari baris perintah: AWS CLI dan klien mysql
- Menghubungkan ke instans DB Anda menggunakan autentikasi IAM dari baris perintah: AWS CLI dan klien psql
- Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK untuk .NET
- Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dan AWS SDK untuk Go
- Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dan AWS SDK untuk Java
- Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dan AWS SDK untuk Python (Boto3)

Menghubungkan ke cluster DB Anda menggunakan autentikasi IAM dengan driver AWS

AWS Rangkaian driver telah dirancang untuk memberikan dukungan untuk waktu peralihan dan failover yang lebih cepat, dan otentikasi dengan, AWS Identity and Access Management (IAM) AWS Secrets Manager, dan Federated Identity. AWS Driver mengandalkan pemantauan status dan menyadari topologi cluster untuk menentukan penulis baru. Pendekatan ini mengurangi waktu peralihan dan failover menjadi satu digit detik, dibandingkan dengan puluhan detik untuk driver open-source.

Untuk informasi lebih lanjut tentang AWS driver, lihat driver bahasa yang sesuai untuk klaster Aurora MySQL atau Aurora PostgreSQL DB Anda.

Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dari baris perintah: AWS CLI dan klien mysql

Anda dapat terhubung dari baris perintah ke cluster Aurora DB dengan AWS CLI alat baris perintah `mysql` dan seperti yang dijelaskan berikut.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke klaster DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

> **Note**
>
> Untuk informasi tentang cara menghubungkan ke basis data menggunakan SQL Workbench/J dengan autentikasi IAM, lihat postingan blog Menggunakan autentikasi IAM untuk terhubung dengan SQL Workbench/J ke Aurora MySQL atau Amazon RDS for MySQL.

Topik

- Membuat token autentikasi IAM
- Menghubungkan ke klaster DB

### Membuat token autentikasi IAM

Contoh berikut menunjukkan cara mendapatkan token autentikasi yang ditandatangani menggunakan AWS CLI.

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

Dalam contoh, parameternya adalah sebagai berikut:

- `--hostname` – Nama host klaster DB yang ingin Anda akses
- `--port` – Nomor port yang digunakan untuk menghubungkan ke klaster DB Anda
- `--region`— AWS Wilayah tempat cluster DB berjalan
- `--username` – Akun basis data yang ingin Anda akses

Beberapa karakter pertama dari token terlihat seperti berikut.

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> Anda tidak dapat menggunakan catatan DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir klaster DB untuk menghasilkan token autentikasi.

Menghubungkan ke klaster DB

Format umum untuk terhubung ditampilkan sebagai berikut.

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-cleartext-plugin --user=userName --password=authToken
```

Parameternya adalah sebagai berikut:

- `--host` – Nama host klaster DB yang ingin Anda akses
- `--port` – Nomor port yang digunakan untuk menghubungkan ke klaster DB Anda
- `--ssl-ca` – Jalur lengkap ke file sertifikat SSL yang berisi kunci publik

  Untuk informasi selengkapnya, lihat Koneksi TLS ke cluster DB MySQL Aurora.

  Untuk mengunduh sertifikat SSL, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .
- `--enable-cleartext-plugin` – Nilai yang menentukan bahwa `AWSAuthenticationPlugin` harus digunakan untuk koneksi ini

  Jika Anda menggunakan klien MariaDB, opsi `--enable-cleartext-plugin` tidak diperlukan.
- `--user` – Akun basis data yang ingin Anda akses
- `--password` – Token autentikasi IAM yang ditandatangani

Token autentikasi terdiri atas beberapa ratus karakter. Hal ini dapat sulit ditangani di baris perintah. Salah satu cara untuk mengatasinya adalah dengan menyimpan token ke variabel lingkungan, lalu menggunakan variabel tersebut saat Anda terhubung. Contoh berikut menunjukkan satu cara untuk melakukan solusi ini. Dalam contoh ini, */sample_dir/* adalah jalur lengkap ke file sertifikat SSL yang berisi kunci publik.

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
```

```
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

Saat Anda terhubung menggunakan `AWSAuthenticationPlugin`, koneksi diamankan menggunakan SSL. Untuk memverifikasi hal ini, ketik berikut ini di prompt perintah `mysql>`.

```
show status like 'Ssl%';
```

Baris berikut dalam output menampilkan lebih banyak detail.

```
+---------------+-------------+
| Variable_name | Value

                                                                    |

+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                    |

| ...           | ...
| Ssl_version   | TLSv1.1

                                                                    |

| ...           | ...
+-----------------------------+
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke instans DB Anda menggunakan autentikasi IAM dari baris perintah: AWS CLI dan klien psql

Anda dapat terhubung dari baris perintah ke klaster DB Aurora PostgreSQL dengan AWS CLI dan alat baris perintah psql seperti yang dijelaskan berikut.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

> **Note**
>
> Untuk informasi tentang cara menghubungkan ke basis data menggunakan pgAdmin dengan autentikasi IAM, lihat postingan blog Menggunakan autentikasi IAM untuk terhubung dengan pgAdmin Amazon Aurora PostgreSQL atau Amazon RDS for PostgreSQL.

Topik

- Membuat token autentikasi IAM
- Menghubungkan ke klaster Aurora PostgreSQL

### Membuat token autentikasi IAM

Token autentikasi terdiri dari ratusan karakter sehingga kemungkinan menjadi sulit ditangani di baris perintah. Salah satu cara untuk mengatasi ini adalah dengan menyimpan token ke variabel lingkungan, lalu menggunakan variabel tersebut saat Anda terhubung. Contoh berikut menunjukkan cara menggunakan AWS CLI untuk mendapatkan token autentikasi yang ditandatangani menggunakan perintah `generate-db-auth-token` dan menyimpannya di variabel lingkungan `PGPASSWORD`.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

Dalam contoh, parameter untuk perintah `generate-db-auth-token` adalah sebagai berikut:

- `--hostname` – Nama host instans DB yang ingin Anda akses
- `--port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `--region` – Wilayah AWS tempat instans DB beroperasi
- `--username` – Akun basis data yang ingin Anda akses

Beberapa karakter pertama dari token yang dihasilkan terlihat sebagai berikut.

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Menghubungkan ke klaster Aurora PostgreSQL

Format umum untuk menggunakan psql untuk terhubung ditampilkan sebagai berikut.

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

Parameternya adalah sebagai berikut:

- `host` – Nama host instans DB yang ingin Anda akses
- `port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `sslmode` – Mode SSL yang akan digunakan

  Saat Anda menggunakan `sslmode=verify-full`, koneksi SSL memverifikasi titik akhir instans DB di sertifikat SSL.
- `sslrootcert` – Jalur lengkap ke file sertifikat SSL yang berisi kunci publik

  Untuk informasi selengkapnya, lihat Mengamankan data Aurora PostgreSQL dengan SSL/TLS.

  Untuk mengunduh sertifikat SSL, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .
- `dbname` – Basis data yang ingin Anda akses
- `user` – Akun basis data yang ingin Anda akses
- `password` – Token autentikasi IAM yang ditandatangani

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Contoh berikut menunjukkan cara menggunakan psql untuk terhubung. Dalam contoh, psql menggunakan variabel lingkungan `RDSHOST` untuk host dan variabel lingkungan `PGPASSWORD` untuk token yang dihasilkan. Selain itu, *`/sample_dir/`* adalah jalur penuh ke file sertifikat SSL yang berisi kunci publik.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK untuk .NET

Anda dapat menghubungkan ke klaster DB Aurora MySQL atau Aurora PostgreSQL dengan AWS SDK untuk .NET seperti yang dijelaskan berikut ini.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Contoh

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke instans DB.

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK untuk .NET, yang ada di situs AWS. Paket `AWSSDK.CORE` dan `AWSSDK.RDS` diperlukan. Untuk terhubung ke instans DB, gunakan konektor basis data .NET untuk mesin DB, seperti MySqlConnector for MariaDB atau MySQL, atau Npgsql for PostgreSQL.

Kode ini terhubung ke klaster DB Aurora MySQL. Ubah nilai variabel berikut sesuai kebutuhan:

- `server` – Titik akhir instans DB yang ingin Anda akses
- `user` – Akun basis data yang ingin Anda akses
- `database` – Basis data yang ingin Anda akses
- `port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `SslMode` – Mode SSL yang akan digunakan

  Saat Anda menggunakan `SslMode=Required`, koneksi SSL memverifikasi titik akhir instans DB di sertifikat SSL.
- `SslCa` – Jalur lengkap ke sertifikat SSL untuk Amazon Aurora

  Untuk mengunduh sertifikat SSL, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
```

```
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

Ubah nilai variabel berikut sesuai kebutuhan:

- `Server` – Titik akhir instans DB yang ingin Anda akses
- `User ID` – Akun basis data yang ingin Anda akses
- `Database` – Basis data yang ingin Anda akses
- `Port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `SSL Mode` – Mode SSL yang akan digunakan

  Saat Anda menggunakan `SSL Mode=Required`, koneksi SSL memverifikasi titik akhir instans DB di sertifikat SSL.
- `Root Certificate` – Jalur lengkap ke sertifikat SSL untuk Amazon Aurora

Untuk mengunduh sertifikat SSL, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
```

```
            conn.Close();
        }
    }
}
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dan AWS SDK untuk Go

Anda dapat terhubung ke dengan seperti yang dijelaskan berikut. AWS SDK untuk Go

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke klaster DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Contoh

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK untuk Go, ditemukan di AWS situs.

Ubah nilai variabel berikut sesuai kebutuhan:

- `dbName` – Basis data yang ingin Anda akses
- `dbUser` – Akun basis data yang ingin Anda akses
- `dbHost` – Titik akhir klaster DB yang ingin Anda akses

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir klaster DB untuk menghasilkan token autentikasi.

- `dbPort` – Nomor port yang digunakan untuk menghubungkan ke klaster DB Anda
- `region`— AWS Wilayah tempat cluster DB berjalan

Selain itu, pastikan pustaka yang diimpor dalam kode sampel ada di sistem Anda.

**Important**

Contoh dalam bagian ini menggunakan kode berikut untuk menyediakan kredensial yang mengakses basis data dari lingkungan lokal:

```
creds := credentials.NewEnvCredentials()
```

Jika Anda mengakses database dari AWS layanan, seperti Amazon EC2 atau Amazon ECS, Anda dapat mengganti kode dengan kode berikut:

```
sess := session.Must(session.NewSession())
creds := sess.Config.Credentials
```

Jika Anda membuat perubahan ini, pastikan Anda menambahkan impor berikut:

```
"github.com/aws/aws-sdk-go/aws/session"
```

Topik


- Menghubungkan menggunakan otentikasi IAM dan V2 AWS SDK untuk Go
- Menghubungkan menggunakan otentikasi IAM dan V1. AWS SDK untuk Go


## Menghubungkan menggunakan otentikasi IAM dan V2 AWS SDK untuk Go

Anda dapat terhubung ke cluster DB menggunakan autentikasi IAM dan V2. AWS SDK untuk Go

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke klaster DB.

Kode ini terhubung ke klaster DB Aurora MySQL.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)

func main() {
```

```
     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 3306
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

```
package main

import (
     "context"
     "database/sql"
     "fmt"
```

```
     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan menggunakan otentikasi IAM dan V1. AWS SDK untuk Go

Anda dapat terhubung ke cluster DB menggunakan otentikasi IAM dan V1 AWS SDK untuk Go

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke klaster DB.

Kode ini terhubung ke klaster DB Aurora MySQL.

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
```

```
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
```

```
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dan AWS SDK untuk Java

Anda dapat terhubung ke dengan seperti yang dijelaskan berikut. AWS SDK untuk Java

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke klaster DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM
- Menyiapkan AWS SDK for Java

Untuk contoh tentang cara menggunakan SDK for Java 2.x, lihat contoh Amazon RDS menggunakan SDK for Java 2.x. Anda juga dapat menggunakan AWS Advanced JDBC Wrapper, lihat Dokumentasi AWS Advanced JDBC Wrapper.

Topik

- Membuat token autentikasi IAM
- Membuat token autentikasi IAM secara manual
- Menghubungkan ke klaster DB

Membuat token autentikasi IAM

Jika Anda menulis program menggunakan AWS SDK untuk Java, Anda bisa mendapatkan token otentikasi yang ditandatangani menggunakan `RdsIamAuthTokenGenerator` kelas. Menggunakan kelas ini mengharuskan Anda memberikan AWS kredensil. Untuk melakukan ini, Anda membuat instance dari `DefaultAWSCredentialsProviderChain` kelas.

`DefaultAWSCredentialsProviderChain`menggunakan kunci AWS akses pertama dan kunci rahasia yang ditemukan di rantai penyedia kredensi default. Untuk informasi selengkapnya tentang kunci akses AWS , lihat Mengelola kunci akses untuk pengguna.

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir klaster DB untuk menghasilkan token autentikasi.

Setelah membuat instans `RdsIamAuthTokenGenerator`, Anda dapat memanggil metode `getAuthToken` untuk mendapatkan token yang ditandatangani. Berikan Wilayah AWS , nama host, nomor port, dan nama pengguna. Contoh kode berikut menunjukkan cara melakukannya.

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
```

```
      .hostname(hostName)
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

Membuat token autentikasi IAM secara manual

Di Java, cara termudah untuk menghasilkan token autentikasi adalah dengan menggunakan `RdsIamAuthTokenGenerator`. Kelas ini membuat token otentikasi untuk Anda, dan kemudian menandatanganinya menggunakan AWS tanda tangan versi 4. Untuk informasi selengkapnya, lihat Proses penandatanganan Signature versi 4 dalam Referensi Umum AWS.

Namun, Anda juga dapat membuat dan menandatangani token autentikasi secara manual, seperti ditunjukkan dalam contoh kode berikut.

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
```

```
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
    String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1:  Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2:  Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3:  Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();
```

```
        System.out.println("Step 4:  Add the signing info to the request");

        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String
date, String dateTime, String region, String expiryPeriod, String hostName, String
port) throws Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
+ '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
String accessKey, String date, String region) throws Exception {
```

```
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                          SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
 serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }
```

```
    public static byte[] sign(String stringData, byte[] key,
                      SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

Menghubungkan ke klaster DB

Contoh kode berikut menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke klaster yang menjalankan Aurora MySQL.

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK untuk Java, ditemukan di AWS situs. Selain itu, Anda memerlukan hal berikut:

- MySQL Connector/J. Contoh kode ini diuji dengan `mysql-connector-java-5.1.33-bin.jar`.
- Sertifikat perantara untuk Amazon Aurora yang khusus untuk AWS suatu Wilayah. (Untuk informasi selengkapnya, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .) Saat runtime,

pemuat kelas mencari sertifikat di direktori yang sama seperti contoh kode Java ini, sehingga pemuat kelas dapat menemukannya.

- Ubah nilai variabel berikut sesuai kebutuhan:
  - `RDS_INSTANCE_HOSTNAME` – Nama host klaster DB yang ingin Anda akses.
  - `RDS_INSTANCE_PORT` – Nomor port yang digunakan untuk menghubungkan ke klaster DB PostgreSQL Anda.
  - `REGION_NAME`— AWS Wilayah tempat cluster DB berjalan.
  - `DB_USER` – Akun basis data yang ingin Anda akses.
  - `SSL_CERTIFICATE`— Sertifikat SSL untuk Amazon Aurora yang khusus untuk suatu Wilayah. AWS

    Untuk mengunduh sertifikat untuk Wilayah AWS Anda, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke . Tempatkan sertifikat SSL di direktori yang sama dengan file program Java ini, sehingga pemuat kelas dapat menemukan sertifikat saat runtime.

Contoh kode ini memperoleh AWS kredensil dari rantai penyedia kredensi default.

> **Note**
>
> Tentukan kata sandi untuk `DEFAULT_KEY_STORE_PASSWORD` selain prompt yang ditampilkan di sini sebagai praktik terbaik keamanan.

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
```

```
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
```

```
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
 AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
 AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

 .hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type
 and password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword",
 DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL
 verification during the IAM Database Authentication to
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
```

```
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
 Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
 KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke cluster DB Anda menggunakan otentikasi IAM dan AWS SDK untuk Python (Boto3)

Anda dapat terhubung ke dengan seperti yang dijelaskan berikut. AWS SDK untuk Python (Boto3)

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke klaster DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Selain itu, pastikan pustaka yang diimpor dalam kode sampel ada di sistem Anda.

Contoh

Contoh kode ini menggunakan profil untuk kredensial bersama. Untuk informasi tentang kredensi yang menentukan, lihat Kredensial dalam dokumentasi. AWS SDK untuk Python (Boto3)

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke klaster DB.

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK untuk Python (Boto3), ditemukan di AWS situs.

Ubah nilai variabel berikut sesuai kebutuhan:

- `ENDPOINT` – Titik akhir klaster DB yang ingin Anda akses
- `PORT` – Nomor port yang digunakan untuk menghubungkan ke klaster DB Anda
- `USER` – Akun basis data yang ingin Anda akses
- `REGION`— AWS Wilayah tempat cluster DB berjalan
- `DBNAME` – Basis data yang ingin Anda akses
- `SSLCERTIFICATE` – Jalur lengkap ke sertifikat SSL untuk Amazon Aurora

  Untuk `ssl_ca`, tentukan sertifikat SSL. Untuk mengunduh sertifikat SSL, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke .

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir klaster DB untuk menghasilkan token autentikasi.

Kode ini terhubung ke klaster DB Aurora MySQL.

Sebelum menjalankan kode ini, instal driver PyMy SQL dengan mengikuti instruksi dalam Indeks Paket Python.

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =
 pymysql.connect(auth_plugin_map={'mysql_clear_password':None},host=ENDPOINT,
 user=USER, password=token, port=PORT, database=DBNAME, ssl_ca='SSLCERTIFICATE',
 ssl_verify_identity=True, ssl_verify_cert=True)
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

Sebelum menjalankan kode ini, instal `psycopg2` dengan mengikuti petunjuk dalam dokumentasi Psycopg.

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

## Pemecahan masalah untuk otentikasi IAM DB

Berikut ini, Anda dapat menemukan ide pemecahan masalah untuk beberapa masalah otentikasi IAM DB umum dan informasi tentang CloudWatch log dan metrik untuk otentikasi IAM DB.

Mengekspor log kesalahan otentikasi IAM DB ke Log CloudWatch

Log kesalahan otentikasi IAM DB disimpan di host database, dan Anda dapat mengekspor log ini ke akun CloudWatch Log Anda. Gunakan log dan metode remediasi di halaman ini untuk memecahkan masalah autentikasi IAM DB.

Anda dapat mengaktifkan ekspor log ke CloudWatch Log dari konsol AWS CLI, dan RDS API. Untuk petunjuk konsol, lihatMenerbitkan log database ke Amazon CloudWatch Logs.

Untuk mengekspor log kesalahan autentikasi IAM DB Anda ke CloudWatch Log saat membuat cluster DB dari AWS CLI, gunakan perintah berikut:

```
aws rds create-db-cluster --db-cluster-identifier mydbinstance \
--region us-east-1 \
--engine postgres \
--engine-version 16 \
--master-username master \
--master-user-password password \
--publicly-accessible \
--enable-iam-database-authentication \
--enable-cloudwatch-logs-exports=iam-db-auth-error
```

Untuk mengekspor log kesalahan autentikasi IAM DB Anda ke CloudWatch Log saat memodifikasi cluster DB dari AWS CLI, gunakan perintah berikut:

```
aws rds modify-db-cluster --db-instance-identifier mydbcluster \
--region us-east-1 \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["iam-db-auth-error"]}'
```

Untuk memverifikasi apakah cluster DB Anda mengekspor log otentikasi IAM DB ke CloudWatch Log, periksa apakah `EnabledCloudwatchLogsExports` parameter diatur ke `iam-db-auth-error` dalam output untuk perintah tersebut. `describe-db-instances`

```
aws rds describe-db-cluster --region us-east-1 --db-cluster-identifier mydbcluster
            ...

             "EnabledCloudwatchLogsExports": [
                "iam-db-auth-error"
            ],
            ...
```

Metrik otentikasi CloudWatch IAM DB

Amazon Aurora memberikan metrik waktu nyaris nyata tentang autentikasi IAM DB ke akun Amazon Anda. CloudWatch Tabel berikut mencantumkan metrik autentikasi IAM DB yang tersedia menggunakan: CloudWatch

| Metrik | Deskripsi |
| --- | --- |
| `IamDbAuthConnectionRequests` | Jumlah total permintaan koneksi yang dibuat dengan otentikasi IAM DB. |
| `IamDbAuthConnectionSuccess` | Jumlah total permintaan otentikasi IAM DB yang berhasil. |
| `IamDbAuthConnectionFailure` | Jumlah total permintaan otentikasi IAM DB yang gagal. |
| `IamDbAuthConnectionFailureInvalidToken` | Jumlah total permintaan otentikasi IAM DB yang gagal karena token tidak valid. |
| `IamDbAuthConnectionFailureInsufficientPermissions` | Jumlah total permintaan otentikasi IAM DB yang gagal karena kebijakan atau izin yang salah. |
| `IamDbAuthConnectionFailureThrottling` | Jumlah total permintaan otentikasi IAM DB yang gagal karena pelambatan otentikasi IAM DB. |
| `IamDbAuthConnectionFailureServerError` | Jumlah total permintaan otentikasi IAM DB yang gagal karena kesalahan server internal dalam fitur otentikasi IAM DB. |

Masalah dan solusi umum

Anda mungkin mengalami masalah berikut saat menggunakan keaslian IAM DB. Gunakan langkah-langkah remediasi dalam tabel untuk menyelesaikan masalah:

| Kesalahan | Metrik | Penyebab | Solusi |
| --- | --- | --- | --- |
| `[ERROR] Failed to authenticate the connection request for user db_user because the provided token is malformed or otherwise invalid. (Status Code: 400, Error Code: InvalidToken)` | `IamDbAuthConnectionFailure`<br><br>`IamDbAuthConnectionFailureInvalidToken` | Token autentiasi IAM DB dalam permintaan koneksi bukan token Sigv4a yang valid, atau tidak diformat dengan benar. | Periksa strategi pembuatan token Anda di aplikasi Anda. Dalam beberapa kasus, pastikan Anda meneruskan token dengan format yang valid. Memangkas token (atau pemformatan string yang salah) akan membuat token tidak valid. |
| `[ERROR] Failed to authenticate the connection request for user db_user because the token age is longer than 15 minutes. (Status Code: 400, Error Code:ExpiredToken)` | `IamDbAuthConnectionFailure`<br><br>`IamDbAuthConnectionFailureInvalidToken` | Token otentikasi IAM DB telah kedaluwarsa. Token hanya berlaku selama 15 menit. | Periksa caching token dan/atau logika penggunaan kembali token Anda di aplikasi Anda. Anda tidak boleh menggunakan kembali token yang lebih tua dari 15 menit. |
| `[ERROR] Failed to authorize the connection request for user db_user because the IAM policy assumed by the caller 'arn:aws:` | `IamDbAuthConnectionFailure`<br><br>`IamDbAuthConnectionFailureInsufficientPermissions` | Kesalahan ini mungkin terjadi karena alasan berikut:<br><br>• Kebijakan IAM yang diasumsikan oleh aplikasi tidak mengizinkan tindakan tersebut. | Verifikasi bahwa peran dan/atau kebijakan IAM yang Anda asumsikan dalam aplikasi Anda. Pastikan Anda mengasumsikan kebijakan yang sama untuk menghasilkan |

| Kesalahan | Metrik | Penyebab | Solusi |
| --- | --- | --- | --- |
| `sts::1234 56789012: assumed-r ole/ <RoleName >/ <RoleSess ion>' is not authorized to perform `rds-db:connect` on the DB instance. (Status Code: 403, Error Code:NotA uthorized)` | | `rds-db:co nnect`<br>• Anda mengasums ikan peran/keb ijakan yang salah untuk terhubung *db_user* ke database.<br>• Anda mengasums ikan kebijakan yang benar untuk*db_user*, tetapi Anda tidak terhubung ke database yang benar. | token untuk terhubung ke DB. |
| `[ERROR] Failed to authorize the connection request for user` *db_user* `due to IAM DB authentic ation throttlin g. (Status Code: 429, Error Code: Throttlin gException)` | `IamDbAuth Connectio nFailure`<br><br>`IamDbAuth Connectio nFailureT hrottling` | Anda membuat terlalu banyak permintaan koneksi ke DB Anda dalam waktu singkat. Batas pelambatan otentikasi IAM DB adalah 200 koneksi per detik. | Kurangi tingkat membangun koneksi baru dengan otentikas i IAM. Pertimbangkan untuk menerapkan penyatuan koneksi menggunakan RDS Proxy untuk menggunakan kembali koneksi yang sudah ada di aplikasi Anda. |

| Kesalahan | Metrik | Penyebab | Solusi |
| --- | --- | --- | --- |
| `[ERROR] Failed to authorize the connection request for user db_user due to an internal IAM DB authentication error. (Status Code: 500, Error Code: InternalError)` | `IamDbAuthConnectionFailure`<br><br>`IamDbAuthConnectionFailureThrottling` | Ada kesalahan internal saat mengotorisasi koneksi DB dengan otentikasi IAM DB. | Jangkau https://aws.amazon.com/premiumsupport/ untuk menyelidiki masalah ini. |

## Memecahkan masalah identitas dan akses Amazon Aurora

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda hadapi saat bekerja dengan Aurora dan IAM.

Topik

- Saya tidak diberi otorisasi untuk melakukan tindakan di Aurora
- Saya tidak memiliki izin untuk melakukan iam:PassRole
- Saya ingin mengizinkan orang di luar akun AWS saya untuk mengakses sumber daya Aurora

### Saya tidak diberi otorisasi untuk melakukan tindakan di Aurora

Jika AWS Management Console memberi tahu bahwa Anda tidak diberi otorisasi untuk melakukan tindakan, Anda harus menghubungi administrator untuk mendapatkan bantuan. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Contoh kesalahan berikut terjadi saat pengguna `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang *`widget`*, tetapi tidak memiliki izin `rds:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk mengizinkan dia mengakses sumber daya *my-example-widget* menggunakan tindakan `rds:GetWidget`.

## Saya tidak memiliki izin untuk melakukan iam:PassRole

Jika Anda menerima kesalahan bahwa Anda tidak diberi otorisasi untuk melakukan tindakan `iam:PassRole`, Anda harus menghubungi administrator untuk mendapatkan bantuan. Administrator Anda adalah orang yang memberi Anda kredensial masuk. Minta orang tersebut untuk memperbarui kebijakan Anda agar Anda dapat meneruskan peran ke Aurora.

Beberapa layanan AWS mengizinkan Anda untuk melewati peran yang sudah ada ke layanan tersebut, alih-alih membuat peran layanan atau peran yang ditautkan ke layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi saat pengguna bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Aurora. Namun, tindakan ini mengharuskan layanan memiliki izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut ke layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, Mary meminta administrator untuk memperbarui kebijakannya agar dia dapat melakukan tindakan `iam:PassRole`.

## Saya ingin mengizinkan orang di luar akun AWS saya untuk mengakses sumber daya Aurora

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau pengguna di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi pengguna akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa hal berikut:

- Untuk mempelajari apakah Aurora mendukung fitur ini, lihat Bagaimana Aurora bekerja dengan IAM.

- Untuk mempelajari cara memberikan akses ke sumber daya Anda di seluruh akun AWS yang Anda miliki, lihat Memberikan akses kepada pengguna IAM di akun AWS lain yang Anda miliki dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses ke sumber daya Anda ke akun AWS pihak ke tiga, lihat Memberikan akses ke akun AWS milik pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Memberikan akses kepada pengguna eksternal yang sah (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara penggunaan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Perbedaan antara peran IAM dan kebijakan berbasis sumber daya di Panduan Pengguna IAM.

# Pencatatan log dan pemantauan di Amazon Aurora

Pemantauan adalah bagian penting dalam menjaga keandalan, ketersediaan, dan kinerja Aurora dan AWS solusi Anda. Anda harus mengumpulkan data pemantauan dari semua bagian AWS solusi Anda sehingga Anda dapat lebih mudah men-debug kegagalan multi-titik jika terjadi. AWS menyediakan beberapa alat untuk memantau sumber daya Amazon Aurora Anda dan menanggapi potensi insiden:

CloudWatch Alarm Amazon

Menggunakan CloudWatch alarm Amazon, Anda menonton satu metrik selama periode waktu yang Anda tentukan. Jika metrik melebihi ambang batas tertentu, pemberitahuan dikirim ke topik atau AWS Auto Scaling kebijakan Amazon SNS. CloudWatch alarm tidak memanggil tindakan karena mereka berada dalam keadaan tertentu. Sebaliknya, status harus diubah dan dipelihara selama jangka waktu tertentu.

AWS CloudTrail Log

CloudTrail menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Aurora. CloudTrail menangkap semua panggilan API untuk Aurora sebagai peristiwa, termasuk panggilan dari konsol dan dari panggilan kode ke operasi Amazon RDS API. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Amazon Aurora, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan. Untuk informasi selengkapnya, lihat Memantau panggilan Amazon Aurora  API AWS CloudTrail.

Pemantauan yang Ditingkatkan

Amazon Aurora menyediakan metrik secara real time untuk sistem operasi (OS) tempat cluster instans . Anda dapat melihat metrik untuk cluster DB menggunakan konsol, atau menggunakan output JSON Pemantauan yang Ditingkatkan dari Amazon CloudWatch Logs dalam sistem pemantauan pilihan Anda. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Ditingkatkan.

Wawasan Performa Amazon RDS

Performance Insights memperluas fitur pemantauan RDS yang ada untuk menggambarkan kinerja database Anda dan membantu Anda menganalisis masalah apa pun yang mempengaruhinya. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan beban basis data dan memfilter beban berdasarkan waktu tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Log Basis Data

Anda dapat melihat, mengunduh, dan menonton log database menggunakan AWS Management Console, AWS CLI, atau RDS API. Untuk informasi selengkapnya, lihat Memantau log Amazon Aurora Amazon.

Amazon Aurora

Amazon Aurora memberikan rekomendasi otomatis untuk sumber daya database. Rekomendasi ini memberikan panduan praktik terbaik dengan menganalisis konfigurasi cluster DB, penggunaan, dan data kinerja. Untuk informasi selengkapnya, lihat .

Amazon Aurora

Amazon Aurora menggunakan Amazon Simple Notification Service (Amazon SNS) untuk memberikan pemberitahuan saat peristiwa Amazon Aurora terjadi. Pemberitahuan ini dapat dalam bentuk pemberitahuan apa pun yang didukung oleh Amazon SNS untuk suatu AWS Wilayah, seperti email, pesan teks, atau panggilan ke titik akhir HTTP. Untuk informasi selengkapnya, lihat Bekerja dengan pemberitahuan RDS acara Amazon.

AWS Trusted Advisor

Trusted Advisor mengacu pada praktik terbaik yang dipelajari dari melayani ratusan ribu AWS pelanggan. Trusted Advisor memeriksa AWS lingkungan Anda dan kemudian membuat rekomendasi ketika ada peluang untuk menghemat uang, meningkatkan ketersediaan dan kinerja sistem, atau membantu menutup kesenjangan keamanan. Semua AWS pelanggan memiliki akses

ke lima Trusted Advisor cek. Pelanggan dengan paket dukungan Bisnis atau Perusahaan dapat melihat semua Trusted Advisor pemeriksaan.

Trusted Advisor memiliki pemeriksaan terkait Amazon Aurora berikut:

- Amazon Aurora Instans DB Menganggur
- Amazon Aurora Security Group Risiko Akses
- Amazon Aurora
- Amazon Aurora Multi-AZ
- Aksesibilitas Instans DB Aurora

Untuk informasi selengkapnya tentang pemeriksaan ini, lihat Praktik terbaik Trusted Advisor (pemeriksaan).

Aliran aktivitas basis data

Aliran aktivitas basis data dapat melindungi basis data Anda dari ancaman internal dengan mengontrol akses DBA ke aliran aktivitas basis data. Dengan demikian, pengumpulan, transmisi, penyimpanan, dan pemrosesan selanjutnya dari aliran aktivitas database berada di luar akses DBAs yang mengelola database. Aliran aktivitas basis data dapat menyediakan sarana keamanan RDS untuk basis data Anda dan memenuhi persyaratan kepatuhan dan peraturan. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Untuk informasi lebih lanjut tentang pemantauan Aurora lihat. Memantau metrik di klaster Amazon Aurora

# Validasi kepatuhan untuk Aurora

Auditor pihak ketiga menilai keamanan dan kepatuhan Aurora sebagai bagian dari AWS beberapa program kepatuhan. Ini termasukSOC,PCI, Fed RAMPHIPAA, dan lainnya.

Untuk daftar AWS layanan dalam lingkup program kepatuhan tertentu, lihat AWS layanan dalam lingkup oleh program kepatuhan. Untuk informasi umum, lihat Program kepatuhan AWS.

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh laporan di AWS Artifact.

Tanggung jawab kepatuhan Anda saat menggunakan Amazon Aurora ditentukan oleh sensitivitas data Anda, tujuan kepatuhan organisasi Anda, serta hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Panduan memulai cepat keamanan dan kepatuhan — Panduan penerapan ini membahas pertimbangan arsitektur dan memberikan langkah-langkah untuk menerapkan lingkungan dasar yang berfokus pada keamanan dan kepatuhan. AWS
- Arsitektur untuk HIPAA Keamanan dan Kepatuhan di Amazon Web Services — Whitepaper ini menjelaskan bagaimana perusahaan dapat menggunakannya AWS untuk membuat HIPAA aplikasi yang sesuai.
- AWS sumber daya kepatuhan - Kumpulan buku kerja dan panduan ini yang mungkin berlaku untuk industri dan lokasi Anda.
- AWS Config AWS Layanan ini menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk sumber daya AWS Anda serta untuk memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.

# Ketangguhan di Amazon Aurora

Infrastruktur global AWS dibangun di sekitar Wilayah dan Zona Ketersediaan AWS. AWS Wilayah menyediakan beberapa Zona Ketersediaan yang terpisah dan terisolasi secara fisik, yang terhubung dengan jaringan latensi rendah, throughput tinggi, dan sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara Zona Ketersediaan tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang Wilayah AWS dan Zona Ketersediaan, lihat Infrastruktur Global AWS.

Selain infrastruktur global AWS, Aurora menawarkan beberapa fitur untuk membantu mendukung kebutuhan ketangguhan dan pencadangan data Anda.

## Pencadangan dan pemulihan

Aurora mencadangkan volume klaster Anda secara otomatis dan mempertahankan data yang dipulihkan selama periode retensi cadangan. Pencadangan Aurora bersifat kontinu dan inkremental, sehingga Anda dapat memulihkan dengan cepat ke titik mana pun dalam periode retensi cadangan. Tidak ada dampak performa atau gangguan layanan basis data saat data cadangan ditulis. Anda dapat menentukan periode retensi cadangan, dari 1 hingga 35 hari, saat Anda membuat atau memodifikasi klaster DB.

Jika Anda ingin mempertahankan cadangan di luar periode retensi cadangan, Anda juga dapat mengambil snapshot data itu di volume klaster Anda. Aurora mempertahankan data pemulihan inkremental selama masa retensi cadangan. Karena itu, Anda perlu membuat cuplikan hanya untuk data yang ingin Anda pertahankan setelah periode retensi cadangan. Anda dapat membuat klaster DB baru dari snapshot.

Anda dapat memulihkan data Anda dengan membuat klaster DB Aurora baru dari data cadangan yang dipertahankan Aurora, atau dari snapshot klaster DB yang telah Anda simpan. Anda dapat membuat dengan cepat salinan baru klaster DB dari data cadangan ke sebarang titik waktu selama masa retensi cadangan Anda. Sifat kontinu dan inkremental cadangan Aurora selama masa retensi cadangan berarti Anda tidak perlu sering-sering mengambil snapshot data untuk meningkatkan waktu pemulihan.

Untuk informasi selengkapnya, lihat Mencadangkan dan memulihkan klaster DB Amazon Aurora.

## Replikasi

Aurora Replica adalah titik akhir independen dalam klaster DB Aurora, yang paling berguna untuk menskalakan operasi baca dan meningkatkan ketersediaan. Hingga 15 Aurora Replica dapat disebarkan ke seluruh Zona Ketersediaan yang dijangkau klaster DB di dalam sebuah Wilayah AWS. Volume klaster DB terdiri atas beberapa salinan data untuk klaster DB. Namun, data dalam volume klaster disajikan berupa volume tunggal logis ke instans DB utama dan ke Aurora Replica dalam klaster DB. Jika instans DB primer gagal, Aurora Replica dipromosikan menjadi instans DB primer.

Aurora juga mendukung opsi-opsi replikasi yang khusus untuk Aurora MySQL dan Aurora PostgreSQL.

Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

## Failover

Aurora menyimpan salinan data dalam klaster DB di beberapa Zona Ketersediaan dalam satu Wilayah AWS. Penyimpanan ini terjadi terlepas dari apakah instans DB di klaster DB mencakup beberapa Zona Ketersediaan atau tidak. Saat Anda membuat Aurora Replica di beberapa Zona Ketersediaan, Aurora menyediakan dan memelihara secara otomatis semuanya secara sinkron. Instans DB utama direplikasi secara sinkron di beberapa Zona Ketersediaan ke Aurora Replica untuk menyediakan redundansi data, menghilangkan pembekuan I/O, dan meminimalkan lonjakan latensi selama pencadangan sistem. Menjalankan klaster DB dengan ketersediaan tinggi dapat meningkatkan ketersediaan selama pemeliharaan sistem terencana, dan membantu melindungi basis data Anda terhadap kegagalan dan gangguan Zona Ketersediaan.

Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.

# Keamanan infrastruktur di Aurora

Sebagai layanan terkelola, Amazon Relational Database Service dilindungi AWS oleh keamanan jaringan global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat Keamanan AWS Cloud. Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja yang  AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan API panggilan yang AWS dipublikasikan untuk mengakses Amazon RDS melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Transportasi (TLS). Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Suite cipher dengan kerahasiaan maju yang sempurna (PFS) seperti (Ephemeral Diffie-Hellman) atau DHE (Elliptic Curve Ephemeral Diffie-Hellman). ECDHE Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani dengan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan IAM prinsipal. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Selain itu, Aurora menawarkan fitur untuk membantu mendukung keamanan infrastruktur.

## Grup keamanan

Grup keamanan mengendalikan akses yang dimiliki oleh lalu lintas masuk dan keluar dari klaster basis data. Secara default, akses jaringan ke klaster basis data dinonaktifkan. Anda dapat menentukan aturan dalam grup keamanan yang memungkinkan akses dari rentang alamat IP, port, atau grup keamanan. Setelah aturan masuk dikonfigurasikan, aturan yang sama berlaku untuk semua klaster basis data yang dikaitkan dengan grup keamanan itu.

Untuk informasi selengkapnya, lihat Mengontrol akses dengan grup keamanan.

## Aksesibilitas publik

Saat meluncurkan instans DB di dalam virtual private cloud (VPC) berdasarkan VPC layanan Amazon, Anda dapat mengaktifkan atau menonaktifkan aksesibilitas publik untuk instans DB

tersebut. Untuk menentukan apakah instans DB yang Anda buat memiliki DNS nama yang menyelesaikan ke alamat IP publik, Anda menggunakan parameter aksesibilitas Publik. Dengan menggunakan parameter ini, Anda dapat menetapkan apakah ada akses publik ke instans basis data. Anda dapat mengubah instans basis data untuk mengaktifkan atau menonaktifkan aksesibilitas publik dengan mengubah parameter Aksesibilitas publik.

Untuk informasi selengkapnya, lihat Menyembunyikan klaster DB dalam VPC dari internet.

> **Note**
>
> Jika instans DB Anda ada VPC tetapi tidak dapat diakses publik, Anda juga dapat menggunakan koneksi AWS Site-to-Site atau VPN AWS Direct Connect koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.

# API Amazon RDS dan titik akhir VPC antarmuka (AWS PrivateLink)

Anda dapat membuat koneksi privat antara titik akhir VPC dan API Amazon RDS dengan membuat titik akhir VPC antarmuka. Titik akhir antarmuka didukung oleh AWS PrivateLink.

AWS PrivateLink memungkinkan Anda mengakses operasi Amazon RDS API secara pribadi tanpa gateway internet, perangkat NAT, koneksi VPN, atau koneksi. AWS Direct Connect Instans DB dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan titik akhir API Amazon RDS untuk meluncurkan, memodifikasi, atau menghentikan instans DB dan klaster DB. Instans DB Anda juga tidak memerlukan alamat IP publik untuk menggunakan salah satu dari operasi API RDS yang tersedia. Lalu lintas antara VPC Anda dan Amazon RDS tidak keluar dari jaringan Amazon.

Setiap titik akhir antarmuka direpresentasikan oleh satu atau beberapa antarmuka jaringan elastis di subnet Anda. Untuk informasi selengkapnya tentang antarmuka jaringan elastis, lihat Antarmuka jaringan elastis dalam Panduan Pengguna Amazon EC2.

Untuk informasi selengkapnya tentang titik akhir VPC, lihat Titik akhir VPC Antarmuka () di AWS PrivateLink Panduan Pengguna Amazon VPC. Untuk informasi selengkapnya tentang operasi API RDS, lihat Referensi API Amazon RDS.

Anda tidak memerlukan titik akhir VPC antarmuka untuk terhubung ke klaster DB. Untuk informasi selengkapnya, lihat Skenario untuk mengakses cluster DB di VPC.

## Pertimbangan untuk titik akhir VPC

Sebelum Anda menyiapkan titik akhir VPC antarmuka untuk titik akhir API Amazon RDS, pastikan Anda meninjau Properti dan batasan titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Semua operasi API RDS yang relevan dengan pengelolaan sumber daya Amazon Aurora tersedia dari VPC Anda menggunakan AWS PrivateLink.

Kebijakan titik akhir VPC didukung untuk titik akhir API RDS. Secara default, akses penuh ke operasi API RDS diizinkan melalui titik akhir. Untuk informasi selengkapnya, lihat Mengontrol akses ke layanan dengan titik akhir VPC dalam Panduan Pengguna Amazon VPC.

## Ketersediaan

Amazon RDS API saat ini mendukung titik akhir VPC di Wilayah berikut: AWS

- AS Timur (Ohio)
- AS Timur (Virginia Utara)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Afrika (Cape Town)
- Asia Pasifik (Hong Kong)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- (Canada (Central)
- Kanada Barat (Calgary)
- China (Beijing)
- Tiongkok (Ningxia)
- Eropa (Frankfurt)
- Eropa (Zürich)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)
- Eropa (Stockholm)
- Eropa (Milan)
- Israel (Tel Aviv)
- Timur Tengah (Bahrain)
- Amerika Selatan (Sao Paulo)
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

## Membuat titik akhir VPC antarmuka untuk API Amazon RDS

Anda dapat membuat titik akhir VPC untuk Amazon RDS API menggunakan konsol VPC Amazon atau (). AWS Command Line Interface AWS CLI Untuk informasi selengkapnya, lihat Membuat titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Buat titik akhir VPC untuk API Amazon RDS menggunakan nama layanan `com.amazonaws.region.rds`.

Tidak termasuk AWS Wilayah di China, jika Anda mengaktifkan DNS pribadi untuk titik akhir, Anda dapat membuat permintaan API ke Amazon RDS dengan titik akhir VPC menggunakan nama DNS default untuk Wilayah, misalnya. AWS `rds.us-east-1.amazonaws.com` Untuk AWS Wilayah China (Beijing) dan China (Ningxia), Anda dapat membuat permintaan API dengan titik akhir VPC `rds-api.cn-north-1.amazonaws.com.cn` menggunakan `rds-api.cn-northwest-1.amazonaws.com.cn` dan, masing-masing.

Untuk informasi selengkapnya, lihat Mengakses layanan melalui titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

## Membuat kebijakan titik akhir VPC untuk API Amazon RDS

Anda dapat menyisipkan kebijakan titik akhir ke titik akhir VPC yang mengontrol akses ke API Amazon RDS. Kebijakan titik akhir menentukan informasi berikut:

- Prinsipal yang dapat melakukan tindakan.
- Tindakan yang dapat dilakukan.
- Sumber daya yang menjadi target tindakan.

Untuk informasi selengkapnya, lihat Mengontrol akses ke layanan dengan titik akhir VPC dalam Panduan Pengguna Amazon VPC.

Contoh: Kebijakan titik akhir VPC untuk tindakan API Amazon RDS

Berikut ini adalah contoh kebijakan titik akhir untuk API Amazon RDS. Jika dilampirkan ke sebuah titik akhir, kebijakan ini memberikan akses ke tindakan API Amazon RDS untuk semua prinsipal di semua sumber daya.

```
{
```

```
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

Contoh: Kebijakan titik akhir VPC yang menolak semua akses dari akun tertentu AWS

Kebijakan titik akhir VPC berikut menolak `123456789012` semua akses AWS akun ke sumber daya menggunakan titik akhir. Kebijakan ini mengizinkan semua tindakan dari akun lainnya.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

# Praktik terbaik keamanan untuk Amazon Aurora

Gunakan akun AWS Identity and Access Management (IAM) untuk mengontrol akses ke operasi Amazon RDS API, terutama operasi yang membuat, memodifikasi, atau menghapus sumber daya Amazon . Sumber daya tersebut termasuk cluster DB, grup keamanan, dan grup parameter.

- Buat pengguna individual untuk setiap orang yang mengelola sumber daya Amazon Aurora, termasuk Anda sendiri. Jangan gunakan kredensi AWS root untuk mengelola sumber daya Amazon .
- Beri setiap pengguna set izin minimum yang diperlukan untuk melakukan tugas-tugasnya.
- Gunakan grup IAM untuk mengelola izin secara efektif bagi beberapa pengguna.
- Putar kredensial IAM Anda secara rutin.
- Konfigurasikan AWS Secrets Manager untuk secara otomatis memutar rahasia untuk Amazon Aurora. Untuk informasi selengkapnya, lihat Memutar rahasia AWS Secrets Manager Anda di Panduan Pengguna AWS Secrets Manager . Anda juga dapat mengambil kredensialnya dari AWS Secrets Manager pemrograman. Untuk informasi selengkapnya, lihat Mengambil nilai rahasia di Panduan Pengguna AWS Secrets Manager .

Untuk informasi selengkapnya tentang keamanan Amazon Aurora, lihat. Keamanan dalam Amazon Aurora Untuk informasi selengkapnya tentang IAM, lihat AWS Identity and Access Management. Untuk informasi tentang praktik terbaik IAM, lihat Praktik terbaik IAM.

AWS Security Hub menggunakan kontrol keamanan untuk mengevaluasi konfigurasi sumber daya dan standar keamanan untuk membantu Anda mematuhi berbagai kerangka kerja kepatuhan. Untuk informasi selengkapnya tentang penggunaan Security Hub guna mengevaluasi sumber daya RDS, lihat Kontrol Amazon Relational Database Service di Panduan Pengguna AWS Security Hub .

Anda dapat memantau penggunaan RDS yang berkaitan dengan praktik terbaik keamanan dengan menggunakan Security Hub. Untuk informasi lebih lanjut, lihat Apa itu AWS Security Hub? .

Gunakan AWS Management Console, API AWS CLI, atau RDS untuk mengubah kata sandi untuk pengguna utama Anda. Jika Anda menggunakan alat lain, seperti klien SQL, untuk mengubah kata sandi pengguna utama, hak istimewa pengguna kemungkinan dapat terhapus secara tidak sengaja.

Amazon GuardDuty adalah layanan pemantauan keamanan berkelanjutan yang menganalisis dan memproses berbagai sumber data, termasuk aktivitas login Amazon RDS. Ini menggunakan umpan intelijen ancaman dan pembelajaran mesin untuk mengidentifikasi perilaku login yang tidak terduga, berpotensi tidak sah, mencurigakan, dan aktivitas berbahaya di lingkungan Anda. AWS

Saat Amazon GuardDuty RDS Protection mendeteksi upaya login yang berpotensi mencurigakan atau anomali yang menunjukkan ancaman terhadap database Anda, GuardDuty buat temuan baru dengan detail tentang database yang berpotensi dikompromikan. Untuk informasi selengkapnya, lihat Memantau ancaman dengan Perlindungan Amazon GuardDuty RDS untuk Amazon Aurora.

# Mengontrol akses dengan grup keamanan

Grup keamanan VPC mengontrol akses yang dimiliki lalu lintas masuk dan keluar dari cluster DB. Secara default, akses jaringan dimatikan untuk cluster DB. Anda dapat menentukan aturan dalam grup keamanan yang mengizinkan akses dari rentang alamat IP, port, atau grup keamanan. Setelah aturan ingress dikonfigurasi, aturan yang sama berlaku untuk semua cluster DB yang terkait dengan grup keamanan tersebut. Anda dapat menentukan hingga 20 aturan dalam satu grup keamanan.

## Ikhtisar grup keamanan VPC

Setiap aturan grup keamanan VPC memungkinkan sumber tertentu untuk mengakses cluster DB di VPC yang terkait dengan grup keamanan VPC tersebut. Sumbernya dapat berupa rentang alamat (misalnya, 203.0.113.0/24), atau grup keamanan VPC lain. Dengan menentukan grup keamanan VPC sebagai sumber, Anda mengizinkan lalu lintas masuk dari semua instans (biasanya server aplikasi) yang menggunakan grup keamanan VPC sumber. Grup keamanan VPC dapat memiliki aturan yang mengatur lalu lintas masuk dan keluar. Namun, aturan lalu lintas keluar biasanya tidak berlaku untuk cluster DB. Aturan lalu lintas keluar hanya berlaku jika cluster DB bertindak sebagai klien. Anda harus menggunakan EC2API Amazon atau opsi Grup Keamanan di konsol VPC untuk membuat grup keamanan VPC.

Saat Anda membuat aturan untuk grup keamanan VPC yang mengizinkan akses ke cluster di VPC Anda, Anda harus menentukan port untuk setiap rentang alamat yang diizinkan oleh aturan tersebut. Misalnya, jika Anda ingin mengaktifkan akses Secure Shell (SSH) untuk instans di VPC, buat aturan yang mengizinkan akses ke port 22 TCP untuk rentang alamat tertentu.

Anda dapat mengonfigurasi beberapa grup keamanan VPC yang mengizinkan akses ke port yang berbeda untuk instans yang berbeda di VPC Anda. Misalnya, Anda dapat membuat grup keamanan VPC yang memungkinkan akses ke port 80 TCP untuk server web di VPC Anda. Anda kemudian dapat membuat grup keamanan VPC lain yang memungkinkan akses ke port TCP 3306 untuk RDS untuk instance MySQL DB di VPC Anda.

> **Note**
>
> Dalam klaster DB Aurora, grup keamanan VPC yang terkait dengan klaster DB juga terkait dengan semua instans DB dalam klaster DB. Jika Anda mengubah grup keamanan VPC untuk klaster atau instans DB, perubahan akan diterapkan secara otomatis ke semua instans DB dalam klaster DB.

Untuk informasi selengkapnya tentang grup keamanan VPC, lihat Grup keamanan di Panduan Pengguna Amazon Virtual Private Cloud.

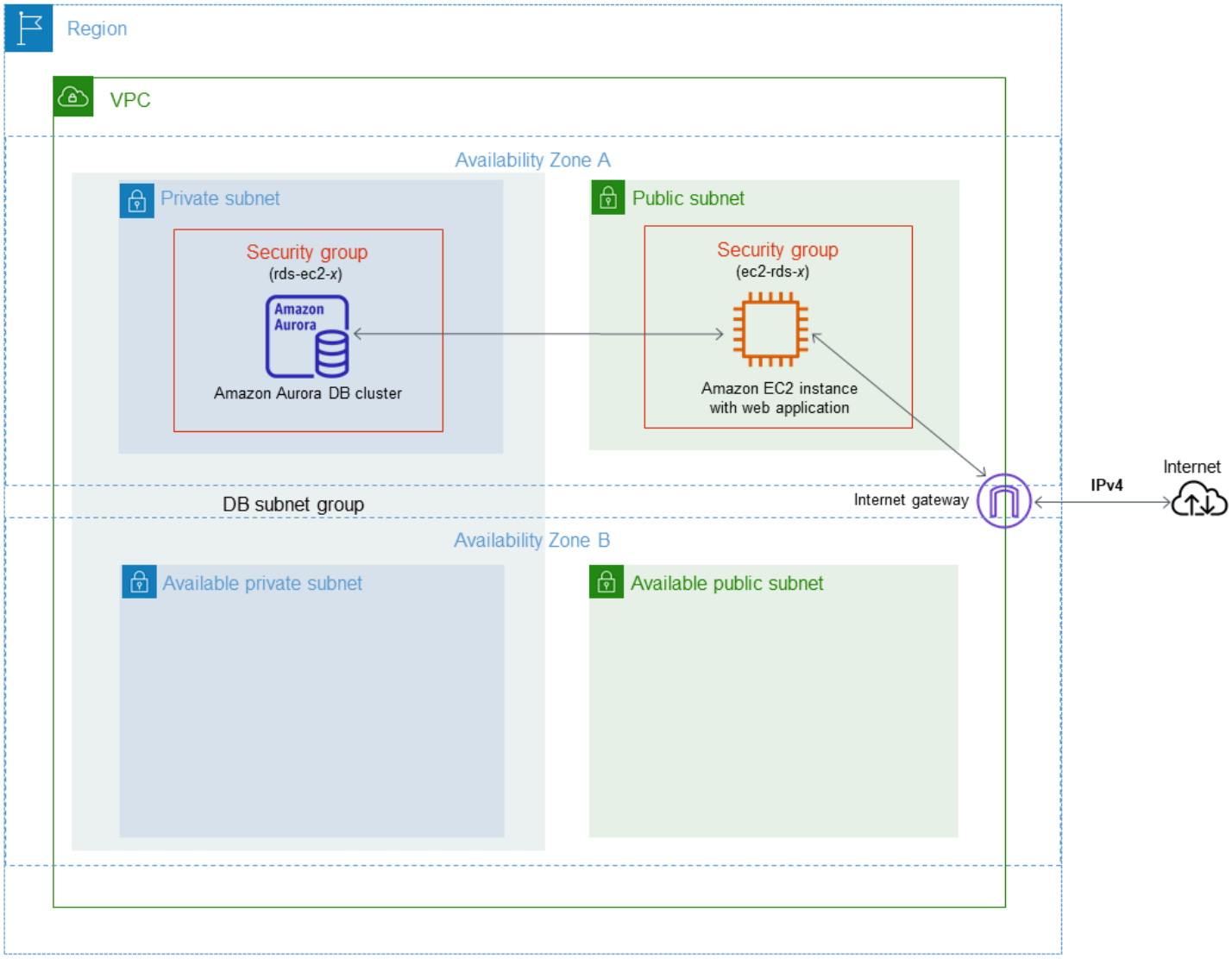
> **Note**
>
> Jika cluster DB Anda berada dalam VPC tetapi tidak dapat diakses publik, Anda juga dapat menggunakan koneksi AWS Site-to-Site VPN atau AWS Direct Connect koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.

## Skenario grup keamanan

Penggunaan umum cluster DB di VPC adalah untuk berbagi data dengan server aplikasi yang berjalan di EC2 instance Amazon di VPC yang sama, yang diakses oleh aplikasi klien di luar VPC. Untuk skenario ini, Anda menggunakan halaman RDS dan VPC pada AWS Management Console atau operasi RDS EC2 dan API untuk membuat instance dan grup keamanan yang diperlukan:

1. Buat grup keamanan VPC (misalnya, `sg-0123ec2example`) dan tentukan aturan masuk yang menggunakan alamat IP aplikasi klien sebagai sumber. Grup keamanan ini memungkinkan aplikasi klien Anda terhubung ke EC2 instance di VPC yang menggunakan grup keamanan ini.
2. Buat EC2 instance untuk aplikasi dan tambahkan EC2 instance ke grup keamanan VPC (`sg-0123ec2example`) yang Anda buat di langkah sebelumnya.
3. Buat grup keamanan VPC kedua (misalnya, `sg-6789rdsexample`) dan buat aturan baru dengan menentukan grup keamanan VPC yang Anda buat di langkah 1 (`sg-0123ec2example`) sebagai sumbernya.
4. Buat cluster DB baru dan tambahkan cluster DB ke grup keamanan VPC (`sg-6789rdsexample`) yang Anda buat di langkah sebelumnya. Saat Anda membuat cluster DB, gunakan nomor port yang sama dengan yang ditentukan untuk aturan grup keamanan VPC (`sg-6789rdsexample`) yang Anda buat di langkah 3.

Diagram berikut menunjukkan skenario ini.




Untuk petunjuk terperinci tentang mengonfigurasi VPC untuk skenario ini, lihat. Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 Untuk informasi selengkapnya tentang menggunakan VPC, lihat. Amazon VPC dan Aurora

## Membuat grup keamanan VPC

Anda dapat membuat grup keamanan VPC untuk instans DB menggunakan konsol VPC. Untuk informasi tentang pembuatan grup keamanan, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan dan Grup Keamanan dalam Panduan Pengguna Amazon Virtual Private Cloud.

## Mengaitkan grup keamanan dengan klaster DB

Anda dapat mengaitkan grup keamanan dengan cluster DB menggunakan Modify cluster di konsol RDS, `ModifyDBCluster` Amazon RDS API, atau perintah. `modify-db-cluster` AWS CLI

Contoh CLI berikut mengaitkan grup VPC tertentu dan menghapus grup keamanan DB dari cluster DB

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

Untuk informasi tentang memodifikasi cluster DB, lihatMemodifikasi klaster DB Amazon Aurora.

# Hak akses akun pengguna master

Saat Anda membuat cluster DB baru, pengguna master default yang Anda gunakan mendapatkan hak istimewa tertentu untuk cluster DB tersebut. Anda tidak dapat mengubah nama pengguna master setelah cluster DB dibuat.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses paling rendah yang diperlukan untuk aplikasi Anda.

> **Note**
>
> Jika Anda secara tidak sengaja menghapus izin untuk pengguna master, Anda dapat memulihkannya dengan memodifikasi cluster DB dan menyetel kata sandi pengguna master baru. Untuk informasi selengkapnya tentang memodifikasi cluster DB, lihat .

Tabel berikut menunjukkan hak akses dan peran basis data yang diperoleh pengguna master untuk masing-masing mesin basis data.

| Mesin basis data | Hak akses sistem | Peran basis data |
| --- | --- | --- |
| Aurora MySQL | Versi 2:<br><br>`ALTER,ALTER ROUTINE, CREATE,CREATE ROUTINE,CREATE TEMPORARY TABLES,CREATE USER,CREATE VIEW,DELETE,DROP, EVENT,EXECUTE,GRANT OPTION,INDEX,INSERT, LOAD FROM S3,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATI ON CLIENT , REPLICATION SLAVE ,SELECT, SELECT INTO S3,SHOW DATABASES , SHOW VIEW,TRIGGER, UPDATE` | — |
| | Versi 3:<br><br>`ALTER,APPLICATION_PASSWORD_ADMIN , ALTER ROUTINE,CONNECTION_ADMIN , CREATE,CREATE ROLE,CREATE ROUTINE,CREATE TEMPORARY TABLES, CREATE USER,CREATE VIEW, DELETE,DROP,DROP ROLE, EVENT,EXECUTE,INDEX, INSERT,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATION CLIENT , REPLICATI ON SLAVE ,ROLE_ADMIN , SET_USER_ ID ,SELECT,SHOW DATABASES ,SHOW VIEW,TRIGGER, UPDATE,XA_RECOVER_ADMIN`<br><br>Dimulai dengan Aurora MySQL versi 3.04.0, pengguna master juga mendapatkan hak istimewa. `SHOW_ROUTINE`<br><br>Dimulai dengan Aurora MySQL versi 3.09.0, pengguna master juga mendapatkan,, dan hak istimewa. `FLUSH_OPTIMIZER_COSTS` | `rds_superuser_role`<br><br>Untuk informasi selengkapnya tentang rds_superuser_role , lihat. Model hak akses berbasis peran |

| Mesin basis data | Hak akses sistem | Peran basis data |
| --- | --- | --- |
| | `FLUSH_STATUS FLUSH_TABLES FLUSH_USER_RESOURCES` | |
| Aurora PostgreSQL | `LOGIN,NOSUPERUSER , INHERIT,CREATEDB, CREATEROLE ,NOREPLICATION ,VALID UNTIL 'infinity'` | `RDS_SUPERUSER`<br><br>Untuk informasi selengkapnya tentang RDS_SUPERUSER, lihat. Memahami peran dan izin PostgreSQL |

# Menggunakan peran terkait layanan untuk Amazon Aurora

Amazon Aurora AWS Identity and Access Management menggunakan peran terkait layanan (IAM). Peran yang terkait layanan adalah jenis peran IAM unik yang ditautkan langsung ke Amazon Aurora. Peran terkait layanan telah ditentukan sebelumnya oleh Aurora dan menyertakan semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran terkait layanan memudahkan penggunaan Amazon Aurora karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Amazon Aurora menentukan izin peran terkait layanannya, dan kecuali jika ditentukan lain, hanya Amazon Aurora yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, serta bahwa kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran hanya setelah terlebih dahulu menghapus sumber daya terkaitnya. Cara ini akan melindungi sumber daya Amazon Aurora karena Anda tidak dapat menghapus izin untuk mengakses sumber daya tersebut secara tidak sengaja.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat Layanan AWS yang berfungsi dengan IAM lalu cari layanan yang menampilkan Ya pada kolom Peran Terkait Layanan. Pilih Ya dengan sebuah tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

## Izin peran terkait layanan untuk Amazon Aurora

Peran terkait layanan AWSService RoleFor RDS mempercayai layanan berikut untuk mengambil peran:

- `rds.amazonaws.com`

Peran terkait layanan ini memiliki kebijakan izin yang menyertainya bernama `AmazonRDSServiceRolePolicy` yang memberikannya izin untuk beroperasi di akun Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat Amazon RDSService RolePolicy di Panduan Referensi Kebijakan AWS Terkelola.

> **Note**
>
> Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan. Jika Anda menemukan pesan kesalahan berikut:
> Tidak dapat membuat sumber daya. Verifikasi bahwa Anda memiliki izin untuk membuat peran terkait layanan. Jika tidak, tunggu dan coba lagi nanti.
> Pastikan Anda telah mengaktifkan izin berikut:
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> Untuk informasi selengkapnya, lihat Izin peran terkait layanan dalam Panduan Pengguna IAM.

## Membuat peran terkait layanan untuk Amazon Aurora

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat klaster DB, Amazon Aurora membuat peran terkait layanan untuk Anda.

> **Important**
>
> Jika Anda menggunakan layanan Amazon Aurora sebelum 1 Desember 2017, ketika mulai mendukung peran terkait layanan, maka Amazon di akun Anda. AWSService RoleFor Untuk mempelajari lebih lanjut, lihat Peran baru muncul di AWS akun saya.

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda membuat klaster DB, Amazon Aurora membuat peran terkait layanan untuk Anda.

## Mengedit peran terkait layanan untuk Amazon Aurora

Amazon Aurora tidak memungkinkan Anda mengedit AWSService RoleFor peran terkait layanan RDS. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin merujuk peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit peran terkait layanan dalam Panduan Pengguna IAM.

## Menghapus peran terkait layanan untuk Amazon Aurora

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, kami merekomendasikan Anda menghapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Namun, Anda harus menghapus semua klaster DB Anda sebelum Anda dapat menghapus peran terkait layanan.

Membersihkan peran terkait layanan

Sebelum dapat menggunakan IAM untuk menghapus peran tertaut layanan, Anda harus mengonfirmasi terlebih dahulu bahwa peran tersebut tidak memiliki sesi aktif dan menghapus sumber daya yang digunakan oleh peran tersebut.

Untuk memeriksa apakah peran terkait layanan memiliki sesi aktif di konsol IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi konsol IAM, pilih Peran. Kemudian pilih nama (bukan kotak centang) dari peran AWSService RoleFor RDS.
3. Pada halaman Ringkasan untuk peran yang dipilih, pilih tab Terakhir Diakses.
4. Pada tab Terakhir Diakses, tinjau aktivitas terbaru untuk peran terkait layanan.

> **Note**
>
> Jika Anda tidak yakin apakah Amazon Aurora menggunakan peran RDS, Anda dapat mencoba menghapus peran AWSService RoleFor tersebut. Jika layanan ini menggunakan peran tersebut, peran tidak dapat dihapus dan Anda dapat melihat

Wilayah AWS tempat peran tersebut digunakan. Jika peran tersebut sedang digunakan, Anda harus menunggu hingga sesi ini berakhir sebelum dapat menghapus peran tersebut. Anda tidak dapat mencabut sesi untuk peran terkait layanan.

Jika Anda ingin menghapus peran AWSService RoleFor RDS, Anda harus terlebih dahulu menghapus semua cluster DB Anda.

Menghapus semua klaster Anda

Gunakan salah satu prosedur berikut untuk menghapus satu klaster. Ulangi prosedur untuk setiap klaster Anda.

Untuk menghapus sebuah klaster (konsol)

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di daftar Basis data pilih klaster yang ingin Anda hapus.
3. Untuk Tindakan Klaster, pilih Hapus.
4. Pilih Hapus.

Untuk menghapus sebuah klaster (CLI)

Lihat `delete-db-cluster` dalam Referensi Perintah AWS CLI .

Untuk menghapus sebuah klaster (API)

Lihat `DeleteDBCluster` dalam Referensi API Amazon RDS.

Anda dapat menggunakan konsol IAM, IAM CLI, atau IAM API untuk menghapus peran terkait layanan RDS. AWSService RoleFor Untuk informasi selengkapnya, lihat Menghapus peran terkait layanan dalam Panduan Pengguna IAM.

## Izin peran terkait layanan untuk Amazon RDS Beta

Amazon Aurora menggunakan peran terkait layanan bernama `AWSServiceRoleForRDSBeta` untuk memungkinkan Amazon Aurora AWS memanggil layanan atas nama sumber daya RDS DB Anda.

Peran AWSService RoleFor RDSBeta terkait layanan mempercayai layanan berikut untuk mengambil peran:

- `rds.amazonaws.com`

Peran terkait layanan ini memiliki kebijakan izin yang menyertainya bernama `AmazonRDSBetaServiceRolePolicy` yang memberikannya izin untuk beroperasi di akun Anda. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: A mazonRDSBeta ServiceRolePolicy.

> **Note**
>
> Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan. Jika Anda menemukan pesan kesalahan berikut:
> Tidak dapat membuat sumber daya. Verifikasi bahwa Anda memiliki izin untuk membuat peran terkait layanan. Jika tidak, tunggu dan coba lagi nanti.
> Pastikan Anda telah mengaktifkan izin berikut:
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSBetaServiceRolePolicy",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"custom.rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> Untuk informasi selengkapnya, lihat Izin peran terkait layanan dalam Panduan Pengguna IAM.

## Peran terkait layanan untuk Amazon RDS Preview

Amazon Aurora menggunakan peran terkait layanan bernama `AWSServiceRoleForRDSPreview` untuk memungkinkan Amazon Aurora AWS memanggil layanan atas nama sumber daya RDS DB Anda.

Peran AWSService RoleFor RDSPreview terkait layanan mempercayai layanan berikut untuk mengambil peran:

- `rds.amazonaws.com`

Peran terkait layanan ini memiliki kebijakan izin yang menyertainya bernama `AmazonRDSPreviewServiceRolePolicy` yang memberikannya izin untuk beroperasi di akun Anda. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: A mazonRDSPreview ServiceRolePolicy.

> **Note**
>
> Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan. Jika Anda menemukan pesan kesalahan berikut:
> Tidak dapat membuat sumber daya. Verifikasi bahwa Anda memiliki izin untuk membuat peran terkait layanan. Jika tidak, tunggu dan coba lagi nanti.
> Pastikan Anda telah mengaktifkan izin berikut:
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSPreviewServiceRolePolicy",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"custom.rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> Untuk informasi selengkapnya, lihat Izin peran terkait layanan dalam Panduan Pengguna IAM.

# Amazon VPC dan Aurora

Amazon Virtual Private Cloud (AmazonVPC) memungkinkan Anda untuk meluncurkan AWS sumber daya, seperti cluster Aurora DB, menjadi virtual private cloud (). VPC

Ketika Anda menggunakanVPC, Anda memiliki kendali atas lingkungan jaringan virtual Anda. Anda dapat memilih rentang alamat IP Anda sendiri, membuat subnet, serta mengonfigurasi perutean dan daftar kontrol akses. Tidak ada biaya tambahan untuk menjalankan cluster DB Anda di fileVPC.

Akun memiliki defaultVPC. Semua cluster DB baru dibuat secara default VPC kecuali Anda menentukan sebaliknya.

Topik


- Menggunakan klaster DB di VPC
- Skenario untuk mengakses cluster DB di VPC
- Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)


Berikut ini, Anda dapat menemukan diskusi tentang VPC fungsionalitas yang relevan dengan Amazon Aurora DB cluster. Untuk informasi selengkapnya tentang AmazonVPC, lihat Panduan VPCMemulai Amazon dan Panduan VPC Pengguna Amazon.

## Menggunakan klaster DB di VPC

Klaster DB Anda berada di cloud privat virtual (VPC). VPC adalah jaringan virtual yang secara logis terisolasi dari jaringan virtual lain di Cloud. AWS Amazon VPC memungkinkan Anda untuk meluncurkan AWS sumber daya, seperti instans Amazon Aurora Amazon atau instans Amazon EC2 , ke dalam VPC. VPC dapat berupa VPC default dari akun Anda atau VPC yang Anda buat. Semua VPCs terkait dengan AWS akun Anda.

VPC default memiliki tiga subnet yang dapat digunakan untuk mengisolasi sumber daya di dalam VPC. VPC default juga memiliki gateway internet yang dapat digunakan untuk memberikan akses ke sumber daya di dalam VPC dari luar VPC.

Untuk daftar skenario yang melibatkan Aurora instans cluster di VPC , lihat. Skenario untuk mengakses cluster DB di VPC

Topik

- Bekerja dengan klaster DB dalam VPC
- Menggunakan grup subnet DB
- Subnet bersama
- Penentuan alamat IP Amazon Aurora
- Menyembunyikan klaster DB dalam VPC dari internet
- Membuat klaster DB dalam VPC

Dalam tutorial berikut, Anda dapat mempelajari cara membuat VPC yang dapat Anda gunakan untuk skenario Amazon Aurora yang umum:

- Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

## Bekerja dengan klaster DB dalam VPC

Berikut ini beberapa tips untuk menggunakan klaster DB di VPC:

- VPC Anda harus memiliki setidaknya dua subnet. Subnet ini harus berada di dua Availability Zone yang berbeda di Wilayah AWS mana Anda ingin menyebarkan cluster DB Anda. Subnet adalah segmen dari rentang alamat IP VPC yang dapat Anda tentukan dan gunakan untuk mengelompokkan klaster DB berdasarkan kebutuhan keamanan dan operasional Anda.
- Jika ingin klaster DB dalam VPC dapat diakses publik, pastikan untuk mengaktifkan nama host DNS dan resolusi DNS pada atribut VPC.
- VPC Anda harus memiliki grup subnet DB yang Anda buat. Anda membuat grup subnet DB dengan menentukan subnet yang Anda buat. Amazon Aurora memilih subnet dan alamat IP dalam subnet tersebut untuk dikaitkan dengan instans DB utama di klaster DB Anda. Instans DB primer menggunakan Zona Ketersediaan yang berisi subnet tersebut.
- VPC Anda harus memiliki grup keamanan VPC yang memungkinkan akses ke klaster DB.

  Untuk informasi selengkapnya, lihat Skenario untuk mengakses cluster DB di VPC.
- Blok CIDR di setiap subnet Anda harus cukup besar untuk mengakomodasi alamat IP cadangan untuk Amazon Aurora untuk digunakan selama aktivitas pemeliharaan, termasuk failover dan penskalaan komputasi. Misalnya, rentang seperti 10.0.0.0/24 dan 10.0.1.0/24 biasanya memiliki kapasitas cukup besar.

- VPC dapat memiliki atribut penghunian instans, baik yang bersifat default atau khusus. Semua default VPCs memiliki atribut penyewaan instance yang disetel ke default, dan VPC default dapat mendukung kelas instans DB apa pun.

  Jika Anda memilih untuk memiliki cluster DB di VPC khusus tempat atribut penyewaan instans disetel ke dedicated, kelas instans DB dari cluster instans DB Anda harus menjadi salah satu jenis EC2 khusus Amazon yang disetujui. Misalnya, instance EC2 khusus r5.large sesuai dengan kelas instans DB db.r5.large. Untuk informasi tentang penghunian instans dalam VPC, lihat Instans khusus dalam Panduan Pengguna Amazon Elastic Compute Cloud.

  Untuk informasi selengkapnya tentang jenis instans yang dapat berada dalam instans khusus, lihat instans EC2 khusus Amazon di halaman EC2 harga Amazon.

  > **Note**
  >
  > Jika Anda menetapkan atribut penghunian instans ke khusus untuk klaster DB, hal tersebut tidak menjamin bahwa klaster DB akan berjalan di host khusus.

## Menggunakan grup subnet DB

Subnet adalah segmen dari rentang alamat IP VPC yang Anda tetapkan untuk mengelompokkan sumber daya Anda berdasarkan kebutuhan keamanan dan operasional. Grup subnet DB adalah kumpulan subnet (biasanya privat) yang Anda buat dalam VPC lalu Anda tetapkan untuk klaster DB Anda. Dengan menggunakan grup subnet DB, Anda dapat menentukan VPC tertentu saat membuat cluster instance menggunakan AWS CLI API atau RDS. Jika menggunakan konsol, Anda dapat memilih VPC dan grup subnet yang ingin Anda gunakan.

Setiap subnet DB harus memiliki subnet di setidaknya dua Zona Ketersediaan dalam satu Wilayah AWS tertentu. Saat membuat klaster DB di VPC, Anda perlu memilih grup subnet DB untuknya. Dari grup subnet DB, Amazon Aurora memilih subnet dan alamat IP dalam subnet tersebut untuk dikaitkan dengan instans DB primer di klaster DB Anda. DB menggunakan Zona Ketersediaan yang berisi subnet tersebut. Aurora Amazon selalu memberikan alamat IP dari subnet yang memiliki ruang alamat IP gratis.

Subnet dalam grup subnet DB bersifat publik atau privat. Subnet bersifat publik atau pribadi, tergantung pada konfigurasi yang Anda tetapkan untuk daftar kontrol akses jaringan (jaringan ACLs) dan tabel perutean mereka. Agar klaster DB dapat diakses publik, semua subnet dalam grup subnet

DB-nya harus bersifat publik. Jika subnet yang terkait dengan klaster DB yang dapat diakses publik berubah dari publik menjadi privat, hal tersebut dapat memengaruhi ketersediaan klaster DB.

Untuk membuat grup subnet DB yang mendukung mode dual-stack, pastikan bahwa setiap subnet yang Anda tambahkan ke grup subnet DB memiliki blok CIDR Internet Protocol versi 6 (IPv6) yang terkait dengannya. Untuk informasi selengkapnya, lihat Penentuan alamat IP Amazon Aurora dan Memigrasi ke IPv6 dalam Panduan Pengguna Amazon VPC.

Saat Amazon Aurora membuat klaster DB dalam VPC, antarmuka jaringan ditetapkan ke klaster DB dengan menggunakan alamat IP dari grup subnet DB. Namun, kami sangat menyarankan agar Anda menggunakan nama Sistem Nama Domain (DNS) untuk terhubung ke klaster DB. Kami merekomendasikan hal ini karena alamat IP pokok berubah selama failover.

> **Note**
>
> Untuk setiap klaster DB yang Anda jalankan dalam VPC, pastikan untuk mencadangkan minimal satu alamat di setiap subnet dalam grup subnet DB agar dapat digunakan oleh Amazon Aurora untuk tindakan pemulihan.

### Subnet bersama

Anda dapat membuat klaster DB dalam VPC bersama.

Beberapa pertimbangan yang perlu diingat saat menggunakan shared VPCs:

- Anda dapat memindahkan klaster DB dari subnet VPC bersama ke subnet VPC privat dan sebaliknya.
- Peserta dalam VPC bersama harus membuat grup keamanan dalam VPC agar dapat membuat klaster DB.
- Pemilik dan peserta dalam VPC bersama dapat mengakses basis data dengan menggunakan kueri SQL. Namun, hanya pembuat sumber daya yang dapat melakukan panggilan API apa pun pada sumber daya.

## Penentuan alamat IP Amazon Aurora

Penentuan alamat IP memungkinkan sumber daya dalam VPC Anda berkomunikasi satu sama lain, dan dengan sumber daya melalui internet. Amazon Aurora mendukung protokol pengalamatan

IPv6 dan IPv4 keduanya. Secara default, Amazon Aurora dan Amazon VPC menggunakan protokol pengalamatan. IPv4 Anda tidak dapat menonaktifkan perilaku ini. Saat Anda membuat VPC, pastikan untuk menentukan blok IPv4 CIDR (berbagai alamat pribadi IPv4 ).

Support untuk IPv6 protokol memperluas jumlah alamat IP yang didukung. Dengan menggunakan IPv6 protokol, Anda memastikan bahwa Anda memiliki alamat yang cukup tersedia untuk pertumbuhan internet di masa depan. Sumber daya RDS baru dan yang sudah ada dapat digunakan IPv4 dan IPv6 alamat dalam VPC Anda. Mengonfigurasi, mengamankan, dan menerjemahkan lalu lintas jaringan di antara dua protokol yang digunakan di berbagai bagian aplikasi dapat menimbulkan overhead operasional. Anda dapat melakukan standarisasi pada IPv6 protokol untuk sumber daya Amazon RDS untuk menyederhanakan konfigurasi jaringan Anda.

Topik

- IPv4 alamat
- IPv6 alamat
- Mode tumpukan ganda

IPv4 alamat

Saat Anda membuat VPC, Anda harus menentukan rentang IPv4 alamat untuk VPC dalam bentuk blok CIDR, seperti. `10.0.0.0/16` Grup subnet DB mendefinisikan rentang alamat IP di blok CIDR ini yang dapat digunakan oleh klaster DB. Alamat IP ini bisa bersifat privat atau publik.

IPv4 Alamat pribadi adalah alamat IP yang tidak dapat dijangkau melalui internet. Anda dapat menggunakan IPv4 alamat pribadi untuk komunikasi antara cluster DB dan sumber daya lainnya, seperti EC2 instans Amazon, di VPC yang sama. Setiap klaster DB memiliki alamat IP privat untuk komunikasi dalam VPC.

Alamat IP publik adalah IPv4 alamat yang dapat dijangkau dari internet. Anda dapat menggunakan alamat publik untuk komunikasi antara klaster DB dan sumber daya di internet, seperti klien SQL. Anda mengontrol apakah klaster DB Anda menerima alamat IP publik.

Untuk tutorial yang menunjukkan cara membuat VPC hanya dengan IPv4 alamat pribadi yang dapat Anda gunakan untuk skenario Amazon , lihat. Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4

### IPv6 alamat

Anda dapat secara opsional mengaitkan blok IPv6 CIDR dengan VPC dan subnet Anda, dan menetapkan IPv6 alamat dari blok itu ke sumber daya di VPC Anda. Setiap IPv6 alamat unik secara global.

Blok IPv6 CIDR untuk VPC Anda secara otomatis ditetapkan dari kumpulan IPv6 alamat Amazon. Anda tidak dapat memilih sendiri rentang tersebut.

Saat menghubungkan ke IPv6 alamat, pastikan bahwa kondisi berikut terpenuhi:

- Klien dikonfigurasi sehingga lalu lintas klien ke database IPv6 diizinkan.
- Grup keamanan RDS yang digunakan oleh instans DB dikonfigurasi dengan benar sehingga lalu lintas klien ke database IPv6 diizinkan.
- Tumpukan sistem operasi klien memungkinkan lalu lintas pada IPv6 alamat, dan driver dan pustaka sistem operasi dikonfigurasi untuk memilih titik akhir instans DB default yang benar (baik IPv4 atau IPv6).

Untuk informasi selengkapnya IPv6, lihat Pengalamatan IP di Panduan Pengguna Amazon VPC.

### Mode tumpukan ganda

Ketika cluster DB dapat berkomunikasi melalui protokol IPv6 pengalamatan IPv4 dan pengalamatan, itu berjalan dalam mode dual-stack. Jadi, sumber daya dapat berkomunikasi dengan cluster DB over IPv4, IPv6, atau keduanya. RDS menonaktifkan akses Internet Gateway untuk IPv6 titik akhir instans DB mode dual-stack pribadi. RDS melakukan ini untuk memastikan bahwa IPv6 titik akhir Anda bersifat pribadi dan hanya dapat diakses dari dalam VPC Anda.

Topik

- Mode tumpukan ganda dan grup subnet DB
- Menggunakan instans DB mode tumpukan ganda
- Memodifikasi cluster DB IPv4 -only untuk menggunakan mode dual-stack
- Ketersediaan klaster DB jaringan tumpukan ganda
- Batasan untuk klaster DB jaringan tumpukan ganda

Untuk tutorial yang menunjukkan cara membuat VPC dengan keduanya IPv4 dan IPv6 alamat yang dapat Anda gunakan untuk skenario Amazon yang umum, lihat. Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

Mode tumpukan ganda dan grup subnet DB

Untuk menggunakan mode dual-stack, pastikan bahwa setiap subnet dalam grup subnet DB yang Anda kaitkan dengan cluster DB memiliki blok IPv6 CIDR yang terkait dengannya. Anda dapat membuat grup subnet DB baru atau memodifikasi grup subnet DB yang ada untuk memenuhi persyaratan ini. Setelah klaster DB berada dalam mode tumpukan ganda, klien dapat terhubung secara normal. Pastikan firewall keamanan klien dan grup keamanan instans RDS DB dikonfigurasi secara akurat untuk memungkinkan lalu lintas berakhir. IPv6 Untuk terhubung, klien menggunakan titik akhir instans primer klaster DB. Aplikasi klien dapat menentukan protokol mana yang lebih disukai saat terhubung ke basis data. Dalam mode dual-stack, cluster DB mendeteksi protokol jaringan pilihan klien, baik IPv4 atau IPv6, dan menggunakan protokol itu untuk koneksi.

Jika grup subnet DB berhenti mendukung mode tumpukan ganda karena penghapusan subnet atau pembatalan pengaitan CIDR, ada risiko status jaringan menjadi tidak kompatibel untuk instans DB yang terkait dengan grup subnet DB. Selain itu, Anda tidak dapat menggunakan grup subnet DB saat membuat klaster DB mode tumpukan ganda baru.

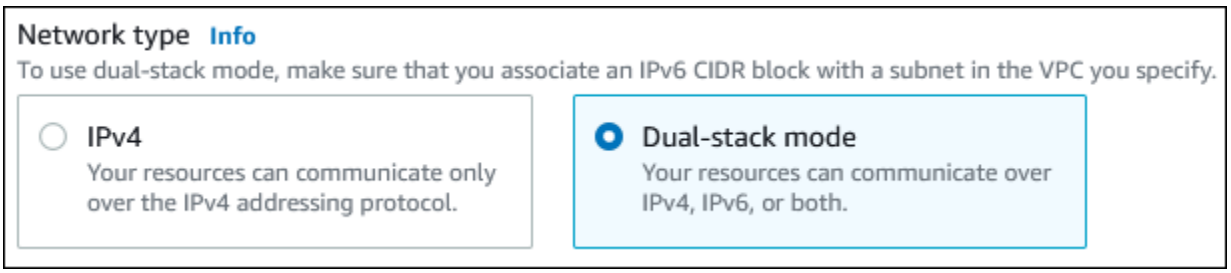
Untuk menentukan apakah grup subnet DB mendukung mode dual-stack dengan menggunakan AWS Management Console, lihat tipe Jaringan pada halaman detail grup subnet DB. Untuk menentukan apakah grup subnet DB mendukung mode dual-stack dengan menggunakan AWS CLI, jalankan describe-db-subnet-groupsperintah dan lihat `SupportedNetworkTypes` di output.

Replika baca dianggap sebagai instans DB independen dan dapat memiliki jenis jaringan yang berbeda dengan instans DB primer. Jika Anda mengubah jenis jaringan instans DB primer replika baca, replika baca ini tidak akan terpengaruh. Saat memulihkan instans DB, Anda dapat memulihkannya ke jenis jaringan apa pun yang didukung.

Menggunakan instans DB mode tumpukan ganda

Saat Anda membuat atau memodifikasi cluster DB, Anda dapat menentukan mode dual-stack untuk memungkinkan sumber daya Anda berkomunikasi dengan cluster DB Anda melalui IPv4, IPv6, atau keduanya.

Bila Anda menggunakan AWS Management Console untuk membuat atau memodifikasi instans DB, Anda dapat menentukan mode dual-stack di bagian Jenis jaringan. Gambar berikut menampilkan bagian Jenis jaringan di konsol.




Saat Anda menggunakan AWS CLI untuk membuat atau memodifikasi cluster DB, atur `--network-type` opsi `DUAL` untuk menggunakan mode dual-stack. Saat Anda menggunakan API RDS untuk membuat atau memodifikasi klaster DB, atur parameter `NetworkType` ke `DUAL` untuk menggunakan mode tumpukan ganda. Saat Anda memodifikasi jenis jaringan instans DB, waktu henti mungkin akan terjadi. Jika mode tumpukan ganda tidak didukung oleh versi mesin DB yang ditentukan atau grup subnet DB, kesalahan `NetworkTypeNotSupported` akan ditampilkan.

Untuk informasi selengkapnya tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang cara mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Untuk menentukan apakah klaster DB berada dalam mode tumpukan ganda dengan menggunakan konsol, lihat Jenis jaringan pada tab Konektivitas & keamanan untuk klaster DB.

Memodifikasi cluster DB IPv4 -only untuk menggunakan mode dual-stack

Anda dapat memodifikasi cluster DB IPv4 -only untuk menggunakan mode dual-stack. Untuk melakukannya, ubah jenis jaringan klaster DB. Modifikasi dapat mengakibatkan waktu henti.

Sebaiknya Anda mengubah jenis jaringan klaster DB Amazon Aurora selama periode pemeliharaan. Saat ini, pengaturan jenis jaringan instans baru ke mode tumpukan ganda tidak didukung. Anda dapat mengatur jenis jaringan secara manual dengan menggunakan perintah `modify-db-cluster`.

Sebelum memodifikasi klaster DB untuk menggunakan mode tumpukan ganda, pastikan grup subnet DB-nya mendukung mode tumpukan ganda. Jika grup subnet DB yang terkait dengan klaster DB tidak mendukung mode tumpukan ganda, tentukan grup subnet DB yang berbeda yang mendukungnya saat Anda memodifikasi klaster DB. Modifikasi terhadap grup subnet DB dari klaster DB dapat menyebabkan waktu henti.

Jika Anda memodifikasi grup subnet DB dari klaster DB sebelum mengubah klaster DB untuk menggunakan mode tumpukan ganda, pastikan grup subnet DB tersebut valid untuk klaster DB sebelum dan sesudah perubahan.

Kami menyarankan Anda menjalankan modify-db-clusterAPI hanya dengan `--network-type` parameter dengan nilai `DUAL` untuk mengubah jaringan cluster Amazon Aurora ke mode dual-stack. Penambahan parameter lain bersama dengan parameter `--network-type` dalam panggilan API yang sama dapat mengakibatkan waktu henti.

Jika Anda tidak dapat terhubung ke cluster DB setelah perubahan, pastikan firewall keamanan klien dan database serta tabel rute dikonfigurasi secara akurat untuk memungkinkan lalu lintas ke database pada jaringan yang dipilih (baik IPv4 atau IPv6). Anda mungkin juga perlu memodifikasi parameter sistem operasi, pustaka, atau driver untuk terhubung menggunakan IPv6 alamat.

Untuk memodifikasi cluster DB IPv4 -only agar menggunakan mode dual-stack

1. Ubah grup subnet DB untuk mendukung mode tumpukan ganda, atau buat grup subnet DB yang mendukung mode tumpukan ganda:

   a. Kaitkan blok IPv6 CIDR dengan VPC Anda.

      Untuk petunjuknya, lihat Menambahkan blok IPv6 CIDR ke VPC Anda di Panduan Pengguna Amazon VPC.

   b. Pasang blok IPv6 CIDR ke semua subnet di grup subnet DB Anda.

      Untuk petunjuk, lihat Menambahkan blok IPv6 CIDR ke subnet Anda di Panduan Pengguna Amazon VPC.

   c. Konfirmasikan bahwa grup subnet DB mendukung mode tumpukan ganda.

      Jika Anda menggunakan AWS Management Console, pilih grup subnet DB, dan pastikan bahwa nilai tipe jaringan yang didukung adalah Dual, IPv4.

      Jika Anda menggunakan AWS CLI, jalankan describe-db-subnet-groupsperintah, dan pastikan bahwa `SupportedNetworkType` nilai untuk instans DB adalah`Dual, IPv4`.

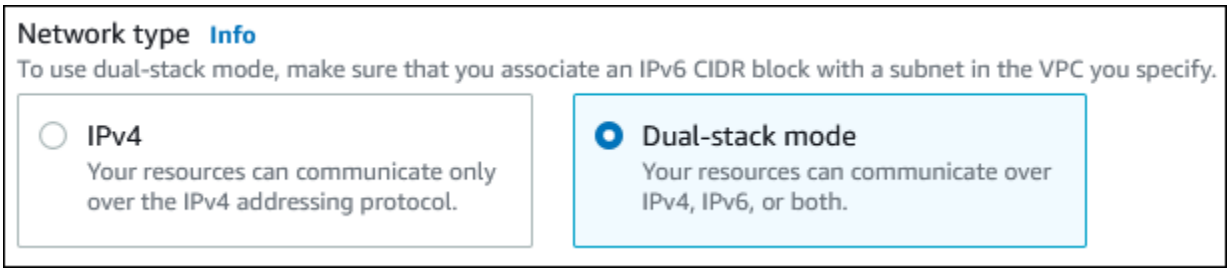
2. Ubah grup keamanan yang terkait dengan cluster DB untuk memungkinkan IPv6 koneksi ke database, atau buat grup keamanan baru yang memungkinkan IPv6 koneksi.

   Untuk mendapatkan petunjuk, lihat Aturan grup keamanan dalam Panduan Pengguna Amazon VPC.

3. Modifikasi klaster DB untuk mendukung mode tumpukan ganda. Untuk melakukannya, tetapkan Jenis jaringan ke Mode tumpukan ganda.

   Jika Anda menggunakan konsol, pastikan bahwa pengaturan berikut sudah benar:

- Jenis jaringan – Mode tumpukan ganda




- Grup subnet DB – Grup subnet DB yang telah Anda konfigurasikan pada langkah sebelumnya
- Grup keamanan — Grup keamanan yang telah Anda konfigurasikan pada langkah sebelumnya

Jika Anda menggunakan AWS CLI, pastikan bahwa pengaturan berikut sudah benar:

- `--network-type` – `dual`
- `--db-subnet-group-name` – Grup subnet DB yang Anda konfigurasikan pada langkah sebelumnya
- `--vpc-security-group-ids` – Grup keamanan VPC yang Anda konfigurasikan pada langkah sebelumnya

Sebagai contoh:

```
aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
```

4. Konfirmasikan bahwa klaster DB mendukung mode tumpukan ganda.

   Jika Anda menggunakan konsol, pilih tab Konfigurasi untuk klaster DB. Pada tab tersebut, pastikan nilai Jenis jaringan adalah Mode tumpukan ganda.

   Jika Anda menggunakan AWS CLI, jalankan describe-db-clustersperintah, dan pastikan bahwa `NetworkType` nilai untuk cluster DB adalah`dual`.

   Jalankan `dig` perintah pada titik akhir instance DB penulis untuk mengidentifikasi IPv6 alamat yang terkait dengannya.

```
dig db-instance-endpoint AAAA
```

Gunakan titik akhir instans DB penulis, bukan IPv6 alamatnya, untuk terhubung ke cluster DB.

### Ketersediaan klaster DB jaringan tumpukan ganda

Cluster DB jaringan dual-stack tersedia di semua Wilayah AWS kecuali untuk yang berikut:

- Asia Pasifik (Hyderabad)
- Asia Pasifik (Malaysia)
- Asia Pasifik (Melbourne)
- Kanada Barat (Calgary)
- Eropa (Spanyol)
- Eropa (Zürich)
- Israel (Tel Aviv)
- Timur Tengah (UEA)

Versi mesin DB berikut mendukung cluster DB jaringan dual-stack:

- Versi Aurora MySQL:
  - 3.02 dan versi 3 yang lebih tinggi
  - 2.09.1 dan versi 2 yang lebih tinggi

  Untuk informasi selengkapnya tentang versi Aurora MySQL, lihat Catatan Rilis untuk Aurora MySQL.
- Versi Aurora PostgreSQL:
  - 15.2 dan semua versi yang lebih tinggi
  - 14.3 dan versi 14 yang lebih tinggi
  - 13.7 dan versi 13 yang lebih tinggi

  Untuk informasi selengkapnya tentang versi Aurora PostgreSQL, lihat Catatan Rilis untuk Aurora PostgreSQL.

### Batasan untuk klaster DB jaringan tumpukan ganda

Batasan berikut berlaku untuk klaster DB jaringan tumpukan ganda:

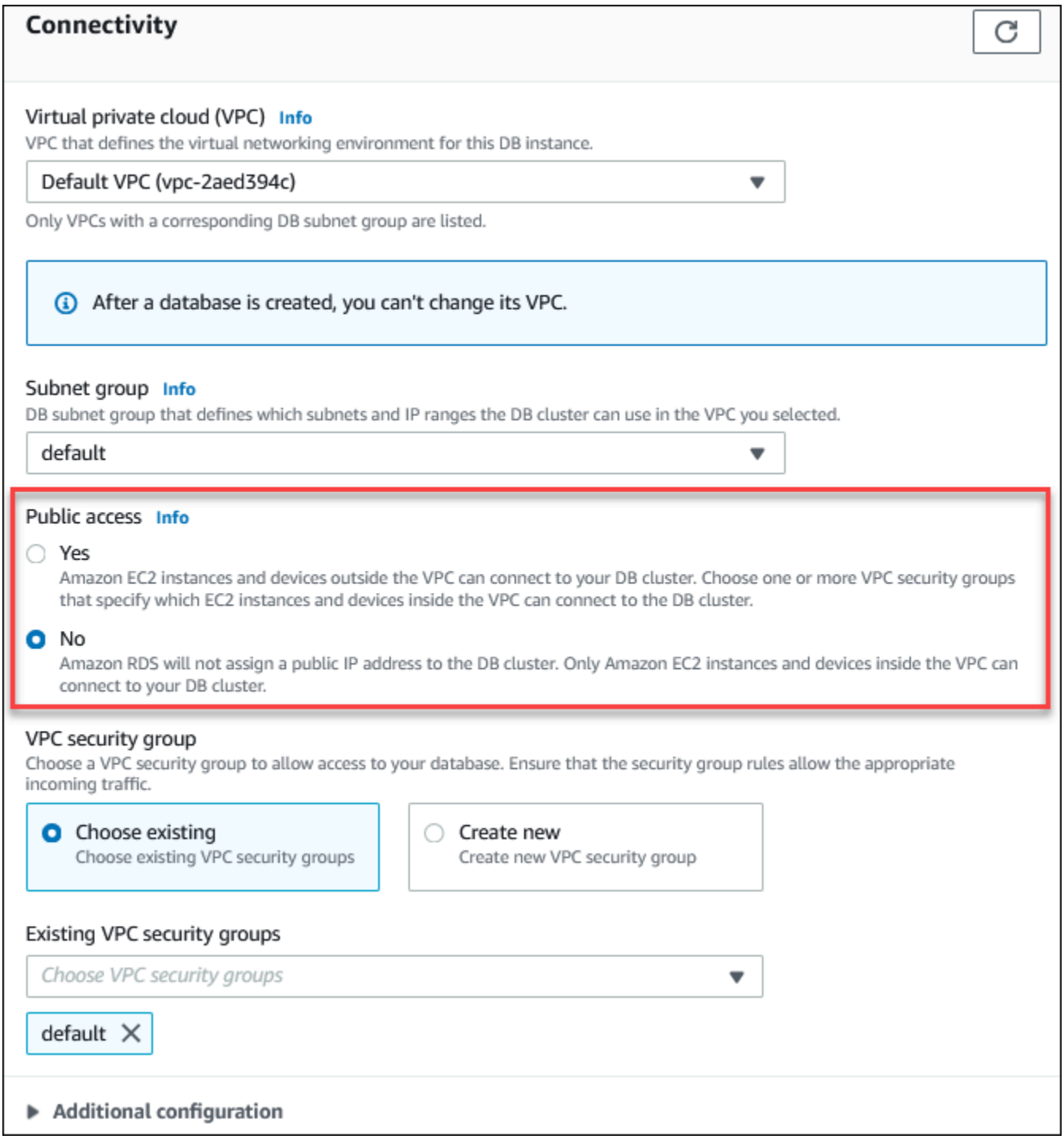
- Cluster DB tidak dapat menggunakan IPv6 protokol secara eksklusif. Mereka dapat menggunakan IPv4 secara eksklusif, atau mereka dapat menggunakan IPv6 protokol IPv4 and (mode dual-stack).
- Amazon RDS tidak mendukung IPv6 subnet asli.
- Klaster DB yang menggunakan mode tumpukan ganda harus bersifat privat. Hal ini tidak boleh dapat diakses oleh publik.
- Anda tidak dapat menggunakan Proksi RDS dengan klaster DB mode tumpukan ganda.

## Menyembunyikan klaster DB dalam VPC dari internet

Salah satu skenario Amazon Aurora yang umum adalah memiliki VPC di mana Anda memiliki instans EC2 Amazon dengan aplikasi web yang menghadap publik dan cluster instans dengan database yang tidak dapat diakses publik. Misalnya, Anda dapat membuat VPC yang memiliki subnet publik dan subnet pribadi. EC2 instance yang berfungsi sebagai server web dapat digunakan di subnet publik. Klaster DB di-deploy di subnet privat. Dalam deployment tersebut, hanya server web yang memiliki akses ke klaster DB. Untuk ilustrasi skenario ini, lihat Kluster DB dalam instans yang VPC diakses oleh EC2 instans Amazon dalam hal yang sama VPC.

Saat Anda meluncurkan klaster DB di dalam VPC, klaster DB memiliki alamat IP privat untuk lalu lintas di dalam VPC. Alamat IP privat ini tidak dapat diakses publik. Anda dapat menggunakan opsi Akses publik untuk menetapkan apakah klaster DB juga memiliki alamat IP publik selain alamat IP privat. Jika klaster DB ditetapkan sebagai dapat diakses publik, titik akhir DNS-nya akan diresolusi ke alamat IP privat dari dalam VPC. Titik akhir tersebut akan diresolusi ke alamat IP publik dari luar VPC. Akses ke klaster DB pada dasarnya dikendalikan oleh grup keamanan yang menggunakannya. Akses publik tersebut tidak diizinkan jika grup keamanan yang ditetapkan ke klaster DB tidak menyertakan aturan masuk yang mengizinkannya. Selain itu, agar klaster DB dapat diakses publik, subnet dalam grup subnet DB-nya harus memiliki gateway internet. Untuk informasi selengkapnya, lihat Tidak dapat terhubung ke instans DB Amazon RDS

Anda dapat memodifikasi klaster DB untuk mengaktifkan atau menonaktifkan aksesibilitas publik dengan memodifikasi opsi Akses publik. Ilustrasi berikut ini menunjukkan opsi Akses publik di bagian Konfigurasi konektivitas tambahan. Untuk mengatur opsi, buka bagian Konfigurasi konektivitas tambahan di bagian Konektivitas.




Untuk informasi tentang cara memodifikasi instans DB untuk menetapkan opsi Akses publik, lihat Memodifikasi instans DB dalam klaster DB.

## Membuat klaster DB dalam VPC

Prosedur berikut membantu Anda membuat klaster DB dalam VPC. Untuk menggunakan VPC default, Anda dapat memulai dengan langkah 2, dan menggunakan VPC dan grup subnet DB yang sudah dibuat untuk Anda. Jika Anda ingin membuat VPC tambahan, Anda dapat membuat VPC baru.

> **Note**
>
> Jika Anda ingin klaster DB dalam VPC dapat diakses publik, Anda harus memperbarui informasi DNS untuk VPC dengan mengaktifkan nama host DNS dan resolusi DNS atribut VPC. Untuk informasi tentang cara memperbarui informasi DNS untuk instans VPC, lihat Memperbarui dukungan DNS untuk VPC Anda.

Ikuti langkah-langkah ini untuk membuat instans DB dalam VPC:

- Langkah 1: Buat VPC
- Langkah 2: Buat grup subnet DB
- Langkah 3: Buat grup keamanan VPC
- Langkah 4: Buat instans DB dalam VPC

### Langkah 1: Buat VPC

Buat VPC dengan subnet di setidaknya dua Zona Ketersediaan. Anda menggunakan subnet ini ketika membuat grup subnet DB. Jika Anda memiliki VPC default, subnet secara otomatis dibuat untuk Anda dalam setiap Zona Ketersediaan di Wilayah AWS.

Untuk informasi selengkapnya, lihat Membuat VPC dengan subnet publik dan privat, atau lihat Membuat VPC dalam Panduan Pengguna Amazon VPC.

### Langkah 2: Buat grup subnet DB

Grup subnet DB adalah kumpulan subnet (biasanya privat) yang Anda buat untuk VPC lalu ditetapkan untuk klaster DB Anda. Grup subnet DB memungkinkan Anda menentukan VPC tertentu saat Anda membuat cluster instans menggunakan AWS CLI API atau RDS. Jika menggunakan konsol, Anda dapat memilih VPC dan subnet yang ingin Anda gunakan. Setiap subnet DB harus memiliki setidaknya satu subnet di setidaknya dua Zona Ketersediaan dalam Wilayah AWS. Sebagai

praktik terbaik, setiap grup subnet DB harus memiliki minimal satu subnet untuk setiap Zona Ketersediaan di Wilayah AWS.

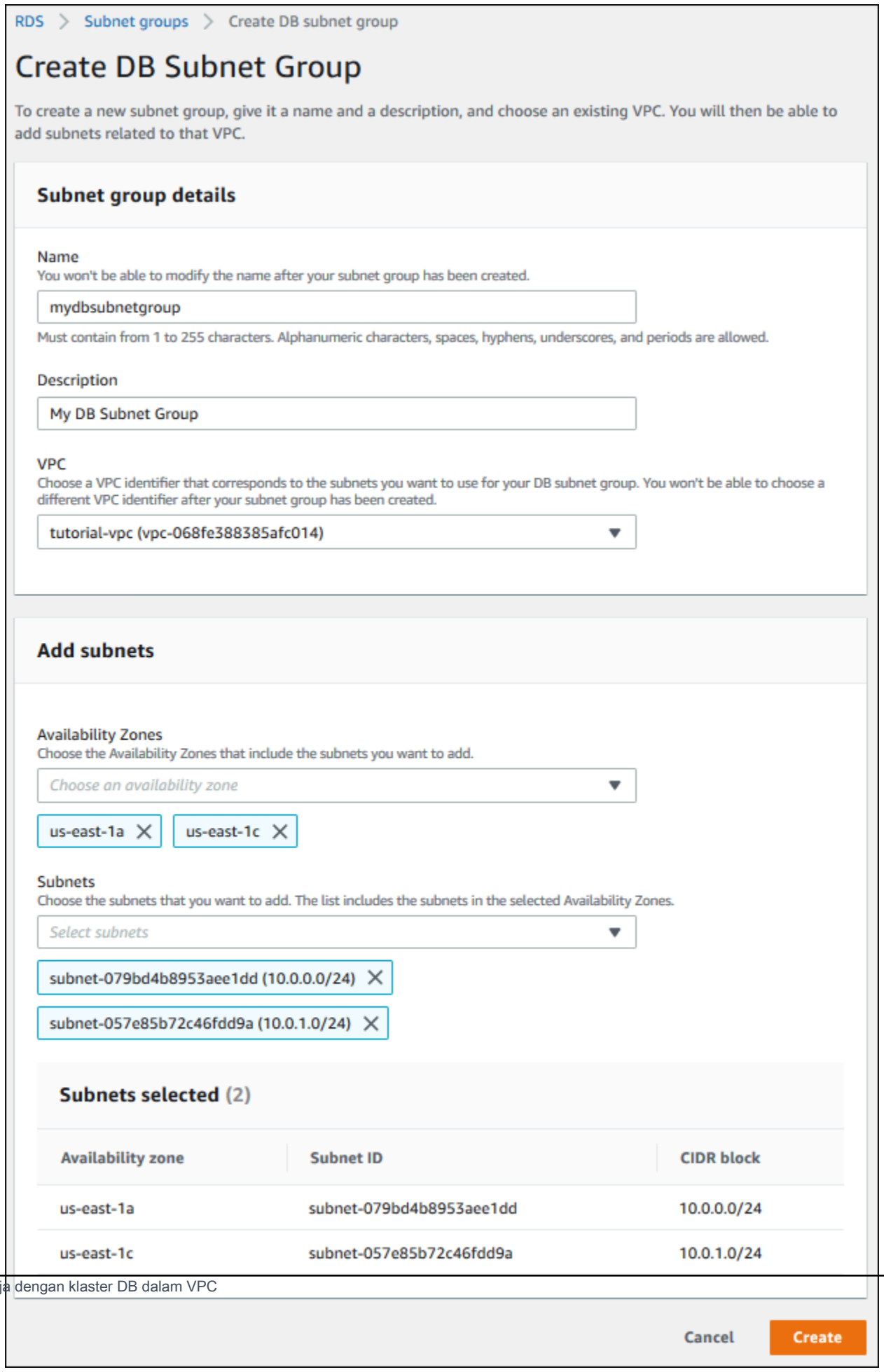
Agar klaster DB dapat diakses publik, subnet dalam grup subnet DB harus memiliki gateway internet. Untuk informasi selengkapnya tentang gateway internet untuk subnet, lihat Terhubung ke internet menggunakan gateway internet dalam Panduan Pengguna Amazon VPC.

Saat Anda membuat klaster DB dalam VPC, Anda dapat memilih grup subnet DB. Amazon Aurora memilih subnet dan alamat IP dalam subnet tersebut untuk dikaitkan dengan klaster DB Anda. Jika tidak ada grup subnet DB, Amazon Aurora membuat grup subnet default saat Anda membuat klaster DB. Amazon Aurora membuat dan mengaitkan Antarmuka Jaringan Elastis ke klaster DB Anda dengan alamat IP tersebut. Klaster DB menggunakan Zona Ketersediaan yang berisi subnet.

Pada langkah ini, Anda membuat grup subnet DB dan menambahkan subnet yang Anda buat untuk VPC Anda.

Untuk membuat grup subnet DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup subnet.
3. Pilih Buat Grup Subnet DB.
4. Untuk Nama, ketik nama grup subnet DB Anda.
5. Untuk Deskripsi, ketik deskripsi untuk grup subnet DB Anda.
6. Untuk VPC, pilih VPC default atau VPC yang Anda buat.
7. Di bagian Tambahkan subnet, pilih Zona Ketersediaan yang mencakup subnet dari Zona Ketersediaan, lalu pilih subnet dari Subnet.


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. Pilih Buat.

   Grup subnet DB baru Anda muncul dalam daftar grup subnet DB di konsol RDS. Anda dapat memilih grup subnet DB untuk melihat detail, termasuk semua subnet yang dikaitkan dengan grup, dalam panel detail di bagian bawah jendela.

Langkah 3: Buat grup keamanan VPC

Sebelum membuat klaster DB, Anda dapat membuat grup keamanan VPC untuk dikaitkan dengan klaster DB. Jika Anda tidak membuat grup keamanan VPC, Anda dapat menggunakan grup keamanan default saat membuat klaster DB. Untuk petunjuk cara membuat grup keamanan untuk klaster DB, lihat Membuat grup keamanan VPC untuk klaster DB privat, atau lihat Mengontrol lalu lintas ke sumber daya menggunakan grup keamanan dalam Panduan Pengguna Amazon VPC.

Langkah 4: Buat instans DB dalam VPC

Pada langkah ini, Anda membuat klaster DB dan menggunakan nama VPC, grup subnet DB, dan grup keamanan VPC yang telah Anda buat pada langkah sebelumnya.

> **Note**
>
> Jika ingin klaster DB dalam VPC dapat diakses publik, Anda harus mengaktifkan Nama host DNS dan Resolusi DNS atribut VPC. Untuk informasi selengkapnya, lihat Atribut DNS untuk VPC Anda dalam Panduan Pengguna Amazon VPC.

Untuk detail tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora.

Saat diminta di bagian Konektivitas, masukkan nama VPC, grup subnet DB, dan grup keamanan VPC.

> **Note**
>
> Pembaruan saat ini VPCs tidak didukung untuk klaster Aurora DB.

## Skenario untuk mengakses cluster DB di VPC

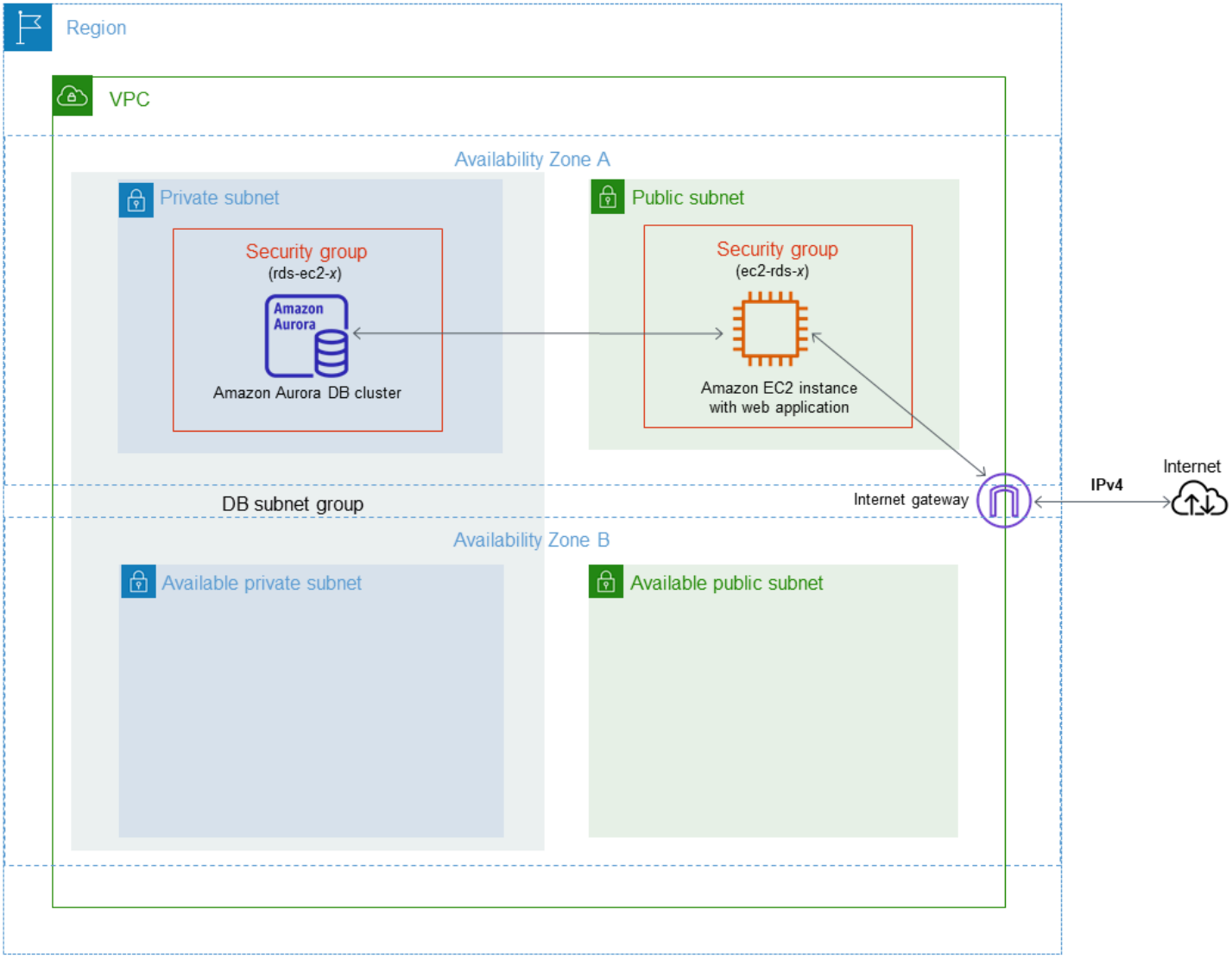
Amazon Aurora mendukung skenario berikut untuk mengakses cluster DB di: VPC

- EC2Contoh Amazon dalam hal yang sama VPC
- Sebuah EC2 contoh dalam yang berbeda VPC
- Aplikasi klien melalui internet
- Jaringan privat

## Kluster DB dalam instans yang VPC diakses oleh EC2 instans Amazon dalam hal yang sama VPC

Penggunaan umum cluster DB di a VPC adalah untuk berbagi data dengan server aplikasi yang berjalan dalam EC2 instance Amazon dalam hal yang samaVPC.

Diagram berikut menunjukkan skenario ini.



Cara termudah untuk mengelola akses antara EC2 instans dan cluster instans DB dalam yang sama VPC adalah dengan melakukan hal berikut:

- Buat grup VPC keamanan untuk kluster DB Anda. Grup keamanan ini dapat digunakan untuk membatasi akses ke klaster DB. Misalnya, Anda dapat membuat aturan kustom untuk grup keamanan ini. Ini mungkin memungkinkan TCP akses menggunakan port yang Anda tetapkan ke cluster DB saat Anda membuatnya dan alamat IP yang Anda gunakan untuk mengakses cluster DB untuk pengembangan atau tujuan lain.
- Buat grup VPC keamanan untuk EC2 instans Anda (server web dan klien) untuk masuk. Grup keamanan ini dapat, jika diperlukan, mengizinkan akses ke EC2 instance dari internet dengan menggunakan tabel VPC routing. Misalnya, Anda dapat menetapkan aturan pada grup keamanan ini untuk mengizinkan TCP akses ke EC2 instance melalui port 22.
- Buat aturan khusus dalam grup keamanan untuk cluster DB yang memungkinkan koneksi dari grup keamanan yang Anda buat untuk instans AndaEC2. Aturan ini dapat memungkinkan anggota grup keamanan untuk mengakses instans DB.

Ada subnet publik dan privat tambahan di Zona Ketersediaan terpisah. Grup subnet RDS DB memerlukan subnet di setidaknya dua Availability Zone. Subnet tambahan memudahkan Anda untuk beralih ke deployment instans DB Multi-AZ di masa mendatang.

Untuk tutorial yang menunjukkan cara membuat VPC subnet publik dan pribadi untuk skenario ini, lihatTutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4 .

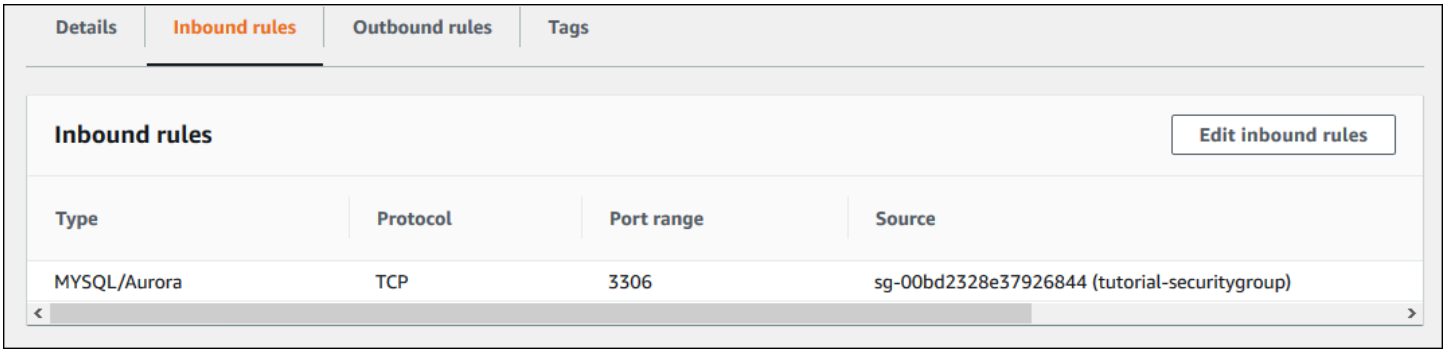
> **Tip**
>
> Anda dapat mengatur konektivitas jaringan antara instans Amazon dan cluster EC2 DB secara otomatis saat membuat cluster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance .

Untuk membuat aturan dalam grup VPC keamanan yang memungkinkan koneksi dari grup keamanan lain, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka VPC konsol Amazon di https://console.aws.amazon.com/vpc.
2. Di panel navigasi, pilih Grup keamanan.

3. Pilih atau buat grup keamanan yang ingin Anda berikan aksesnya ke anggota grup keamanan lain. Dalam skenario sebelumnya, ini adalah grup keamanan yang Anda gunakan untuk klaster DB Anda. Pilih tab Aturan masuk, lalu pilih Edit aturan masuk.
4. Di halaman Edit aturan masuk, pilih Tambahkan aturan.
5. Untuk Type, pilih entri yang sesuai dengan port yang Anda gunakan saat membuat cluster DB, seperti MYSQL/Aurora.
6. Di kotak Sumber, mulai ketikkan ID dari grup keamanan, sehingga akan menampilkan daftar grup keamanan yang sesuai. Pilih grup keamanan dengan anggota yang ingin diberi akses ke sumber daya yang dilindungi oleh grup keamanan ini. Dalam skenario sebelumnya, ini adalah grup keamanan yang Anda gunakan untuk instance AndaEC2.
7. Jika diperlukan, ulangi langkah-langkah untuk TCP protokol dengan membuat aturan dengan Semua TCP sebagai Jenis dan grup keamanan Anda di kotak Sumber. Jika Anda ingin menggunakan UDP protokol, buat aturan dengan All UDP as the Type dan grup keamanan Anda di Source.
8. Pilih Simpan aturan.

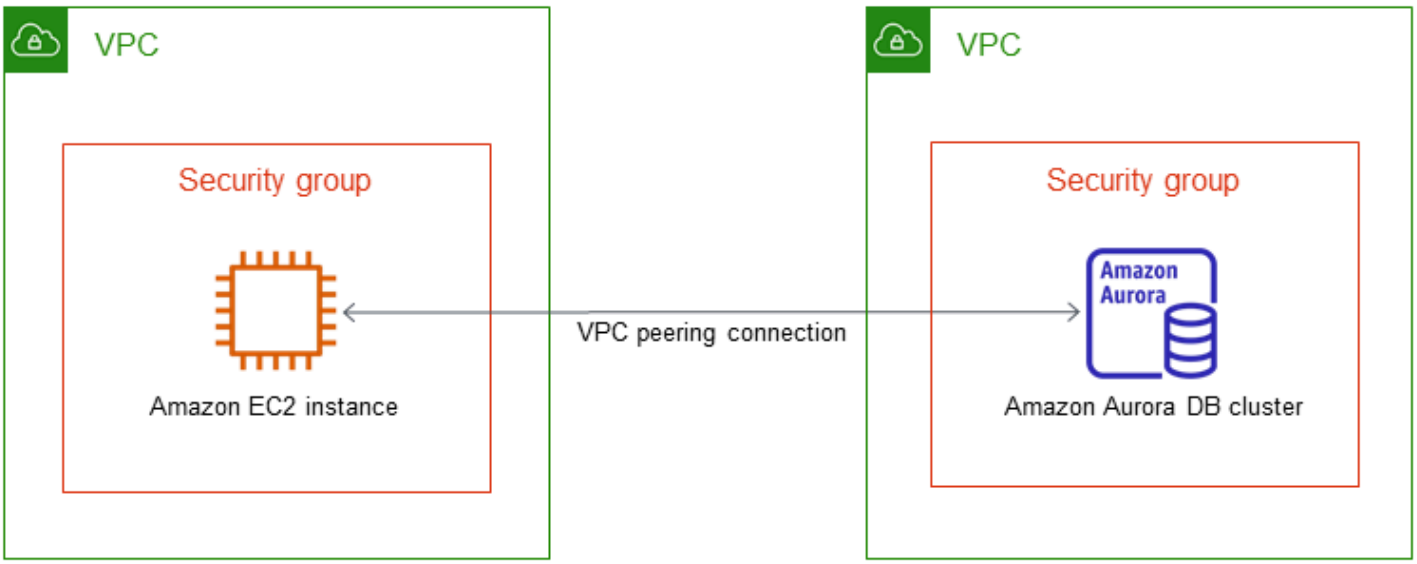
Layar berikut menunjukkan aturan masuk dengan grup keamanan sebagai sumbernya.




Untuk informasi selengkapnya tentang menghubungkan ke cluster DB dari EC2 instans Anda, lihat .

## Cluster DB dalam VPC diakses oleh sebuah EC2 instance di tempat yang berbeda VPC

Diagram berikut menunjukkan skenario ini.

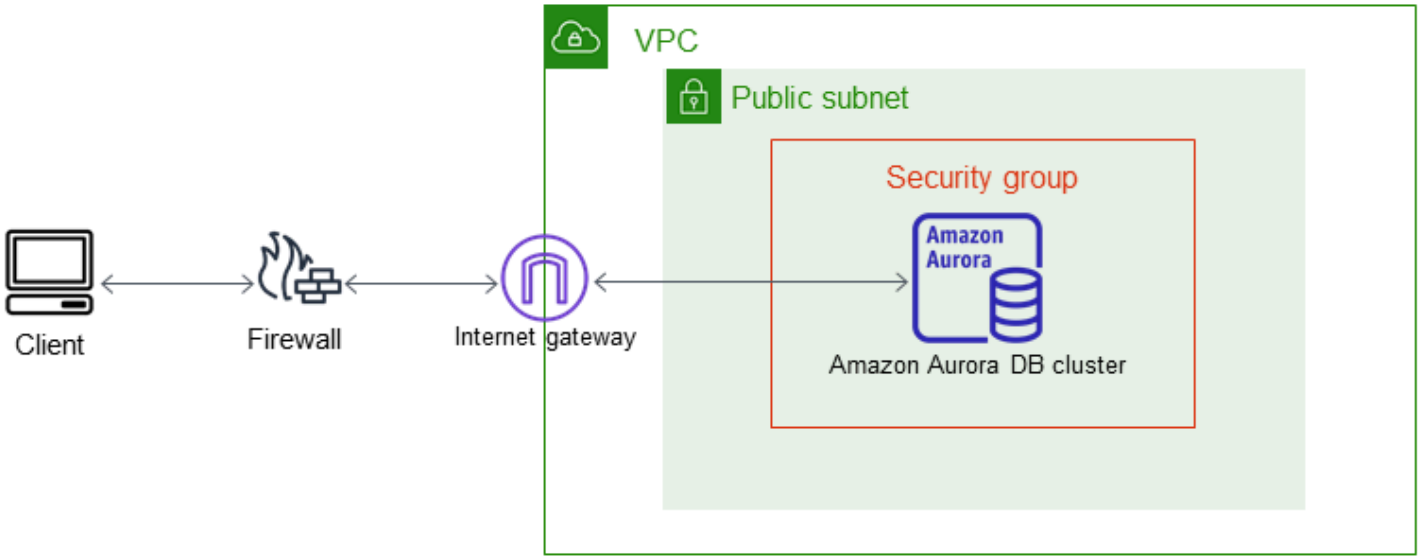



Koneksi VPC peering adalah koneksi jaringan antara dua VPCs yang memungkinkan Anda untuk merutekan lalu lintas di antara mereka menggunakan alamat IP pribadi. Sumber daya di keduanya VPC dapat berkomunikasi satu sama lain seolah-olah mereka berada dalam jaringan yang sama. Anda dapat membuat koneksi VPC peering antara Anda sendiriVPCs, dengan VPC AWS akun lain, atau dengan VPC di akun lain Wilayah AWS. Untuk mempelajari lebih lanjut tentang VPC mengintip, lihat VPCmengintip di Panduan Pengguna Amazon Virtual Private Cloud.

## Sebuah cluster DB dalam VPC diakses oleh aplikasi klien melalui internet

Untuk mengakses cluster DB dalam aplikasi VPC dari klien melalui internet, Anda mengonfigurasi subnet publik VPC dengan satu subnet publik, dan gateway internet untuk mengaktifkan komunikasi melalui internet.

Diagram berikut menunjukkan skenario ini.




Kami merekomendasikan konfigurasi berikut:

- A VPC ukuran /16 (misalnyaCIDR: 10.0.0.0/16). Ukuran ini menyediakan 65.536 alamat IP privat.
- Subnet ukuran /24 (misalnyaCIDR: 10.0.0.0/24). Ukuran ini menyediakan 256 alamat IP privat.
- Cluster Aurora DB Amazon yang terkait dengan VPC dan subnet. Amazon RDS menetapkan alamat IP dalam subnet ke cluster DB Anda.
- Sebuah gateway internet yang menghubungkan VPC ke internet dan ke AWS produk lainnya.
- Grup keamanan yang terkait dengan klaster DB. Aturan masuk grup keamanan memungkinkan aplikasi klien Anda mengakses klaster DB Anda.

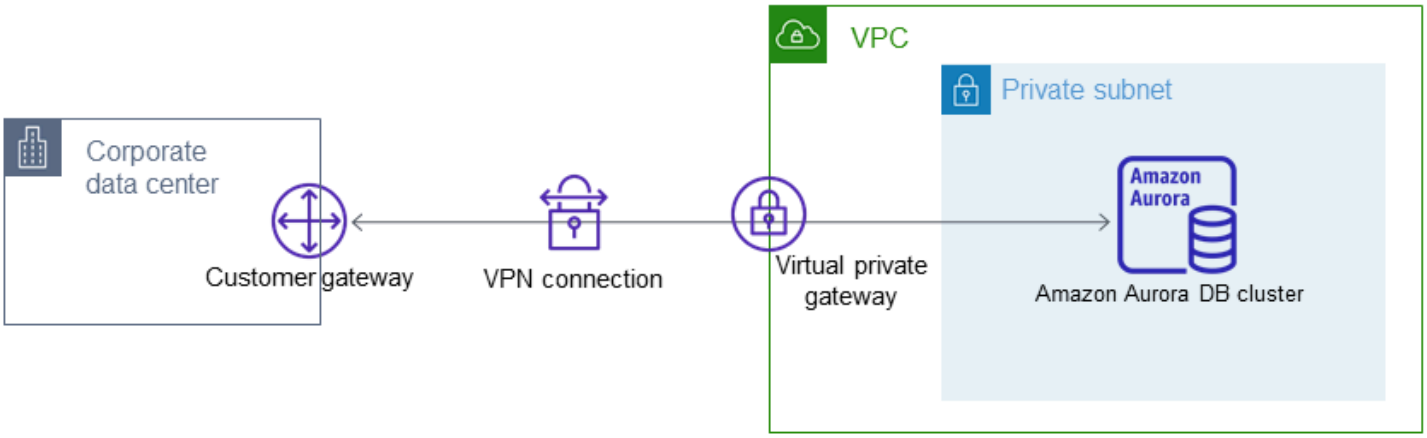
Untuk informasi tentang membuat cluster DB di aVPC, lihat. Membuat klaster DB dalam VPC

## Sebuah cluster DB dalam VPC diakses oleh jaringan pribadi

Jika klaster DB Anda tidak dapat diakses secara publik, Anda memiliki opsi berikut untuk mengaksesnya dari jaringan privat:

- AWS Site-to-Site VPNKoneksi. Untuk informasi selengkapnya, lihat Apa itu AWS Site-to-Site VPN?
- AWS Direct Connect Koneksi. Untuk informasi lebih lanjut, lihat Apa itu AWS Direct Connect?
- AWS Client VPN Koneksi. Untuk informasi selengkapnya, lihat Apa itu AWS Client VPN?

Diagram berikut menunjukkan skenario dengan AWS Site-to-Site VPN koneksi.

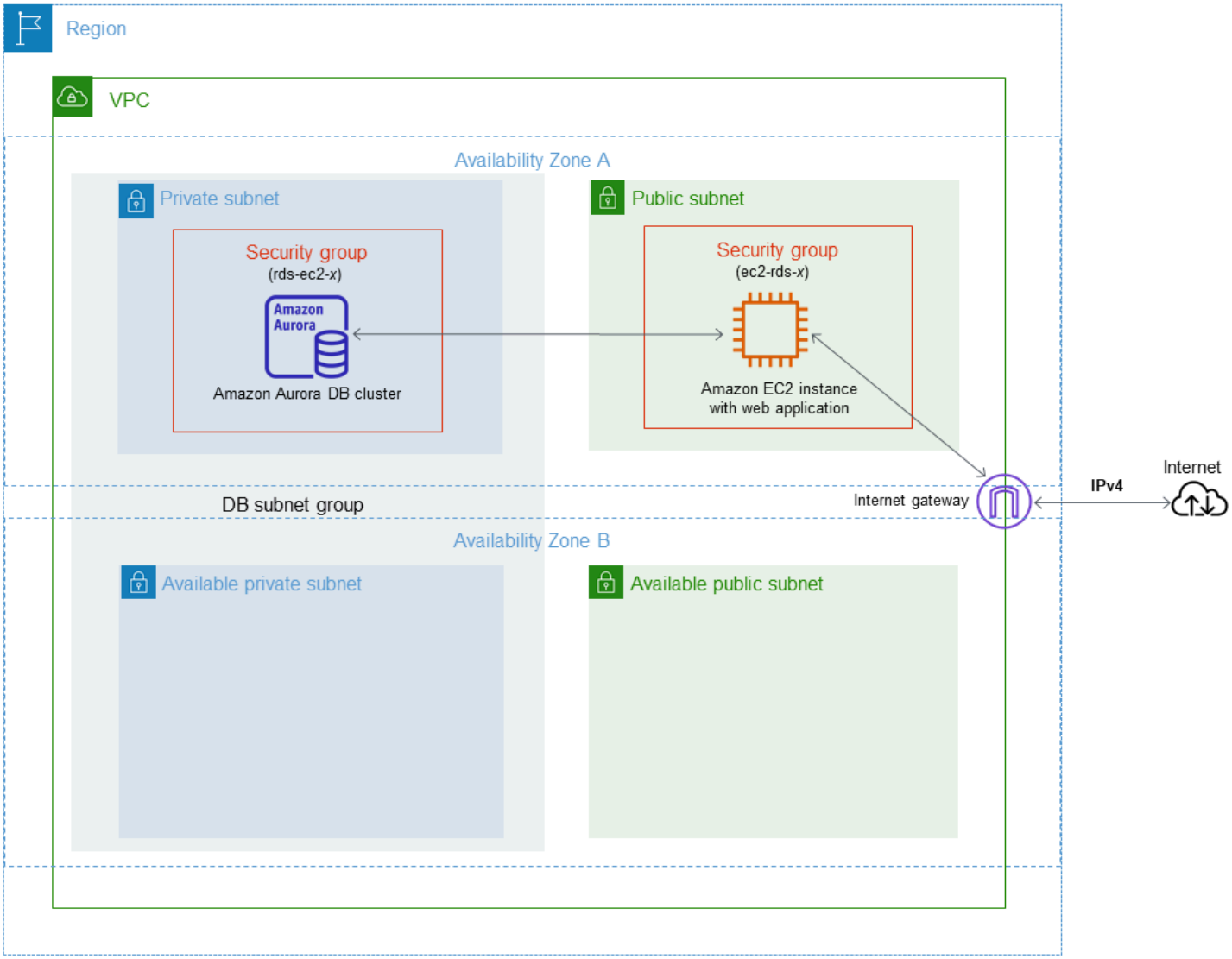



Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.

# Tutorial: Buat VPC untuk digunakan dengan cluster DB (hanya) IPv4

Skenario umum mencakup klaster DB cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. VPC ini berbagi data dengan server web yang berjalan di VPC yang sama. Dalam tutorial ini, Anda membuat VPC untuk skenario ini.

Diagram berikut menunjukkan skenario ini. Untuk informasi tentang skenario lain, lihat Skenario untuk mengakses cluster DB di VPC.




Klaster DB Anda harus tersedia hanya untuk server web Anda, dan bukan untuk internet publik. Dengan demikian, Anda membuat VPC dengan subnet publik maupun privat. Server web di-host di subnet publik, sehingga dapat menjangkau internet publik. Klaster DB di-host di subnet privat. Server web dapat terhubung ke klaster DB karena di-host dalam VPC yang sama. Tetapi klaster DB tidak tersedia untuk internet publik, sehingga memberikan keamanan yang lebih baik.

Tutorial ini mengonfigurasi subnet publik dan privat tambahan di Zona Ketersediaan terpisah. Subnet ini tidak digunakan oleh tutorial. Grup subnet DB RDS membutuhkan subnet, setidaknya di dua Zona Ketersediaan. Subnet tambahan memudahkan untuk mengonfigurasi lebih dari satu instans DB Aurora.

Tutorial ini menjelaskan konfigurasi VPC untuk klaster DB Amazon Aurora. Untuk tutorial yang menunjukkan cara membuat server web untuk skenario VPC ini, lihat Tutorial: Membuat server web dan klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang Amazon VPC, lihat Panduan Memulai Amazon VPC dan Panduan Pengguna Amazon VPC.

> **Tip**
>
> Anda dapat mengatur konektivitas jaringan antara instans Amazon dan cluster EC2 DB secara otomatis saat membuat cluster DB. Konfigurasi jaringan mirip dengan yang dijelaskan dalam tutorial ini. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan sebuah EC2 instance.

## Membuat VPC dengan subnet publik dan privat

Gunakan prosedur berikut untuk membuat VPC dengan subnet publik maupun privat.

Untuk membuat VPC dan subnet

1. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
2. Di pojok kanan atas AWS Management Console, pilih Wilayah untuk membuat VPC Anda. Contoh ini menggunakan Wilayah AS Barat (Oregon).
3. Di sudut kiri atas, pilih Dasbor VPC . Untuk mulai membuat VPC, pilih Buat VPC.
4. Agar Sumber daya dapat dibuat di bagian pengaturan VPC, pilih VPC dan lainnya.
5. Untuk pengaturan VPC, atur nilai-nilai ini:

   - Pembuatan otomatis tag nama – **`tutorial`**
   - IPv4 Blok CIDR — **`10.0.0.0/16`**
   - IPv6 Blok CIDR - Tidak ada blok IPv6 CIDR
   - Penghunian – Default
   - Jumlah Availability Zone (AZs) — 2
   - Kustomisasi AZs - Pertahankan nilai default.

- Jumlah subnet publik – 2
- Jumlah subnet privat – 2
- Sesuaikan subnet blok CIDR – Pertahankan nilai default.
- Gateway NAT ($) – Tidak ada
- Titik akhir VPC – Tidak ada
- Opsi DNS – Pertahankan nilai default.

6. Pilih Buat VPC.

## Buat grup keamanan VPC untuk server web publik

Berikutnya, Anda membuat grup keamanan untuk akses publik. Untuk terhubung ke EC2 instans publik di VPC Anda, Anda menambahkan aturan masuk ke grup keamanan VPC Anda. Ini memungkinkan lalu lintas untuk terhubung dari internet.

Untuk membuat grup keamanan VPC

1. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Buat grup keamanan, atur nilai ini:
   - Nama grup keamanan: **`tutorial-securitygroup`**
   - Deskripsi: **`Tutorial Security Group`**
   - VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc- (tutorial-vpc) *`identifier`*
4. Tambahkan aturan masuk ke grup keamanan.
   a. Tentukan alamat IP yang akan digunakan untuk menyambung ke EC2 instance di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, di jendela atau tab browser yang berbeda, Anda dapat menggunakan layanan di https://checkip.amazonaws.com. Contoh alamat IP adalah `203.0.113.25/32`.

      Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, temukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans-instans Anda menggunakan SSH.

b. Di bagian Aturan masuk, pilih Tambahkan aturan.

c. Tetapkan nilai berikut untuk aturan masuk baru Anda untuk mengizinkan akses SSH ke instans Amazon EC2 Anda. Jika Anda melakukan ini, Anda dapat terhubung ke EC2 instans Amazon Anda untuk menginstal server web dan utilitas lainnya. Anda juga terhubung ke EC2 instans Anda untuk mengunggah konten untuk server web Anda.

   - Jenis: **`SSH`**
   - Sumber: Rentang atau alamat IP dari Langkah a, misalnya: **`203.0.113.25/32`**.

d. Pilih Tambahkan aturan.

e. Atur nilai berikut untuk aturan masuk baru yang akan mengizinkan akses HTTP ke server web Anda.

   - Jenis: **`HTTP`**
   - Sumber: **`0.0.0.0/0`**

5. Untuk membuat grup keamanan, pilih Buat grup keamanan.

   Catat ID grup keamanan karena Anda membutuhkannya nanti dalam tutorial ini.

## Membuat grup keamanan VPC untuk klaster DB privat

Agar klaster DB tetap privat, buat grup keamanan kedua untuk akses privat. Untuk terhubung ke cluster DB pribadi di VPC Anda, Anda menambahkan aturan masuk ke grup keamanan VPC Anda yang mengizinkan lalu lintas dari server web Anda saja.

Untuk membuat grup keamanan VPC

1. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/

2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Buat grup keamanan, atur nilai ini:
   - Nama grup keamanan: **`tutorial-db-securitygroup`**
   - Deskripsi: **`Tutorial DB Instance Security Group`**
   - VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc- (tutorial-vpc) *`identifier`*
4. Tambahkan aturan masuk ke grup keamanan.
   a. Di bagian Aturan masuk, pilih Tambahkan aturan.
   b. Tetapkan nilai berikut untuk aturan masuk baru Anda untuk mengizinkan lalu lintas MySQL di port 3306 dari instans Amazon Anda. EC2 Tindakan ini memungkinkan Anda terhubung dari server web Anda ke klaster DB Anda. Dengan demikian, Anda dapat menyimpan dan mengambil data dari aplikasi web Anda ke basis data Anda.
      - Jenis: **`MySQL/Aurora`**
      - Sumber: Pengidentifikasi grup keamanantutorial-securitygroup yang Anda buat sebelumnya dalam tutorial ini, misalnya: sg-9edd5cfb.
5. Untuk membuat grup keamanan, pilih Buat grup keamanan.

## Membuat grup subnet DB

Grup subnet DB adalah kumpulan subnet yang Anda buat dalam VPC dan yang Anda tetapkan untuk klaster DB. Grup subnet DB memungkinkan Anda untuk menentukan VPC tertentu saat membuat klaster DB.

Untuk membuat grup subnet DB

1. Identifikasi subnet privat untuk basis data Anda di VPC.
   a. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
   b. Pilih Dasbor VPC, lalu pilih Subnet.
   c. Perhatikan subnet dari subnet bernama IDs tutorial-subnet-private1-us-west-2a dan 2-us-west-2b. tutorial-subnet-private

      Anda memerlukan subnet IDs saat Anda membuat grup subnet DB Anda.
2. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

Pastikan Anda terhubung ke konsol Amazon RDS, bukan konsol Amazon VPC.

3. Di panel navigasi, pilih Grup subnet.
4. Pilih Buat grup subnet DB.
5. Di halaman Buat kelompok subnet DB, atur nilai ini di Detail grup subnet:
   - Nama: **`tutorial-db-subnet-group`**
   - Deskripsi: **`Tutorial DB Subnet Group`**
   - VPC: tutorial-vpc (vpc -) *`identifier`*
6. Di bagian Tambahkan subnet, pilih Zona Ketersediaan dan Subnet.

   Untuk tutorial ini, pilih us-west-2a dan us-west-2b untuk Zona Ketersediaan. Untuk Subnet, pilih subnet privat yang Anda identifikasi pada langkah sebelumnya.
7. Pilih Buat.

   Grup subnet DB baru Anda muncul dalam daftar grup subnet DB di konsol RDS. Anda dapat memilih grup subnet DB untuk melihat detail di panel detail di bagian bawah jendela. Detail ini mencakup semua subnet yang terkait dengan grup.

> **Note**
>
> Jika Anda membuat VPC ini untuk menyelesaikan Tutorial: Membuat server web dan klaster DB Amazon Aurora, buat klaster DB dengan mengikuti petunjuk di Membuat klaster DB Amazon Aurora.

## Menghapus VPC

Setelah membuat VPC dan sumber daya lainnya untuk tutorial ini, Anda dapat menghapusnya jika tidak dibutuhkan lagi.

> **Note**
>
> Jika Anda menambahkan sumber daya di VPC yang Anda buat untuk tutorial ini, Anda mungkin perlu menghapusnya sebelum menghapus VPC. Untuk informasi selengkapnya, lihat Menghapus VPC di Panduan Pengguna Amazon VPC.

Untuk menghapus VPC dan sumber daya terkait

1. Hapus grup subnet DB.
    a. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
    b. Di panel navigasi, pilih Grup Subnet.
    c. Pilih grup subnet DB yang ingin Anda hapus, seperti tutorial-db-subnet-group.
    d. Pilih Hapus, lalu pilih Hapus di jendela konfirmasi.
2. Catat ID VPC.
    a. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
    b. Pilih Dasbor VPC, lalu pilih. VPCs
    c. Dalam daftar, identifikasi VPC yang Anda buat, seperti tutorial-vpc.
    d. Catat ID VPC dari VPC yang Anda buat. ID VPC diperlukan di langkah berikutnya.
3. Hapus grup keamanan.
    a. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
    b. Pilih Dasbor VPC, lalu pilih Grup Keamanan.
    c. Pilih grup keamanan untuk instans Amazon RDS DB, seperti tutorial-db-securitygroup.
    d. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.
    e. Pada halaman Grup Keamanan, pilih grup keamanan untuk EC2 instans Amazon, seperti tutorial-securitygroup.
    f. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.
4. Hapus VPC.
    a. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
    b. Pilih Dasbor VPC, lalu pilih. VPCs
    c. Pilih VPC yang ingin Anda hapus, seperti tutorial-vpc.
    d. Untuk Tindakan, pilih Hapus VPC.

        Halaman konfirmasi menunjukkan sumber daya lain yang terkait dengan VPC yang juga akan dihapus, termasuk subnet yang terkait dengannya.
    e. Di halaman konfirmasi, masukkan **`delete`**, lalu pilih Hapus.

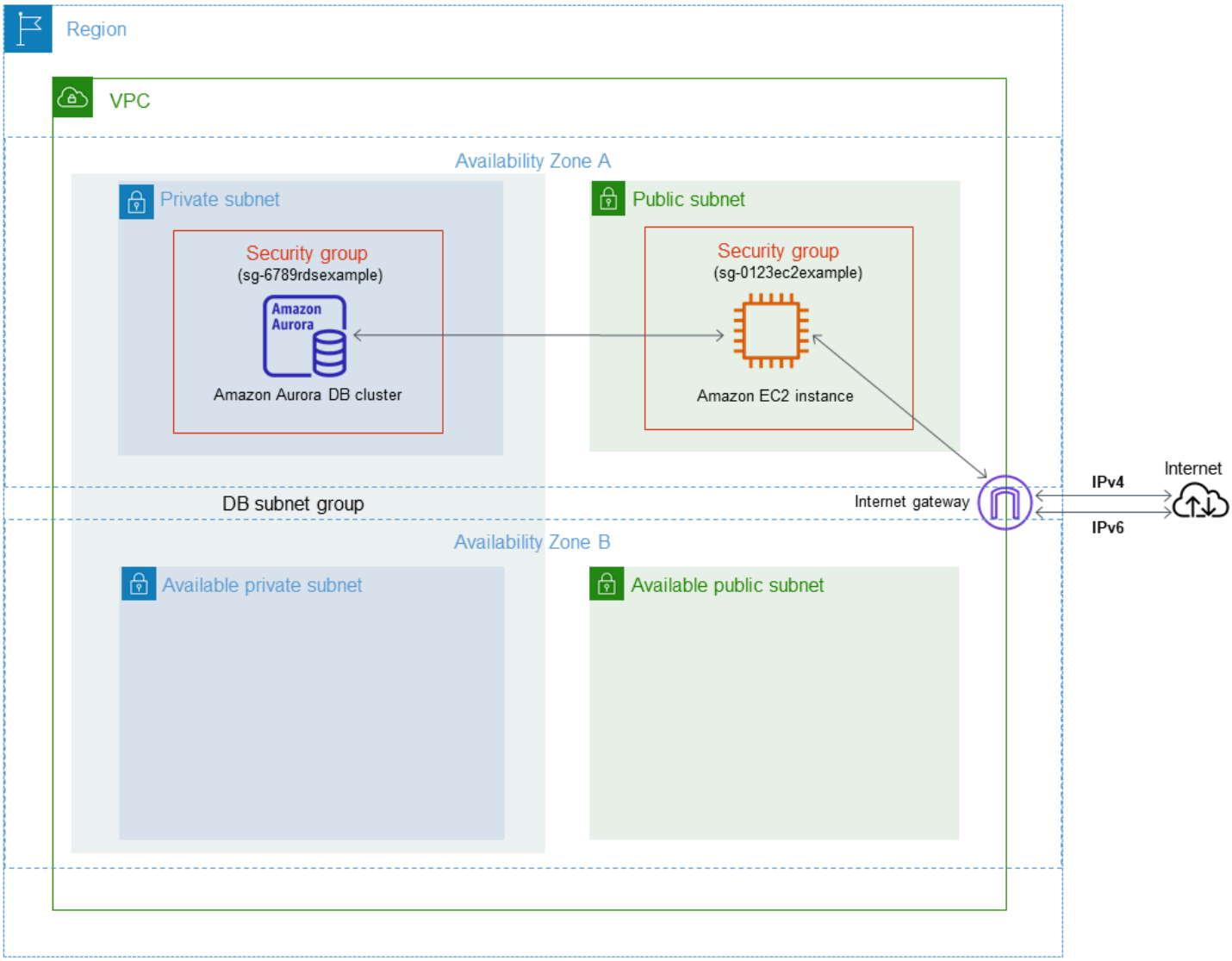
# Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

Skenario umum mencakup klaster DB cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. VPC ini membagikan data dengan instans Amazon EC2 publik yang dijalankan di VPC yang sama.

Dalam tutorial ini, Anda akan membuat VPC untuk skenario ini yang berfungsi dengan basis data yang berjalan dalam mode tumpukan ganda. Mode tumpukan ganda untuk mengaktifkan koneksi melalui protokol penentuan alamat IPv6. Untuk informasi selengkapnya tentang penentuan alamat IP, lihat Penentuan alamat IP Amazon Aurora.

Klaster jaringan tumpukan ganda didukung di sebagian besar wilayah. Untuk informasi selengkapnya, lihat Ketersediaan klaster DB jaringan tumpukan ganda. Untuk melihat batasan mode tumpukan ganda, lihat Batasan untuk klaster DB jaringan tumpukan ganda.

Diagram berikut menunjukkan skenario ini.




Untuk informasi tentang skenario lain, lihat Skenario untuk mengakses cluster DB di VPC.

Klaster DB Anda harus tersedia hanya untuk instans Amazon EC2 Anda, dan bukan untuk internet publik. Dengan demikian, Anda membuat VPC dengan subnet publik maupun privat. Instans Amazon EC2 di-hosting di subnet publik agar dapat menjangkau internet publik. Klaster DB di-host di subnet privat. Instans Amazon EC2 dapat terhubung ke klaster DB karena di-hosting dalam VPC yang sama. Namun, klaster DB tidak tersedia untuk internet publik, sehingga memberikan keamanan yang lebih besar.

Tutorial ini mengonfigurasi subnet publik dan privat tambahan di Zona Ketersediaan terpisah. Subnet ini tidak digunakan oleh tutorial. Grup subnet DB RDS membutuhkan subnet, setidaknya di dua Zona Ketersediaan. Subnet tambahan memudahkan untuk mengonfigurasi lebih dari satu instans DB Aurora.

Untuk membuat klaster DB yang menggunakan mode tumpukan ganda, tetapkan Mode tumpukan ganda untuk pengaturan Jenis jaringan. Anda juga dapat mengubah klaster DB dengan pengaturan yang sama. Untuk informasi selengkapnya tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Tutorial ini menjelaskan konfigurasi VPC untuk klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang Amazon VPC, lihat Panduan Pengguna Amazon VPC.

## Membuat VPC dengan subnet publik dan privat

Gunakan prosedur berikut untuk membuat VPC dengan subnet publik maupun privat.

Untuk membuat VPC dan subnet

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah untuk membuat VPC Anda. Contoh ini menggunakan Wilayah US East (Ohio).
3. Di sudut kiri atas, pilih Dasbor VPC . Untuk mulai membuat VPC, pilih Buat VPC.
4. Agar Sumber daya dapat dibuat di bagian pengaturan VPC, pilih VPC dan lainnya.
5. Untuk Pengaturan VPC lainnya, atur nilai-nilai ini:

   - Pembuatan otomatis tag nama – **`tutorial-dual-stack`**
   - Blok CIDR IPv4: – **`10.0.0.0/16`**
   - Blok CIDR IPv6 – Blok CIDR IPv6 yang disediakan Amazon
   - Penghunian – Default
   - Jumlah Zona Ketersediaan (AZ) – 2
   - Sesuaikan AZ – Pertahankan nilai default.
   - Jumlah subnet publik – 2
   - Jumlah subnet privat – 2
   - Sesuaikan subnet blok CIDR – Pertahankan nilai default.
   - Gateway NAT ($) – Tidak ada
   - Gateway internet khusus egress – Tidak
   - Titik akhir VPC – Tidak ada
   - Opsi DNS – Pertahankan nilai default.

> **Note**
>
> Amazon RDS membutuhkan setidaknya dua subnet di dua Zona Ketersediaan yang berbeda untuk mendukung deployment instans DB Multi-AZ. Tutorial ini membuat deployment AZ Tunggal, tetapi persyaratannya memudahkan konversi ke deployment instans DB Multi-AZ di masa mendatang.

6. Pilih Buat VPC.

## Membuat grup keamanan VPC untuk instans Amazon EC2 publik

Berikutnya, Anda membuat grup keamanan untuk akses publik. Untuk terhubung ke instans EC2 publik di VPC Anda, tambahkan aturan masuk ke grup keamanan VPC Anda yang mengizinkan lalu lintas untuk terhubung dari internet.

Untuk membuat grup keamanan VPC

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Buat grup keamanan, atur nilai ini:
   - Nama grup keamanan: **`tutorial-dual-stack-securitygroup`**
   - Deskripsi: **`Tutorial Dual-Stack Security Group`**
   - VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc-***`identifier`*** (tutorial–tumpukan ganda-vpc)
4. Tambahkan aturan masuk ke grup keamanan.
   a. Tentukan alamat IP yang akan digunakan untuk terhubung ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH).

      Contoh rentang alamat Protokol Internet versi 4 (IPv4) adalah `203.0.113.25/32`. Contoh rentang alamat Protokol Internet versi 6 (IPv6) adalah `2001:db8:1234:1a00::/64`.

      Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, temukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk IPv4 atau `::0` untuk IPv6, Anda memungkinkan semua alamat IP untuk mengakses instans publik Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans-instans Anda.

b. Di bagian Aturan masuk, pilih Tambahkan aturan.

c. Atur nilai berikut untuk aturan masuk baru Anda yang akan mengizinkan akses Secure Shell (SSH) ke instans Amazon EC2 Anda. Jika Anda melakukan ini, Anda dapat terhubung ke instans EC2 Anda untuk menginstal klien SQL dan aplikasi lainnya. Tentukan alamat IP agar Anda dapat mengakses instans EC2 Anda:

- Jenis: **`SSH`**
- Sumber: Rentang atau alamat IP dari langkah a. Contoh alamat IP IPv4 adalah **`203.0.113.25/32`**. Contoh alamat IP IPv6 adalah **`2001:DB8::/32`**.

5. Untuk membuat grup keamanan, pilih Buat grup keamanan.

   Catat ID grup keamanan karena Anda membutuhkannya nanti dalam tutorial ini.

## Membuat grup keamanan VPC untuk klaster DB privat

Agar klaster DB tetap privat, buat grup keamanan kedua untuk akses privat. Untuk terhubung ke klaster DB di VPC Anda, tambahkan aturan masuk ke grup keamanan VPC Anda. Hal ini mengizinkan lalu lintas dari instans Amazon EC2 Anda saja.

Untuk membuat grup keamanan VPC

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Buat grup keamanan, atur nilai ini:

- Nama grup keamanan: **`tutorial-dual-stack-db-securitygroup`**
- Deskripsi: **`Tutorial Dual-Stack DB Instance Security Group`**

- VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc-***identifier*** (tutorial–tumpukan ganda-vpc)

4. Tambahkan aturan masuk ke grup keamanan:

   a. Di bagian Aturan masuk, pilih Tambahkan aturan.

   b. Atur nilai berikut untuk aturan masuk baru Anda yang akan mengizinkan lalu lintas MySQL di port 3306 dari instans Amazon EC2 Anda. Dengan melakukannya, Anda dapat terhubung dari instans EC2 Anda ke klaster DB Anda. Melakukan hal ini berarti Anda dapat mengirimkan data dari instans EC2 Anda ke basis data Anda.

      - Jenis: MySQL/Aurora
      - Sumber: Pengidentifikasi grup keamanan tutorial-tumpukan ganda-securitygroup yang Anda buat sebelumnya dalam tutorial ini, misalnya sg-9edd5cfb.

5. Untuk membuat grup keamanan, pilih Buat grup keamanan.

## Membuat grup subnet DB

Grup subnet DB adalah kumpulan subnet yang Anda buat dalam VPC dan yang Anda tetapkan untuk klaster DB. Dengan menggunakan grup subnet DB, Anda dapat menentukan VPC tertentu saat membuat klaster DB. Untuk membuat grup subnet DB yang kompatibel dengan `DUAL`, semua subnet harus kompatibel dengan `DUAL`. Agar kompatibel dengan `DUAL`, subnet harus memiliki CIDR IPv6 yang dikaitkan dengan subnet tersebut.

Untuk membuat grup subnet DB

1. Identifikasi subnet privat untuk basis data Anda di VPC.

   a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.

   b. Pilih Dasbor VPC, lalu pilih Subnet.

   c. Perhatikan ID subnet dari subnet bernama tutorial-dual-stack-subnet-private1-us-west-2a dan tutorial-dual-stack-subnet-private2-us-west-2b.

      Anda memerlukan ID subnet saat membuat grup subnet DB.

2. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   Pastikan Anda terhubung ke konsol Amazon RDS, bukan konsol Amazon VPC.

3. Di panel navigasi, pilih Grup subnet.

4. Pilih Buat grup subnet DB.
5. Di halaman Buat kelompok subnet DB, atur nilai ini di Detail grup subnet:
   - Nama: **`tutorial-dual-stack-db-subnet-group`**
   - Deskripsi: **`Tutorial Dual-Stack DB Subnet Group`**
   - VPC: tutorial-dual-stack-vpc ***`(pengidentifikasi vpc)`***
6. Di bagian Tambahkan subnet, pilih nilai untuk opsi Zona Ketersediaan dan Subnet.

   Untuk tutorial ini, pilih us-east-2a dan us-east-2b untuk Zona Ketersediaan. Untuk Subnet, pilih subnet privat yang Anda identifikasi pada langkah sebelumnya.
7. Pilih Buat.

Grup subnet DB baru Anda muncul dalam daftar grup subnet DB di konsol RDS. Anda dapat memilih grup subnet DB untuk melihat detailnya. Detail ini termasuk protokol penentuan alamat yang didukung, semua subnet yang terkait dengan grup tersebut, dan jenis jaringan yang didukung oleh grup subnet DB.

## Membuat instans Amazon EC2 dalam mode tumpukan ganda

Untuk membuat instans Amazon EC2, ikuti petunjuk dalam Meluncurkan instans menggunakan panduan instans peluncuran baru di Panduan Pengguna Amazon EC2.

Di halaman Mengonfigurasi Detail Instans, atur nilai-nilai ini dan biarkan nilai lainnya sebagai default:

- Jaringan – Pilih VPC yang ada dengan subnet publik dan privat seperti tutorial-tumpukan ganda-vpc (vpc-*`identifier`*) yang dibuat di Membuat VPC dengan subnet publik dan privat.
- Subnet — Pilih subnet publik yang ada, seperti subnet- ***`identifier`*** | tutorial-dual-stack-subnet -public1-us-east-2a | us-east-2a dibuat di. Membuat grup keamanan VPC untuk instans Amazon EC2 publik
- IP Publik yang Otomatis Ditetapkan – Pilih Aktifkan.
- IP IPv6 yang Otomatis Ditetapkan – Pilih Aktifkan.
- Firewall (grup keamanan) – Pilih Pilih grup keamanan yang ada.
- Grup keamanan umum – Pilih grup keamanan yang ada, seperti `tutorial-securitygroup` yang dibuat di Membuat grup keamanan VPC untuk instans Amazon EC2 publik. Pastikan grup keamanan yang Anda pilih menyertakan aturan masuk untuk akses Secure Shell (SSH) dan HTTP.

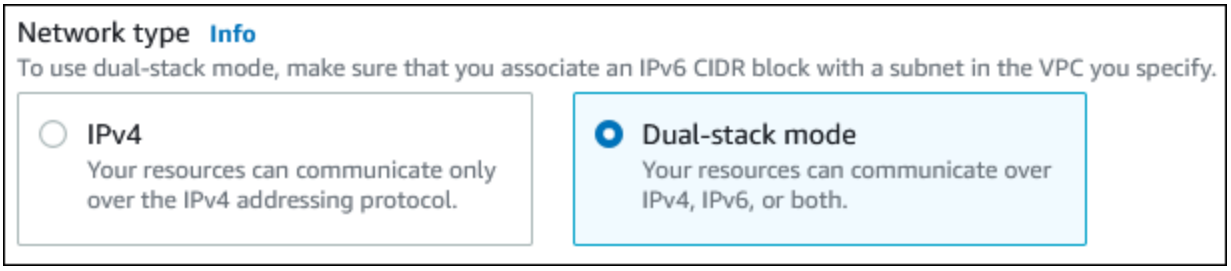
## Membuat klaster DB dalam mode tumpukan ganda

Pada langkah ini, Anda akan membuat klaster DB yang berjalan dalam mode tumpukan ganda.

Membuat instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Contoh ini menggunakan Wilayah AS Timur (Ohio).
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.
5. Di halaman Membuat basis data, pastikan opsi Pembuatan standar dipilih, lalu pilih jenis mesin Aurora MySQL DB.
6. Di bagian Konektivitas, atur nilai-nilai ini:

   - Jenis jaringan – Pilih Mode tumpukan ganda.

   - Cloud privat virtual (VPC) – Pilih VPC yang ada dengan subnet publik dan privat seperti tutorial-tumpukan ganda-vpc (vpc-*identifier*) yang dibuat di Membuat VPC dengan subnet publik dan privat.

     VPC tersebut harus memiliki subnet di Zona Ketersediaan yang berbeda.

   - Grup subnet DB – Pilih grup subnet DB untuk VPC, seperti tutorial-tumpukan ganda-db-subnet-group yang dibuat di Membuat grup subnet DB.
   - Akses publik – Pilih Tidak.
   - Grup keamanan VPC (firewall) – Pilih Pilih grup keamanan yang ada.
   - Grup keamanan VPC yang ada – Pilih grup keamanan VPC yang sudah ada dan dikonfigurasi untuk akses privat, seperti tutorial-tumpukan ganda-db-securitygroup yang dibuat di Membuat grup keamanan VPC untuk klaster DB privat.

Hapus grup keamanan lainnya, seperti grup keamanan default, dengan memilih X yang dikaitkan dengan masing-masing grup keamanan.

- Zona Ketersediaan – Pilih us-west-2a.

  Untuk menghindari lalu lintas AZ, pastikan instans DB dan instans EC2 berada di Zona Ketersediaan yang sama.

7. Untuk bagian yang lainnya, tentukan pengaturan klaster DB Anda. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

## Menghubungkan ke instans Amazon EC2 dan klaster DB

Setelah membuat instans Amazon EC2 dan klaster DB dalam mode tumpukan ganda, Anda dapat terhubung ke masing-masing instans menggunakan protokol IPv6. Untuk menyambung ke instans Amazon EC2 menggunakan protokol IPv6, ikuti petunjuk di Connect to Linux Anda di Panduan Pengguna Amazon EC2.

Untuk menghubungkan ke klaster DB Aurora MySQL dari instans Amazon EC2, ikuti petunjuk di Menghubungkan ke klaster DB Aurora MySQL.

## Menghapus VPC

Setelah membuat VPC dan sumber daya lainnya untuk tutorial ini, Anda dapat menghapusnya jika tidak dibutuhkan lagi.

Jika Anda menambahkan sumber daya di VPC yang Anda buat untuk tutorial ini, Anda mungkin perlu menghapusnya sebelum menghapus VPC. Contoh sumber dayanya adalah instans Amazon EC2 atau klaster DB. Untuk informasi selengkapnya, lihat Menghapus VPC di Panduan Pengguna Amazon VPC.

Untuk menghapus VPC dan sumber daya terkait

1. Hapus grup subnet DB:

   a. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
   b. Di panel navigasi, pilih Grup subnet.
   c. Pilih grup subnet DB yang akan dihapus, seperti tutorial-db-subnet-group.
   d. Pilih Hapus, lalu pilih Hapus di jendela konfirmasi.

2. Catat ID VPC:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih VPC.
c. Dalam daftar, identifikasi VPC yang Anda buat, seperti tutorial-tumpukan ganda-vpc.
d. Catat nilai ID VPC dari VPC yang Anda buat. Anda memerlukan ID VPC ini di langkah selanjutnya.

3. Hapus grup keamanan:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih Grup Keamanan.
c. Pilih grup keamanan untuk instans DB Amazon RDS, seperti tutorial-tumpukan ganda-db-securitygroup.
d. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.
e. Di halaman Grup Keamanan, pilih grup keamanan untuk instans Amazon EC2, seperti tutorial-tumpukan ganda-securitygroup.
f. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.

4. Hapus gateway NAT:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih Gateway NAT.
c. Pilih gateway NAT dari VPC yang Anda buat. Gunakan ID VPC untuk mengidentifikasi gateway NAT yang benar.
d. Untuk Tindakan, pilih Hapus gateway NAT.
e. Di halaman konfirmasi, masukkan **`delete`**, lalu pilih Hapus.

5. Hapus VPC:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih VPC.
c. Pilih VPC yang ingin Anda hapus, seperti tutorial-tumpukan ganda-vpc.
d. Untuk Tindakan, pilih Hapus VPC.

   Halaman konfirmasi menunjukkan sumber daya lain yang terkait dengan VPC yang juga akan dihapus, termasuk subnet yang terkait dengannya.

e. Di halaman konfirmasi, masukkan **`delete`**, lalu pilih Hapus.

6. Rilis alamat IP Elastis:

    a. Buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
    b. Pilih Dasbor EC2, lalu pilih IP Elastis.
    c. Pilih alamat IP Elastis yang ingin Anda rilis.
    d. Untuk Tindakan, pilih Rilis alamat IP Elastis.
    e. Di halaman konfirmasi, pilih Rilis.

# Kuota dan batasan untuk Amazon Aurora

Setelah itu, Anda dapat menemukan deskripsi kuota sumber daya dan batasan penamaan untuk Amazon Aurora.

Topik


- Kuota dalam Amazon Aurora
- Batasan penamaan dalam Amazon Aurora
- Batas ukuran Amazon Aurora


## Kuota dalam Amazon Aurora

Setiap AWS akun memiliki kuota, untuk setiap AWS Wilayah, pada jumlah sumber daya Amazon Aurora yang dapat dibuat. Setelah kuota sumber daya tercapai, panggilan tambahan untuk membuat sumber daya tersebut akan gagal dengan pengecualian.

Tabel berikut mencantumkan sumber daya dan kuota mereka per AWS Wilayah.

| Nama | Default | Dapat disesuaikan | Deskripsi |
|---|---|---|---|
| Otorisasi per grup keamanan DB | Setiap Wilayah yang didukung: 20 | Tidak | Jumlah otorisasi grup keamanan per grup keamanan DB |
| Titik akhir kustom per klaster DB | Setiap Wilayah yang didukung: 5 | Ya | Jumlah maksimum titik akhir kustom yang dapat Anda buat per cluster Aurora DB di akun ini di Wilayah saat ini. Nilai ini mencerminkan jumlah titik akhir kustom tertinggi dalam cluster DB di akun. Cluster DB lain di akun mungkin memiliki jumlah |

| Nama | Default | Dapat disesuaikan | Deskripsi |
| --- | --- | --- | --- |
| | | | titik akhir kustom yang lebih rendah. |
| Versi mesin kustom | Setiap Wilayah yang didukung: 40 | Ya | Jumlah maksimum versi mesin kustom yang diizinkan di akun di Wilayah saat ini |
| Grup parameter klaster DB | Setiap Wilayah yang didukung: 50 | Ya | Jumlah maksimum grup parameter klaster DB |
| Klaster DB | Setiap Wilayah yang didukung: 40 | Ya | Jumlah maksimum klaster Aurora yang diizinkan di akun ini dalam Wilayah saat ini |
| Instans DB | ap-south-1:20<br><br>Masing-masing Wilayah yang didukung lainnya: 40 | Ya | Jumlah maksimum instans DB yang diizinkan di akun ini dalam Wilayah saat ini |
| Grup pecahan DB | Setiap Wilayah yang didukung: 5 | Ya | Jumlah maksimum grup pecahan DB untuk Aurora Limitless Database di akun ini di Wilayah saat ini |
| Grup subnet DB | ap-south-1:20<br><br>Masing-masing Wilayah yang didukung lainnya: 50 | Ya | Jumlah maksimum grup subnet DB |

| Nama | Default | Dapat disesu an | Deskripsi |
| --- | --- | --- | --- |
| Ukuran konten permintaan HTTP API Data | Setiap Wilayah yang didukung: 4 Megabyte | Tidak | Ukuran maksimum yang diizinkan untuk konten permintaan HTTP. |
| Pasangan rahasia klaster serentak maksimum API Data | Setiap Wilayah yang didukung: 30 | Tidak | Jumlah maksimum pasangan unik cluster dan rahasia Aurora Serverless v1 DB dalam permintaan API Data bersamaan untuk akun ini di Wilayah saat ini. AWS |
| Permintaan serentak maksimum API Data | Setiap Wilayah yang didukung: 500 | Tidak | Jumlah maksimum permintaan Data API ke cluster Aurora Serverles s v1 DB yang menggunak an rahasia yang sama dan dapat diproses pada saat yang sama. Permintaan tambahan dimasukkan ke dalam antrean dan diproses setelah permintaan dalam proses selesai. |
| Ukuran set hasil maksimum API Data | Setiap Wilayah yang didukung: 1 Megabyte | Tidak | Ukuran maksimum set hasil basis data yang dapat dikembalikan oleh API Data. |

| Nama | Default | Dapat disesuaikan | Deskripsi |
| --- | --- | --- | --- |
| Ukuran maksimum API Data string respons JSON | Setiap Wilayah yang didukung: 10 Megabyte | Tidak | Ukuran maksimum string respons JSON yang disederhanakan dikembalikan oleh API Data RDS. |
| Permintaan API Data per detik | Setiap Wilayah yang didukung: 1.000 per detik | Tidak | Jumlah maksimum permintaan ke Data API per detik yang diizinkan untuk akun ini di AWS Wilayah saat ini. Kuota ini hanya berlaku untuk klaster v1 Amazon Aurora Tanpa Server. |
| Langganan peristiwa | Setiap Wilayah yang didukung: 20 | Ya | Jumlah maksimum langganan peristiwa |
| Peran IAM per klaster DB | Setiap Wilayah yang didukung: 5 | Ya | Jumlah maksimum peran IAM yang terkait dengan klaster DB |
| Peran IAM per instans DB | Setiap Wilayah yang didukung: 5 | Ya | Jumlah maksimum peran IAM yang terkait dengan instans DB |
| Integrasi | Setiap Wilayah yang didukung: 100 | Tidak | Jumlah maksimum integrasi yang diizinkan dalam akun ini di Wilayah saat ini AWS |
| Snapshot klaster DB manual | Setiap Wilayah yang didukung: 100 | Ya | Jumlah maksimum snapshot klaster DB manual |

| Nama | Default | Dapat disesuaikan | Deskripsi |
| --- | --- | --- | --- |
| Snapshot instans DB manual | Setiap Wilayah yang didukung: 100 | Ya | Jumlah maksimum snapshot instans DB manual |
| Grup opsi | Setiap Wilayah yang didukung: 20 | Ya | Jumlah maksimum grup opsi |
| Grup parameter | ap-south-1:20<br><br>Masing-masing Wilayah yang didukung lainnya: 50 | Ya | Jumlah maksimum grup parameter |
| Proksi | Setiap Wilayah yang didukung: 20 | Ya | Jumlah maksimum proxy yang diizinkan di akun ini di Wilayah saat ini AWS |
| Replika baca per primer | Setiap Wilayah yang didukung: 15 | Ya | Jumlah maksimum replika baca per instans DB primer. Kuota ini tidak dapat disesuaikan untuk Amazon Aurora. |
| Instans DB cadangan | Setiap Wilayah yang didukung: 40 | Ya | Jumlah maksimum instans DB cadangan yang diizinkan di akun ini di Wilayah saat ini AWS |
| Grup keamanan | ap-south-1:20<br><br>Masing-masing Wilayah yang didukung lainnya: 25 | Ya | Jumlah maksimum aturan grup keamanan DB |

| Nama | Default | Dapat disesuaikan | Deskripsi |
| --- | --- | --- | --- |
| Subnet per grup subnet DB | Setiap Wilayah yang didukung: 20 | Tidak | Jumlah maksimum subnet per grup subnet DB |
| Total penyimpanan untuk semua instans DB | Setiap Wilayah yang didukung: 100.000 Gigabyte | Ya | Total penyimpanan maksimum (dalam GB) pada volume EBS untuk semua instans DB Amazon RDS yang ditambahkan bersama. Kuota ini tidak berlaku untuk Amazon Aurora, yang memiliki volume cluster maksimum 128 TiB untuk setiap cluster DB. |

**Note**

Secara default, Anda dapat memiliki hingga total 40 instans DB. Instans DB RDS, instans DB Aurora, instans Amazon Neptune, dan instans Amazon DocumentDB berlaku untuk kuota ini. Jika aplikasi Anda memerlukan lebih banyak instans DB, Anda dapat meminta instans DB tambahan dengan membuka Konsol Service Quotas. Di panel navigasi, pilih Layanan AWS . Pilih Amazon Relational Database Service (Amazon RDS), pilih kuota, dan ikuti arahan untuk meminta peningkatan kuota. Untuk informasi selengkapnya, lihat Meminta peningkatan kuota di Panduan Pengguna Service Quotas.

Untuk RDS untuk Oracle, Anda dapat membuat hingga 15 replika baca per instans DB sumber di setiap Wilayah, tetapi sebaiknya batasi replika menjadi 5 untuk meminimalkan kelambatan replikasi.

Pencadangan yang dikelola oleh AWS Backup dianggap sebagai snapshot cluster DB manual, tetapi tidak dihitung dalam kuota snapshot cluster manual. Untuk selengkapnya AWS Backup, lihat Panduan AWS Backup Pengembang.

Jika Anda menggunakan operasi API RDS dan melebihi kuota default untuk jumlah panggilan per detik, API Amazon RDS akan mengeluarkan kesalahan seperti berikut.

ClientError: Terjadi kesalahan (ThrottlingException) saat memanggil *API_name* operasi: Tingkat terlampaui.

Di sini, kurangi jumlah panggilan per detik. Kuota dimaksudkan untuk mencakup sebagian besar kasus penggunaan. Jika kuota yang lebih tinggi diperlukan, Anda dapat meminta kenaikan kuota dengan menggunakan salah satu opsi berikut:

- Dari konsol, buka konsol Service Quotas.
- Dari AWS CLI, gunakan request-service-quota-increase AWS CLI perintah.

Untuk informasi lebih lanjut, lihat Panduan Pengguna Kuota Layanan.

## Batasan penamaan dalam Amazon Aurora

Kendala penamaan di Amazon RDS Aurora adalah sebagai berikut:

- Pengidentifikasi cluster DB:
  - Harus berisi 1–63 karakter alfanumerik atau tanda hubung.
  - Karakter pertamanya harus berupa huruf.
  - Tidak boleh diakhiri dengan satu tanda hubung atau berisi dua tanda hubung berturut-turut.
  - Harus unik untuk semua instans DB per AWS akun, per AWS Wilayah.
- Nama database awal — Batasan nama database berbeda antara Aurora MySQL dan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat pengaturan yang tersedia saat membuat setiap cluster DB.
- Nama pengguna master — Kendala nama pengguna master berbeda untuk setiap mesin database. Untuk informasi selengkapnya, lihat pengaturan yang tersedia saat membuat cluster DB.
- Kata sandi utama:

  - Kata sandi untuk pengguna utama basis data dapat mencakup karakter ASCII yang dapat dicetak kecuali /, ', ", @, atau spasi.
  - Kata sandi dapat berisi jumlah karakter ASCII yang dapat dicetak berikut tergantung pada mesin DB:
    - Aurora MySQL: 8–41
    - Aurora PostgreSQL: 8–99
- Grup parameter DB:
  - Harus berisi 1–255 karakter alfanumerik.
  - Karakter pertamanya harus berupa huruf.
  - Tanda hubung diperbolehkan, tetapi nama tidak boleh diakhiri dengan tanda hubung atau berisi dua tanda hubung berturut-turut.
- Grup subnet DB:
  - Harus berisi 1–255 karakter.
  - Karakter alfanumerik, spasi, tanda hubung, garis bawah, dan titik diperbolehkan.

# Batas ukuran Amazon Aurora

Batas ukuran penyimpanan

Ukuran maksimum volume cluster Aurora bervariasi menurut versi mesin:

Maksimum 256 TiB:

- Versi Aurora PostgreSQL:
  - 17.5 dan semua versi yang lebih tinggi
  - 16.9 dan lebih tinggi
  - 15.13 dan lebih tinggi

Maksimum 128 TiB:

- Semua versi Aurora PostgreSQL sebelumnya
- Semua tersedia Aurora MySQL 3 versi; Aurora MySQL versi 2, versi 2.09 dan lebih tinggi

Untuk informasi selengkapnya tentang penskalaan penyimpanan otomatis, lihatCara penyimpanan Aurora berubah ukuran secara otomatis.

Untuk memantau ruang penyimpanan yang tersisa, Anda dapat menggunakan metrik `AuroraVolumeBytesLeftTotal`. Untuk informasi selengkapnya, lihat Metrik tingkat klaster untuk Amazon Aurora.

Batas ukuran tabel SQL

Untuk klaster DB Aurora MySQL, ukuran tabel maksimumnya adalah 64 tebibyte (TiB). Untuk klaster DB Aurora PostgreSQL, ukuran tabel maksimumnya adalah 32 tebibyte (TiB). Sebaiknya Anda mengikuti praktik terbaik desain tabel, seperti mempartisi tabel besar.

Batas ID ruang tabel

ID ruang tabel maksimum untuk Aurora MySQL adalah 2147483647. Jika Anda sering membuat dan menjatuhkan tabel, pastikan untuk mengetahui ruang tabel Anda IDs dan rencanakan untuk menggunakan dump logis. Lihat informasi yang lebih lengkap di Migrasi logis dari My SQL ke Amazon Aurora My SQL dengan menggunakan mysqldump.

# Pemecahan Masalah untuk Amazon Aurora

Gunakan bagian berikut untuk membantu memecahkan masalah yang Anda miliki dengan instans DB di Amazon RDS dan Amazon Aurora.

Topik


- Tidak dapat terhubung ke instans DB Amazon RDS
- Masalah keamanan Amazon RDS
- Mengatur ulang kata sandi pemilik instans DB
- Penghentian atau boot ulang instans DB Amazon RDS
- Perubahan parameter DB Amazon RDS tidak diberlakukan
- Masalah memori yang dapat dikosongkan di Amazon Aurora
- Masalah replikasi Amazon Aurora MySQL


Untuk informasi tentang masalah debug menggunakan API Amazon RDS, lihat Pemecahan masalah aplikasi di Aurora.

## Tidak dapat terhubung ke instans DB Amazon RDS

Jika Anda tidak dapat terhubung ke instans DB, berikut adalah penyebab-penyebab umumnya:

- Aturan masuk – Aturan akses yang diberlakukan oleh firewall lokal dan alamat IP yang diizinkan untuk mengakses instans DB mungkin tidak cocok. Kemungkinan besar masalahnya adalah aturan masuk dalam grup keamanan Anda.

  Secara default, instans DB tidak mengizinkan akses. Akses diberikan melalui grup keamanan yang terkait dengan VPC yang mengizinkan lalu lintas masuk dan keluar dari instans DB. Jika perlu, tambahkan aturan masuk dan keluar untuk situasi khusus Anda ke grup keamanan. Anda dapat menentukan alamat IP, rentang alamat IP, atau grup keamanan VPC lainnya.

  > **Note**
  >
  > Saat menambahkan aturan masuk baru, Anda dapat memilih IP Saya untuk Sumber agar dapat mengizinkan akses ke instans DB dari alamat IP yang terdeteksi di browser.

Untuk informasi selengkapnya tentang menyiapkan grup keamanan, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan.

> **Note**
>
> Koneksi klien dari alamat IP di dalam rentang 169.254.0.0/16 tidak diizinkan. Rentang ini adalah Automatic Private IP Addressing Range (APIPA) yang digunakan untuk alamat tautan lokal.

- Aksesibilitas publik – Untuk terhubung ke instans DB dari luar VPC, seperti menggunakan aplikasi klien, instans harus memiliki alamat IP publik yang ditetapkan untuk instans tersebut.

  Agar instans dapat diakses oleh publik, ubah instans dan pilih Ya di bagian Aksesibilitas publik. Untuk informasi selengkapnya, lihat Menyembunyikan klaster DB dalam VPC dari internet.
- Port — Port yang Anda tentukan saat membuat instans DB tidak dapat digunakan untuk mengirim atau menerima komunikasi karena pembatasan firewall lokal Anda. Untuk menentukan apakah jaringan Anda memungkinkan port tertentu digunakan untuk komunikasi masuk dan keluar, hubungi administrator jaringan Anda.
- Ketersediaan – Untuk instans DB yang baru dibuat, instans DB memiliki status `creating` hingga instans DB siap digunakan. Ketika statusnya berubah menjadi `available`, Anda dapat terhubung ke instans DB. Tergantung pada ukuran instans DB Anda, perlu waktu hingga 20 menit sebelum instans tersedia.
- Gateway internet – Agar instans DB dapat diakses publik, subnet dalam grup subnet DB tersebut harus memiliki gateway internet.

  Mengonfigurasi gateway internet untuk subnet

  1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
  2. Di panel navigasi, pilih Basis Data lalu pilih instans DB.
  3. Di tab Konektivitas & keamanan, tuliskan nilai ID VPC di VPC dan subnet ID di Subnet.
  4. Buka konsol VPC Amazon di. https://console.aws.amazon.com/vpc/
  5. Di panel navigasi, pilih Gateway Internet. Pastikan ada gateway internet yang dilampirkan ke VPC Anda. Atau, pilih Buat Gateway Internet untuk membuat gateway internet. Pilih gateway internet, lalu pilih Lampirkan ke VPC dan ikuti arahan untuk melampirkannya ke VPC Anda.

6. Di panel navigasi, pilih Subnet, lalu pilih subnet Anda.
7. Di tab Tabel Rute, pastikan ada rute dengan `0.0.0.0/0` sebagai tujuan dan gateway internet untuk VPC sebagai target.

   Jika Anda terhubung ke instans menggunakan IPv6 alamatnya, verifikasi bahwa ada rute untuk semua IPv6 lalu lintas (`::/0`) yang mengarah ke gateway internet. Jika tidak, lakukan tindakan berikut:

   a. Pilih ID tabel rute (rtb-xxxxxxxx) untuk menavigasi ke tabel rute.
   b. Di tab Rute, pilih Edit rute. Pilih Tambahkan rute, gunakan `0.0.0.0/0` sebagai tujuan, dan gateway internet sebagai target.

      Untuk IPv6, pilih Tambahkan rute, gunakan `::/0` sebagai tujuan dan gateway internet sebagai target.
   c. Pilih Simpan rute.

   Juga, jika Anda mencoba untuk terhubung ke IPv6 endpoint, pastikan bahwa rentang IPv6 alamat klien diotorisasi untuk terhubung ke instans DB.

Untuk informasi selengkapnya, lihat Menggunakan klaster DB di VPC.

## Menguji koneksi ke instans DB

Anda dapat menguji koneksi ke instans DB menggunakan alat Linux atau Microsoft Windows umum.

Dari terminal Linux atau Unix, Anda dapat menguji koneksi dengan memasukkan hal berikut. Ganti *`DB-instance-endpoint`* dengan titik akhir dan *`port`* dengan port instans DB Anda.

```
nc -zv DB-instance-endpoint port
```

Misalnya, hal berikut menunjukkan contoh perintah dan nilai kembali.

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Pengguna Windows dapat menggunakan Telnet untuk menguji koneksi ke instans DB. Tindakan Telnet tidak didukung selain untuk menguji koneksi. Jika koneksi berhasil, tindakan tidak akan mengembalikan pesan. Jika koneksi gagal, Anda akan menerima pesan kesalahan seperti berikut.

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

Jika tindakan Telnet berhasil, artinya grup keamanan Anda dikonfigurasi dengan benar.

> **Note**
>
> Amazon RDS tidak menerima lalu lintas Internet Control Message Protocol (ICMP), termasuk ping.

## Memecahkan masalah autentikasi koneksi

Dalam beberapa kasus, Anda dapat terhubung ke instans DB tetapi mendapatkan kesalahan autentikasi. Dalam kasus ini, Anda mungkin ingin mengatur ulang kata sandi pengguna utama untuk instans DB. Anda dapat melakukan tindakan ini dengan mengubah instans RDS.

# Masalah keamanan Amazon RDS

Untuk menghindari masalah keamanan, jangan pernah menggunakan alamat email pengguna Akun AWS root dan kata sandi Anda untuk akun pengguna. Praktik terbaik adalah menggunakan pengguna root Anda untuk membuat pengguna dan menetapkannya ke akun pengguna DB. Anda juga dapat menggunakan pengguna root Anda untuk membuat akun pengguna lain, jika perlu.

Untuk informasi tentang membuat pengguna, lihat Membuat pengguna IAM di Akun AWS. Untuk informasi tentang membuat pengguna AWS IAM Identity Center, lihat Mengelola identitas di Pusat Identitas IAM.

## Pesan kesalahan “gagal mengambil atribut akun, fungsi konsol tertentu mungkin terganggu.”

Kesalahan ini dapat muncul karena beberapa alasan. Penyebabnya mungkin karena akun Anda kehilangan izin, atau akun Anda belum disiapkan dengan benar. Untuk akun baru, Anda mungkin belum menunggu akun hingga siap digunakan. Untuk akun lama, Anda mungkin tidak memiliki izin dalam kebijakan akses untuk melakukan tindakan tertentu, seperti membuat instans DB. Untuk memecahkan masalah ini, administrator perlu memberikan peran yang diperlukan ke akun Anda. Untuk informasi selengkapnya, lihat dokumentasi IAM.

## Mengatur ulang kata sandi pemilik instans DB

Jika Anda terkunci dari klaster DB, Anda dapat masuk sebagai pengguna utama. Kemudian, Anda dapat mengatur ulang kredensial untuk pengguna administratif atau peran lainnya. Jika Anda tidak dapat masuk sebagai pengguna utama, pemilik AWS akun dapat mengatur ulang kata sandi pengguna utama. Untuk detail tentang akun administratif atau peran yang mungkin perlu Anda atur ulang, lihat Hak akses akun pengguna master.

Anda dapat mengubah kata sandi instans DB dengan menggunakan konsol Amazon RDS, AWS CLI perintah modify-db-instance, atau dengan menggunakan operasi Modify DBInstance API. Untuk informasi selengkapnya tentang cara mengubah instans DB dalam klaster DB, lihat Memodifikasi instans DB dalam klaster DB.

## Penghentian atau boot ulang instans DB Amazon RDS

Penghentian instans DB dapat terjadi ketika instans DB di-boot ulang. Penghentian ini juga dapat terjadi saat instans DB diubah menjadi status yang mencegah akses ke instans tersebut, dan saat basis data di-boot ulang. Boot ulang dapat terjadi saat Anda melakukan boot ulang manual pada instans DB Anda. Boot ulang juga dapat terjadi saat Anda mengubah pengaturan instans DB yang memerlukan boot ulang sebelum dapat diberlakukan.

Boot ulang instans DB terjadi saat Anda mengubah pengaturan yang memerlukan boot ulang, atau ketika Anda secara manual menyebabkan boot ulang. Boot ulang dapat segera terjadi jika Anda mengubah pengaturan dan meminta perubahan segera diberlakukan. Atau, boot ulang dapat terjadi selama jendela pemeliharaan instans DB.

Boot ulang instans DB segera terjadi ketika salah satu hal berikut terjadi:

- Anda mengubah periode retensi pencadangan untuk instans DB dari 0 ke nilai selain nol atau dari nilai selain nol ke 0. Anda mengatur Terapkan Segera ke `true`.
- Anda mengubah kelas instans DB, dan Terapkan Segera diatur menjadi `true`.

Boot ulang instans DB terjadi selama jendela pemeliharaan saat salah satu dari hal berikut terjadi:

- Anda mengubah periode retensi pencadangan untuk instans DB dari 0 ke nilai selain nol atau dari nilai selain nol ke 0, dan Terapkan Segera diatur ke `false`.
- Anda mengubah kelas instans DB, dan Terapkan Segera diatur menjadi `false`.

Ketika Anda mengubah parameter statis dalam grup parameter DB, perubahan tersebut tidak diberlakukan hingga instans DB yang dikaitkan dengan grup parameter di-boot ulang. Perubahan ini memerlukan boot ulang manual. Instans DB tidak di-boot ulang secara otomatis selama jendela pemeliharaan.

## Perubahan parameter DB Amazon RDS tidak diberlakukan

Dalam beberapa kasus, Anda mungkin mengubah parameter dalam grup parameter DB, tetapi tidak melihat perubahan akan diberlakukan. Jika demikian, Anda mungkin perlu melakukan boot ulang instans DB yang dikaitkan dengan grup parameter DB. Saat Anda mengubah parameter dinamis, perubahan akan langsung diberlakukan. Ketika Anda mengubah parameter statis, perubahan tersebut tidak diberlakukan hingga Anda melakukan boot ulang instans DB yang dikaitkan dengan grup parameter.

Anda dapat melakukan boot ulang pada instans DB menggunakan konsol RDS. Atau, Anda dapat secara eksplisit memanggil operasi API `RebootDBInstance`. Anda dapat mem-boot ulang tanpa failover jika instans DB ada dalam deployment Multi-AZ. Persyaratan untuk melakukan boot ulang pada instans DB terkait setelah perubahan parameter statis membantu memitigasi risiko kesalahan konfigurasi parameter yang memengaruhi panggilan API. Contohnya adalah memanggil `ModifyDBInstance` untuk mengubah kelas instans DB. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB di .

## Masalah memori yang dapat dikosongkan di Amazon Aurora

Memori yang dikosongkan adalah total Random Access Memory (RAM) pada instans DB yang dapat dibuat tersedia untuk mesin basis data. Ini adalah jumlah dari memori sistem operasi (OS) yang

kosong serta memori cache halaman dan buffer yang tersedia. Mesin basis data menggunakan sebagian besar memori pada host, tetapi proses OS juga menggunakan sebagian RAM. Memori yang saat ini dialokasikan ke mesin basis data atau digunakan oleh proses OS tidak termasuk dalam memori yang dapat dikosongkan. Ketika mesin basis data kehabisan memori, instans DB dapat menggunakan ruang sementara yang biasanya digunakan untuk buffering dan caching. Seperti disebutkan sebelumnya, ruang sementara ini termasuk dalam memori yang dapat dikosongkan.

Anda menggunakan `FreeableMemory` metrik di Amazon CloudWatch untuk memantau memori yang dapat dibebaskan. Untuk informasi selengkapnya, lihat Alat pemantauan untuk Aurora.

Jika instans DB Anda secara konsisten kehabisan memori yang dapat dikosongkan atau menggunakan ruang swap, pertimbangkan untuk meningkatkan ke kelas instans DB yang lebih besar. Untuk informasi selengkapnya, lihat Kelas instans Amazon Aurora DB.

Anda juga dapat mengubah pengaturan memori. Misalnya, pada Aurora MySQL , Anda dapat menyesuaikan ukuran parameter `innodb_buffer_pool_size`. Parameter ini diatur secara default ke 75 persen memori fisik. Untuk tips pemecahan masalah MySQL lainnya, lihat Bagaimana cara memecahkan masalah memori yang dapat dikosongkan rendah di basis data Amazon RDS for MySQL?

Untuk Aurora Serverless v2, `FreeableMemory` mewakili jumlah memori yang tidak terpakai yang tersedia saat Aurora Serverless v2 Instans DB diskalakan ke kapasitas maksimumnya. Anda mungkin memiliki instans yang diturunkan skalanya ke kapasitas yang relatif rendah, tetapi masih melaporkan nilai tinggi untuk `FreeableMemory`, karena instans dapat dinaikkan skalanya. Memori tersebut tidak tersedia saat ini, tetapi Anda bisa mendapatkannya jika Anda membutuhkannya.

Untuk setiap unit kapasitas Aurora (ACU) yang kapasitasnya saat ini di bawah kapasitas maksimum, `FreeableMemory` meningkat sekitar 2 GiB. Dengan demikian, metrik ini tidak mendekati nol sampai instans DB dinaikkan skalanya setinggi mungkin.

Jika metrik ini mendekati nilai `0`, instans DB telah dinaikkan skalanya setinggi mungkin. Skala ini mendekati batas memori yang tersedia. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Jika metrik ini mendekati nilai `0` pada instans DB pembaca, pertimbangkan untuk menambahkan instans DB pembaca lain ke klaster. Dengan begitu, bagian hanya baca dari beban kerja dapat tersebar di lebih banyak instans DB, sehingga mengurangi penggunaan memori pada setiap instans DB pembaca. Untuk informasi selengkapnya, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Untuk Aurora Serverless v1, Anda dapat mengubah rentang kapasitas untuk menggunakan lebih banyak ACUs. Untuk informasi selengkapnya, lihat Memodifikasi Aurora Serverless v1 Klaster DB.

# Masalah replikasi Amazon Aurora MySQL

Beberapa masalah replikasi MySQL juga berlaku untuk Aurora MySQL. Anda dapat mendiagnosis dan memperbaiki masalah tersebut.

Topik


- Mendiagnosis dan mengatasi jeda di antara replika baca
- Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL
- Kesalahan replikasi terhenti
- Replikasi replika baca gagal menginisialisasi struktur metadata


## Mendiagnosis dan mengatasi jeda di antara replika baca

Setelah Anda membuat replika baca MySQL dan replikanya tersedia, Amazon RDS pertama-tama mereplikasi perubahan yang dibuat ke instans DB sumber sejak operasi replika baca dimulai. Selama fase ini, waktu jeda replikasi untuk replika baca lebih besar dari 0. Anda dapat memantau jeda waktu ini di Amazon CloudWatch dengan melihat metrik Amazon RDS.

Metrik `AuroraBinlogReplicaLag` melaporkan nilai kolom `Seconds_Behind_Master` dari perintah `SHOW REPLICA STATUS` MySQL. Untuk informasi selengkapnya, lihat SHOW REPLICA STATUS Statement di dokumentasi MySQL.

Saat metrik `AuroraBinlogReplicaLag` mencapai 0, replika telah menyamai instans DB sumber. Jika metrik `AuroraBinlogReplicaLag` menunjukkan -1, replikasi mungkin tidak aktif. Untuk memecahkan masalah kesalahan replikasi, lihat Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL. Nilai `AuroraBinlogReplicaLag` sebesar -1 juga dapat berarti bahwa nilai `Seconds_Behind_Master` tidak dapat ditentukan atau `NULL`.

> **Note**
>
> Versi sebelumnya dari Aurora MySQL menggunakan `SHOW SLAVE STATUS`, bukan `SHOW REPLICA STATUS`. Jika Anda menggunakan Aurora MySQL versi 1 atau 2, gunakan `SHOW SLAVE STATUS`. Gunakan `SHOW REPLICA STATUS` untuk Aurora MySQL versi 3 dan yang lebih tinggi.

Metrik `AuroraBinlogReplicaLag` menunjukkan -1 saat penghentian jaringan atau saat patch diterapkan selama jendela pemeliharaan. Dalam kasus ini, tunggu konektivitas jaringan hingga dipulihkan atau tunggu jendela pemeliharaan berakhir sebelum Anda memeriksa metrik `AuroraBinlogReplicaLag` lagi.

Teknologi replikasi baca MySQL bersifat asinkron. Oleh karena itu, Anda dapat sesekali mengharapkan peningkatan bagi metrik `BinLogDiskUsage` pada instans DB sumber dan bagi metrik `AuroraBinlogReplicaLag` pada replika baca. Misalnya, pertimbangkan situasi saat volume operasi tulis tinggi ke instans DB sumber terjadi secara paralel. Pada saat yang sama, operasi tulis ke replika baca akan diserialkan menggunakan thread I/O tunggal. Situasi tersebut dapat menyebabkan jeda antara instans sumber dan replika baca.

Untuk informasi selengkapnya tentang replika baca dan MySQL, lihat Replication implementation details dalam dokumentasi MySQL.

Anda dapat mengurangi lag antara pembaruan ke instans DB sumber dan pembaruan berikutnya ke replika baca dengan melakukan hal berikut:

- Atur kelas instans DB dari replika baca agar memiliki ukuran penyimpanan yang sebanding dengan ukuran dari instans DB sumber.
- Pastikan kompatibilitas pengaturan parameter di grup parameter DB yang digunakan oleh instans DB sumber dan replika baca. Untuk informasi selengkapnya dan contoh, lihat diskusi tentang parameter `max_allowed_packet` di bagian berikutnya.
- Nonaktifkan cache kueri. Untuk tabel yang sering diubah, menggunakan cache kueri dapat meningkatkan lag replika karena cache terkunci dan sering disegarkan. Dalam kasus ini, Anda mungkin akan melihat lebih sedikit lag replika jika menonaktifkan cache kueri. Anda dapat menonaktifkan cache kueri dengan mengatur `query_cache_type parameter` ke 0 dalam grup parameter DB untuk instans DB. Untuk informasi selengkapnya tentang cache kueri, lihat Konfigurasi cache kueri.
- Hangatkan kumpulan buffer pada replika baca untuk InnoDB for MySQL. Misalnya, anggaplah Anda memiliki sejumlah kecil tabel yang sering diperbarui dan Anda menggunakan skema tabel InnoDB atau XtraDB. Dalam kasus ini, dump tabel tersebut pada replika baca. Dengan melakukan hal ini, Anda akan menyebabkan mesin basis data memindai barisan tabel tersebut dari disk, lalu menyimpannya di dalam kumpulan buffer. Pendekatan ini dapat mengurangi jeda replika. Berikut ini menunjukkan contoh.

  Untuk Linux, macOS, atau Unix:

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

Untuk Windows:

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

## Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL

Amazon RDS memantau status replikasi replika baca Anda. RDS memperbarui bidang Status Replikasi instans replika baca menjadi `Error` jika replikasi berhenti karena alasan apa pun. Anda dapat meninjau detail kesalahan terkait yang dilontarkan oleh mesin MySQL dengan melihat kolom Kesalahan Replikasi. Peristiwa yang menunjukkan status replika baca juga dihasilkan, termasuk RDS-EVENT-0045, RDS-EVENT-0046, dan RDS-EVENT-0057. Untuk informasi selengkapnya tentang peristiwa dan berlangganan peristiwa, lihat Bekerja dengan pemberitahuan RDS acara Amazon. Jika pesan kesalahan MySQL muncul, periksa kesalahannya di MySQL error message documentation.

Situasi umum yang dapat menyebabkan kesalahan replikasi mencakup hal-hal berikut:

- Nilai parameter `max_allowed_packet` untuk replika baca lebih kecil dari parameter `max_allowed_packet` untuk instans DB sumber.

  Parameter `max_allowed_packet` adalah parameter kustom yang dapat Anda atur di grup parameter DB. Parameter `max_allowed_packet` digunakan untuk menentukan ukuran maksimum bahasa manipulasi data (DML) yang dapat dijalankan di basis data. Dalam beberapa kasus, nilai `max_allowed_packet` untuk instans DB sumber mungkin lebih besar dari nilai `max_allowed_packet` untuk replika baca. Jika demikian, proses replikasi dapat menimbulkan kesalahan dan menghentikan replikasi. Kesalahan yang paling umum adalah `packet bigger`

`than 'max_allowed_packet' bytes`. Anda dapat memperbaiki kesalahan ini dengan mengatur agar replika sumber dan baca menggunakan grup parameter DB yang sama dengan nilai parameter `max_allowed_packet`.

- Menulis ke tabel di replika baca. Jika Anda membuat indeks pada replika baca, parameter `read_only` harus diatur ke 0 untuk membuat indeks. Jika Anda menulis ke tabel di replika baca, tindakan ini dapat memecah replikasi.
- Gunakan mesin penyimpanan nontransaksional seperti MyISAM. Replika baca membutuhkan mesin penyimpanan transaksional. Replikasi hanya didukung untuk mesin penyimpanan berikut: InnoDB for MySQL atau MariaDB.

  Anda dapat mengonversi tabel MyISAM ke InnoDB dengan perintah berikut:

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- Gunakan kueri nondeterministik yang tidak aman seperti `SYSDATE()`. Untuk informasi selengkapnya, lihat Determination of safe and unsafe statements in binary logging di dokumentasi MySQL.

Langkah-langkah berikut dapat membantu mengatasi kesalahan replikasi Anda:

- Jika Anda mengalami kesalahan logis dan dapat melewatkan kesalahan tersebut dengan aman, ikuti langkah-langkah yang dijelaskan dalam Melewati kesalahan replika saat ini. Instans DB Aurora MySQL Anda harus menjalankan versi yang mencakup prosedur `mysql_rds_skip_repl_error`. Untuk informasi selengkapnya, lihat mysql_rds_skip_repl_error.
- Jika Anda mengalami masalah posisi log biner (binlog), Anda dapat mengubah posisi tayangan ulang replika. Anda melakukannya dengan perintah `mysql.rds_next_master_log` untuk Aurora MySQL versi 1 dan 2. Anda melakukannya dengan perintah `mysql.rds_next_source_log` untuk Aurora MySQL versi 3 dan yang lebih tinggi. Instans DB Aurora MySQL Anda harus menjalankan versi yang mendukung perintah ini untuk mengubah posisi tayangan putar ulang replika. Untuk informasi versi, lihat mysql_rds_next_master_log.
- Jika Anda mengalami masalah kinerja sementara karena beban DHTML yang tinggi, Anda dapat mengatur `innodb_flush_log_at_trx_commit` parameter ke 2 di grup parameter DB pada replika baca. Dengan melakukan hal ini, Anda dapat membantu replika baca mengejar, meskipun tindakan ini akan mengurangi atomisitas, konsistensi, isolasi, dan daya tahan (ACID) untuk sementara.
- Anda dapat menghapus replika baca dan membuat instans menggunakan pengidentifikasi instans DB yang sama. Dengan cara ini, titik akhir tetap sama dengan replika baca lama Anda.

Jika kesalahan replikasi diperbaiki, Status Replikasi berubah menjadi mereplikasi. Untuk informasi selengkapnya, lihat Memecahkan masalah replika baca MySQL.

## Kesalahan replikasi terhenti

Ketika memanggil perintah `mysql.rds_skip_repl_error`, Anda mungkin menerima pesan kesalahan yang menyatakan bahwa replikasi mati atau dinonaktifkan.

Pesan kesalahan ini muncul karena replikasi dihentikan dan tidak dapat dimulai ulang.

Jika Anda perlu melewati sejumlah besar kesalahan, lag replikasi dapat meningkat hingga melampaui periode retensi default untuk file log biner. Dalam hal ini, Anda mungkin mengalami kesalahan fatal karena file log biner dibersihkan sebelum diputar ulang pada replika. Penghapusan ini menyebabkan replikasi berhenti, dan Anda tidak dapat lagi memanggil perintah `mysql.rds_skip_repl_error` untuk melewati kesalahan replikasi.

Anda dapat memitigasi masalah ini dengan meningkatkan jumlah jam penyimpanan file log biner pada sumber replikasi Anda. Setelah meningkatkan waktu retensi binlog, Anda dapat memulai ulang replikasi dan memanggil perintah `mysql.rds_skip_repl_error` sesuai kebutuhan.

Untuk mengatur waktu retensi binlog, gunakan prosedur mysql_rds_set_configuration. Tentukan parameter konfigurasi 'jam retensi binlog' sekaligus jumlah jam untuk menyimpan file binlog di klaster DB, hingga 2160 (90 hari). Default untuk Aurora MySQL adalah 24 (1 hari). Contoh berikut menetapkan periode penyimpanan file binlog menjadi 48 jam.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

## Replikasi replika baca gagal menginisialisasi struktur metadata

Ketika Anda mencoba untuk memulai replikasi, Anda menerima pesan galat berikut:

```
Read Replica Replication Error - SQLError: 13124, reason: Replica failed to initialize
 applier metadata structure from the repository
```

Kesalahan ini terjadi ketika ada masalah dengan struktur metadata replika. Untuk memperbaiki struktur metadata, Anda harus membuat replika baru.

Untuk mencegah hal ini terjadi di masa depan, lakukan salah satu tindakan berikut:

- Jika memungkinkan, nonaktifkan multi-threading pada replika Anda. Dimulai dengan MySQL 8.0.27, multi-threading diaktifkan secara default.
- Jika Anda perlu menggunakan multi-threading pada replika Anda, maka kami sarankan Anda menggunakan replikasi berbasis GTID. Untuk informasi selengkapnya, lihat Menggunakan replikasi GTID berbasis.

# Referensi Amazon RDS API

Selain AWS Management Console dan AWS Command Line Interface (AWS CLI), Amazon RDS juga menyediakan API. Anda dapat menggunakan API untuk mengotomatiskan tugas untuk mengelola instans DB dan objek lain di Amazon RDS.

- Untuk daftar abjad operasi API, lihat Tindakan.
- Untuk daftar abjad jenis data, lihat Jenis data.
- Untuk daftar parameter kueri umum, lihat Parameter umum.
- Untuk deskripsi kode kesalahan, lihat Kesalahan umum.

Untuk informasi selengkapnya tentang AWS CLI, lihat Referensi AWS Command Line Interface untuk Amazon RDS.

Topik

- Menggunakan API Kueri
- Pemecahan masalah aplikasi di Aurora

# Menggunakan API Kueri

Bagian berikut membahas secara singkat parameter dan autentikasi permintaan yang digunakan untuk API Kueri.

Untuk informasi umum tentang cara kerja API Kueri, lihat Permintaan kueri dalam Referensi API Amazon EC2.

## Parameter Kueri

Permintaan berbasis Kueri HTTP adalah permintaan yang menggunakan parameter HTTP verb GET atau POST dan parameter Kueri yang bernama `Action`.

Setiap permintaan Kueri harus menyertakan beberapa parameter umum untuk menangani autentikasi dan pemilihan tindakan.

Beberapa operasi mengambil daftar parameter. Daftar ini ditentukan menggunakan notasi `param.`*n*. Nilai *n* adalah bilangan bulat yang dimulai dari 1.

Untuk informasi tentang Wilayah dan titik akhir Amazon RDS, buka ke Amazon Relational Database Service (RDS) di bagian Wilayah dan Titik Akhir dalam Referensi Umum Amazon Web.

## Autentikasi permintaan Kueri

Anda hanya dapat mengirim permintaan Kueri melalui HTTPS, dan Anda harus menyertakan signature di setiap permintaan Kueri. Anda harus menggunakan AWS signature versi 4 atau signature versi 2. Untuk informasi selengkapnya, lihat Proses penandatanganan Signature Versi 4 dan Proses penandatanganan Signature versi 2.

# Pemecahan masalah aplikasi di Aurora

Amazon RDS memberikan penjelasan tentang kesalahan spesifik dan deskriptif untuk membantu Anda memecahkan masalah saat berinteraksi dengan API Amazon RDS.

Topik

- Mengambil kesalahan
- Tips pemecahan masalah

Untuk informasi tentang pemecahan masalah untuk instans DB Amazon RDS, lihat Pemecahan Masalah untuk Amazon Aurora.

## Mengambil kesalahan

Biasanya, Anda ingin aplikasi memeriksa apakah permintaan membuat kesalahan sebelum Anda menghabiskan waktu untuk memproses hasil. Cara termudah untuk mengetahui jika terjadi kesalahan adalah dengan mencari simpul `Error` di dalam respons dari API Amazon RDS.

Sintaksis XPath memberikan cara sederhana untuk mencari keberadaan simpul `Error`. Sintaksis ini juga memberikan cara yang relatif mudah untuk mengambil kode kesalahan dan pesan. Cuplikan kode berikut menggunakan modul Perl dan XML::XPath untuk menentukan jika kesalahan terjadi selama permintaan. Jika terjadi kesalahan, kode akan mencetak pesan dan kode kesalahan pertama dalam responsnya.

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
```

```
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

## Tips pemecahan masalah

Sebaiknya lakukan proses berikut untuk mendiagnosis dan menyelesaikan masalah dengan API Amazon RDS:

- Verifikasi bahwa Amazon RDS beroperasi secara normal di Wilayah AWS yang Anda targetkan dengan memeriksa http://status.aws.amazon.com.
- Periksa struktur permintaan Anda.

  Setiap operasi Amazon RDS memiliki halaman referensi di Referensi API Amazon RDS. Periksa ulang bahwa Anda menggunakan parameter dengan benar. Untuk mengetahui kemungkinan kesalahan, lihat contoh permintaan atau skenario pengguna untuk melihat apakah contoh tersebut melakukan operasi serupa.
- Periksa AWS re:Post.

  Amazon RDS memiliki komunitas pengembangan tempat Anda dapat mencari solusi untuk masalah yang dialami orang lain selama ini. Untuk melihat topik, buka AWS re:Post.

# Riwayat dokumen

Versi API saat ini: 2014-10-31

Tabel berikut menjelaskan perubahan penting pada Panduan Pengguna Amazon Aurora. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan feed RSS. Untuk informasi tentang Amazon Relational Database Service (Amazon RDS), lihat Panduan Pengguna Amazon Relational Database Service.

> **Note**
>
> Sebelum 31 Agustus 2018, Amazon Aurora didokumentasikan dalam Panduan Pengguna Amazon Relational Database Service. Untuk riwayat dokumen Aurora sebelumnya, lihat Riwayat dokumen dalam Panduan Pengguna Amazon Relational Database Service.

Anda dapat memfilter fitur Amazon Aurora baru di halaman Hal-Hal Baru dalam Basis Data. Untuk Produk, pilih Amazon Aurora. Kemudian, cari menggunakan kata kunci seperti **`global database`** atau **`Serverless`**.

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| Integrasi nol-ETL dengan lakehouse umumnya tersedia Amazon SageMaker | Integrasi nol-ETL membuat data transaksional tersedia dalam Amazon SageMaker hitungan detik setelah ditulis ke instance Aurora MySQL DB. Untuk informasi selengkapnya, lihat Bekerja dengan integrasi Aurora Zero-ETL. | Juni 30, 2025 |
| End-of-life Informasi untuk Performance Insights | AWS telah mengumumkan end-of-life tanggal Performance Insights: 30 November 2025. Setelah tanggal ini, Amazon RDS tidak akan lagi mendukung pengalama | 30 Mei 2025 |

| | | |
|---|---|---|
| | n konsol Performance Insights, periode retensi fleksibel, dan harga terkait. Sebaiknya upgrade cluster DB Anda ke Database Insights sebelum tanggal ini. Untuk informasi selengkapnya, lihat Memantau pemuatan DB dengan Performance Insights di Amazon Aurora. | |
| Aurora PostgreSQL Limitless Database mendukung Wawasan Database CloudWatch | Anda sekarang dapat menggunakan CloudWatch Database Insights untuk memantau Amazon Aurora PostgreSQL Limitless Database dalam skala besar. Untuk informasi selengkapnya, lihat Memantau Database Tanpa Batas Amazon Aurora PostgreSQL dengan Wawasan Database. CloudWatch | 22 Mei 2025 |
| Melihat tanggal dukungan untuk versi utama mesin open source | Anda sekarang dapat melihat informasi tentang tanggal mulai dan berakhir untuk versi utama mesin open source dalam dukungan standar RDS dan RDS Extended Support dengan menggunakan AWS CLI dan RDS API. Untuk informasi selengkapnya, lihat Melihat tanggal dukungan untuk versi engine di Amazon RDS Extended Support. | Mei 20, 2025 |

| | | |
|---|---|---|
| Integrasi Aurora PostgreSQL Zero-ETL mendukung Wilayah tambahan | Integrasi Aurora PostgreSQL Zero-ETL sekarang tersedia di Wilayah tambahan. Untuk informasi selengkapnya, lihat Wilayah yang Didukung dan engine Aurora DB untuk integrasi Nol-ETL dengan Amazon Redshift. | Februari 17, 2025 |

| | | |
|---|---|---|
| End-of-life informasi untuk Aurora Serverless v1 | AWS telah mengumumkan end-of-life tanggal untukAurora Serverless v1: 31 Maret 2025. Kami sangat menyarankan untuk memutakhirkan cluster Aurora Serverless v1 DB apa pun Aurora Serverless v2 sebelum tanggal tersebut. Upgrade dapat melibatkan perubahan dalam nomor versi utama dari mesin database. Oleh karena itu, penting untuk merencanakan, menguji, dan menerapkan peralihan ini sebelum tanggal. end-of-life Mulai 8 Januari 2025, pelanggan tidak akan lagi dapat membuat Aurora Serverless v1 cluster atau instance baru dengan atau AWS Management Console CLI. Untuk informasi tentang prosedur pemutakhiran dari Aurora Serverless v1 keAurora Serverless v2, lihat Memutakhirkan dari Aurora Serverless v1 kluster ke Aurora Serverless v2. | Desember 6, 2024 |

| | | |
|---|---|---|
| Aurora mendukung CloudWatch Database Insights | Anda sekarang dapat menggunakan CloudWatch Database Insights untuk memantau kluster Amazon Aurora DB Anda dalam skala besar. Untuk informasi selengkapnya, lihat Memantau database Amazon Aurora dengan CloudWatch Database Insights. | Desember 1, 2024 |
| Amazon Aurora mendukung kelas instans db.r8g | Anda sekarang dapat menggunakan kelas instance db.r8g untuk membuat cluster Aurora DB. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora. | November 21, 2024 |
| Aurora Serverless v2mendukung jeda otomatis dan melanjutkan | Aurora Serverless v2Instans DB sekarang dapat menskalakan hingga nol unit kapasitas Aurora ACUs () setelah periode tidak aktif. Untuk mengaktifkan fitur ini, Anda menetapkan nilai kapasitas minimum ke 0 ACUs. Untuk informasi selengkapnya, lihat Jeda otomatis dan lanjutkan untuk Aurora Serverless v2. | November 20, 2024 |

| | | |
|---|---|---|
| Amazon Aurora mendukung database migrasi otomatis EC2 | Anda dapat menggunakan konsol RDS untuk memigrasikan EC2 database ke Aurora. Aurora menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan basis data sumber Anda. EC2 AWS DMS memungkinkan Anda untuk memigrasikan database relasional ke dalam database Anda. AWS Cloud Untuk informasi selengkapnya, lihat Migrasi otomatis EC2 database ke Amazon Aurora menggunakan. AWS Database Migration Service | November 20, 2024 |
| AWS Advanced NodeJS Wrapper umumnya tersedia | Amazon Web Services (AWS) Advanced NodeJS Wrapper melengkapi dan memperluas fungsionalitas driver NodeJS yang ada. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Aurora dengan driver. AWS | November 19, 2024 |
| Amazon Aurora mendukung kelas instans db.r7i | Anda sekarang dapat menggunakan kelas instance db.r7i untuk membuat cluster Aurora DB. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora. | November 18, 2024 |

| | | |
|---|---|---|
| RDS Data API untuk Aurora MySQL mendukung semua tampilan Performance Insights | Anda sekarang dapat menggunakan host Top atau tampilan aplikasi Top untuk Performance Insights untuk memantau operasi SQL yang dilakukan menggunakan RDS Data API untuk Aurora MySQL. Untuk informasi tentang dukungan Performance Insights, lihat Memantau kueri API Data RDS dengan Performance Insights. | Oktober 31, 2024 |
| Database Tanpa Batas Aurora PostgreSQL | Amazon Aurora PostgreSQL Limitless Database adalah kemampuan penskalaan horizontal (sharding) baru, otomatis, dan horizontal dari Amazon Aurora. Aurora PostgreSQL Limitless Database memungkinkan Anda menskalakan melampaui batas Aurora yang ada untuk throughput dan penyimpanan tulis dengan mendistribusikan beban kerja database ke beberapa instance penulis Aurora, sambil mempertahankan kemampuan untuk menggunakannya sebagai database tunggal. Untuk informasi selengkapnya, lihat Amazon Aurora PostgreSQL Limitless Database. | Oktober 31, 2024 |

| | | |
|---|---|---|
| Amazon Aurora MySQL versi 2 akhir dukungan standar | Aurora MySQL versi 2 mencapai akhir dukungan standar pada 31 Oktober 2024. Untuk informasi selengkapnya, lihat Mempersiapkan dukungan standar versi 2 Edisi yang kompatibel dengan Amazon Aurora MySQL versi 2. | Oktober 31, 2024 |
| Amazon Aurora mendukung peningkatan OS tingkat cluster | Anda sekarang dapat menerapkan upgrade sistem operasi Aurora di tingkat cluster DB. Peningkatan bergulir secara otomatis menerapkan peningkatan ke beberapa instans DB pembaca sekaligus, sehingga menjaga ketersediaan baca. Untuk informasi selengkapnya, lihat Pembaruan sistem operasi untuk klaster Aurora DB. | Oktober 30, 2024 |

| | | |
|---|---|---|
| Kluster global Aurora mendukung penandaan | Anda sekarang dapat menambahkan tag ke cluster global Aurora. Kluster global Aurora adalah sumber daya induk yang berisi cluster DB primer dan sekunder dalam database global Aurora. Untuk informasi umum tentang penandaan, lihat Menandai sumber daya Aurora dan Amazon RDS. Untuk informasi tentang penandaan dengan Aurora Global Database, lihat Menandai sumber daya Aurora Global Database. | Oktober 24, 2024 |
| Titik akhir penulis Basis Data Global Aurora | Setiap Database Global Aurora sekarang dilengkapi dengan titik akhir penulis yang secara otomatis diperbarui oleh Aurora untuk merutekan permintaan ke instance penulis saat ini dari cluster DB primer setelah pengaliha n dan operasi failover Aurora Global Database. Untuk mempelajari lebih lanjut tentang menggunakan titik akhir penulis bersama dengan peralihan dan failover Aurora Global Database, lihat Menggunakan switchover atau failover di Amazon Aurora Global Database. | Oktober 22, 2024 |

| | | |
|---|---|---|
| Integrasi Aurora PostgreSQL Zero-ETL dengan Amazon Redshift umumnya tersedia | Integrasi nol-ETL membuat data transaksional tersedia di Amazon Redshift dalam hitungan detik setelah ditulis ke cluster DB sumber PostgreSQL Aurora. Fitur ini sekarang tersedia secara umum. Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift. | Oktober 15, 2024 |
| Aurora Serverless v2mendukung 256 ACUs | Anda sekarang dapat membuat cluster Aurora Serverless v2 DB dengan kapasitas maksimum 256 unit ACUs kapasitas Aurora (). Untuk informasi lebih lanjut, lihat Aurora Serverless v2kapasitas. | Oktober 3, 2024 |

| | | |
|---|---|---|
| Amazon Aurora mendukung Console-to-Code | Anda sekarang dapat menggunakan Console-to-Code untuk menghasilkan kode dari tindakan yang Anda lakukan di konsol RDS. Kode yang dihasilkan dapat membantu Anda menulis kode untuk mengotomatiskan penggunaan AWS layanan lain. Untuk informasi selengkapnya, lihat Menggunakan Console-to-Code untuk menghasilkan kode untuk tindakan konsol Aurora Anda. | Oktober 3, 2024 |
| RDS Data API mendukung Performance Insights | Anda sekarang dapat menggunakan Performance Insights untuk memantau operasi SQL yang dilakukan menggunakan RDS Data API. Untuk informasi tentang dukungan Performance Insights, lihat Memantau kueri API Data RDS dengan Performance Insights. | September 26, 2024 |

| | | |
|---|---|---|
| RDS Data API untuk Aurora Serverless v2 dan disediakan tersedia untuk Aurora MySQL | RDS Data API sekarang tersedia untuk klaster MySQL Aurora yang menggunakan atau menyediakan instance. Aurora Serverless v2 Untuk informasi penggunaan untuk RDS Data API, lihat Menggunakan RDS Data API. Untuk informasi tentang versi MySQL Aurora dan dukungan AWS Wilayah untuk RDS Data API, lihat Data API dengan Aurora MySQL Serverless v2 dan disediakan. | September 26, 2024 |
| Amazon Aurora tersedia di Wilayah Asia Pasifik (Malaysia) | Amazon Aurora sekarang tersedia di Wilayah Asia Pasifik (Malaysia). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | Agustus 22, 2024 |
| Perbarui ke kebijakan yang ada | Amazon RDS menghapus `sns:Publish` izin `AmazonRDSBetaServiceRolePolicy` dari peran `AWSServiceRoleForRDSBeta` terkait layanan. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | Agustus 7, 2024 |

| | | |
|---|---|---|
| Perbarui ke kebijakan yang ada | Amazon RDS menghapus `sns:Publish` izin `AmazonRDSPreviewServiceRolePolicy` dari peran `AWSServiceRoleForRDSPreview` terkait layanan. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | Agustus 7, 2024 |
| Aurora Serverless v2mendukung jeda otomatis () scale-to-zero | Aurora Serverless v2Instans DB sekarang dapat menskalakan hingga nol unit kapasitas Aurora ACUs () setelah periode tidak aktif. Untuk mengaktifkan fitur ini, Anda menetapkan nilai kapasitas minimum ke 0. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2. | Juli 25, 2024 |
| AWS ODBC Driver untuk MySQL umumnya tersedia | Amazon Web Services (AWS) ODBC Driver untuk MySQL adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora MySQL. Untuk informasi selengkapnya, lihat Menghubungkan ke Aurora MySQL dengan Driver ODBC Amazon Web Services ()AWS untuk MySQL. | Juli 18, 2024 |

| | | |
|---|---|---|
| RDS Data API untuk Aurora Serverless v2 tersedia di lebih banyak Wilayah | RDS Data API sekarang tersedia untuk Aurora PostgreSQL di beberapa Wilayah tambahan. AWS Untuk informasi tentang dukungan Region untuk RDS Data API, lihat Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan. | 9 Juli 2024 |
| AWS Driver Python umumnya tersedia | Driver Python Amazon Web Services (AWS) dirancang sebagai pembungkus Python tingkat lanjut. Pembungkus ini melengkapi dan memperlua s fungsionalitas driver Psycopg open-source. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Aurora dengan driver. AWS | 23 Mei 2024 |
| Integrasi nol-ETL tersedia di Wilayah Tiongkok | Integrasi nol-ETL sekarang tersedia di Wilayah AWS Tiongkok (Beijing) dan China (Ningxia). Untuk informasi selengkapnya, lihat Integrasi nol-ETL dengan Amazon Redshift. | 21 Mei 2024 |

| | | |
|---|---|---|
| Proxy RDS tersedia di lebih banyak Wilayah | RDS Proxy sekarang tersedia di wilayah Asia Pasifik (Hyderabad), Asia Pasifik (Melbourne), Timur Tengah (UEA), Israel (Tel Aviv), Kanada Barat (Calgary), dan Eropa (Zurich). Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 21 Mei 2024 |
| Amazon RDS mendukung akses halus untuk Performance Insights | Sekarang Anda dapat mengizinkan atau menolak akses ke dimensi individual di Performance Insights. Akses berbutir halus ini dapat digunakan untuk`GetResourceMetrics` ,`DescribeDimensionKeys` , dan tindakan. `GetDimensionKeyDetails` Untuk informasi selengkapnya, lihat https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PerfInsights.access-control.dimensionAccess-policy.html. | 21 Mei 2024 |

| | | |
|---|---|---|
| Dukungan Amazon RDS yang Diperluas | Membuat atau memulihkan Aurora MySQL versi 2 atau 3, atau database Aurora PostgreSQL versi 11 sekarang secara otomatis mendaftar kan database tersebut ke Amazon RDS Extended Support sehingga aplikasi Anda yang ada terus berfungsi sebagaimana adanya. Anda dapat memilih keluar dari RDS Extended Support untuk menghindari biaya setelah Aurora berakhir tanggal dukungan standar untuk mesin database Anda. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Amazon RDS yang Diperluas. | Maret 21, 2024 |
| Pemfilteran data untuk integrasi nol-ETL | Amazon RDS mendukung pemfilteran data di database dan tingkat tabel untuk integrasi nol-ETL dengan Amazon Redshift. Untuk informasi selengkapnya, lihat Pemfilteran data untuk integrasi Aurora Zero-ETL dengan Amazon Redshift. | Maret 20, 2024 |

| | | |
|---|---|---|
| Integrasi Aurora MySQL dengan Amazon Bedrock | Anda sekarang dapat mengintegrasikan database Amazon Aurora MySQL dengan Amazon Bedrock untuk memberi daya pada aplikasi AI generatif. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora MySQL. | 8 Maret 2024 |
| Kebijakan AWS terkelola baru | Amazon RDS menambahkan kebijakan terkelola baru yang diberi nama `AmazonRDSCustomInstanceProfileRolePolicy` untuk memungkinkan RDS Custom melakukan tindakan otomatisasi dan tugas manajemen database melalui profil EC2 instans. Untuk informasi selengkapnya, lihat Pembaruan Amazon RDS terhadap kebijakan terkelola AWS. | Februari 27, 2024 |
| Dukungan Amazon RDS untuk AWS Secrets Manager Wilayah Israel (Tel Aviv) | Amazon RDS mendukung Secrets Manager di Wilayah Israel (Tel Aviv). Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan Amazon RDS dan AWS Secrets Manager. | Februari 21, 2024 |

| | | |
|---|---|---|
| Dukungan Amazon RDS yang Diperluas | Amazon RDS sekarang secara otomatis mengaktif kan Amazon RDS Extended Support ketika Aurora MySQL dan Aurora PostgreSQL versi mesin utama di cluster DB Anda dan cluster global mencapai akhir Aurora dari tanggal dukungan standar. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Amazon RDS yang Diperluas. | Februari 15, 2024 |
| Aurora PostgreSQL 16.1 mendukung Babelfish untuk Aurora PostgreSQL 4.0.0 | Aurora PostgreSQL 16.1 mendukung Babelfish 4.0.0. Untuk daftar fitur baru, lihat 16.1. Untuk daftar fitur yang didukung oleh setiap rilis Babelfish, lihat Fungsionalitas yang didukung di Babelfish menurut versi. Untuk informasi penggunaan, lihat Bekerja dengan Babelfish for Aurora PostgreSQL. | Januari 31, 2024 |
| Perbarui ke Sertifikat CA default | Sertifikat CA default diatur `rds-ca-rsa2048-g1` ke . Untuk informasi selengkapnya, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke kluster DB. | Januari 26, 2024 |

| | | |
|---|---|---|
| RDS Proxy tersedia di Wilayah Eropa (Spanyol) | RDS Proxy sekarang tersedia di wilayah Eropa (Spanyol) . Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 8 Januari 2024 |
| RDS Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan | Anda sekarang dapat menggunakan RDS Data API dengan Aurora PostgreSQL Serverless v2 dan cluster DB yang disediakan. Dengan RDS Data API, Anda dapat mengakses cluster Aurora Anda melalui titik akhir HTTP yang aman dan menjalankan pernyataan SQL tanpa menggunakan driver database atau mengelola koneksi. Untuk informasi selengkapnya, lihat Menggunakan API Data RDS. | 21 Desember 2023 |
| Integrasi Aurora PostgreSQL dengan Amazon Bedrock | Anda sekarang dapat mengintegrasikan database Amazon Aurora PostgreSQL dengan Amazon Bedrock untuk memberi daya pada aplikasi AI generatif. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora PostgreSQL. | 21 Desember 2023 |

| | | |
|---|---|---|
| Amazon Aurora tersedia di Wilayah Kanada Barat (Calgary) | Amazon Aurora sekarang tersedia di Wilayah Kanada Barat (Calgary). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 20 Desember 2023 |
| Amazon RDS mendukung melihat dan menanggapi rekomendasi | Rekomendasi Amazon Aurora sekarang mencakup rekomendasi reaktif berbasis proaktif dan pembelajaran mesin berbasis ambang batas. Untuk informasi selengkapnya, lihat Rekomendasi untuk Amazon Aurora. | Desember 19, 2023 |
| Integrasi nol-ETL Aurora PostgreSQL dengan Amazon Redshift (pratinjau) | Anda kini dapat membuat integrasi nol-ETL dengan Amazon Redshift menggunakan klaster DB sumber Aurora PostgreSQL. Untuk rilis pratinjau, Anda harus membuat semua integrasi di Lingkungan Pratinjau Database Amazon RDS, di Timur AS (Ohio) (us-east-2). Wilayah AWS Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift. | 28 November 2023 |

| | | |
|---|---|---|
| Amazon Aurora PostgreSQL mendukung penerusan tulis basis data global | Anda kini dapat mengaktifkan fungsi penerusan tulis pada klaster sekunder dalam basis data global yang berbasis Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menggunakan fungsi penerusan tulis dalam basis data global Aurora PostgreSQL. | 9 November 2023 |
| Dukungan Aurora PostgreSQL untuk Optimized Reads | Anda dapat menyelesaikan pemrosesan kueri yang lebih cepat untuk Aurora PostgreSQL dengan Aurora Optimized Reads. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads. | 8 November 2023 |

| | | |
|---|---|---|
| Amazon RDS mengekspor metrik Performance Insights ke Amazon CloudWatch | Performance Insights memungkinkan Anda mengekspor dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom ke Amazon. CloudWatch Dasbor metrik yang diekspor tersedia untuk dilihat di konsol. CloudWatch Anda juga dapat mengekspor widget metrik Performance Insights yang dipilih dan melihat data metrik di konsol. CloudWatch Untuk informasi selengkapnya, lihat Mengekspor metrik Performance Insights ke. CloudWatch | 8 November 2023 |
| Integrasi nol-ETL Aurora MySQL dengan ketersediaan umum Amazon Redshift | Integrasi nol-ETL dengan Amazon Redshift kini tersedia secara umum untuk Aurora MySQL. Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift. | 7 November 2023 |
| Dukungan Aurora PostgreSQL untuk Penerapan RDS Blue/Green | Anda sekarang dapat membuat blue/green penyebaran dari cluster DB PostgreSQL Aurora. Untuk informasi selengkapnya, lihat Menggunakan Blue/Green Penerapan Amazon RDS untuk pembaruan database. | 26 Oktober 2023 |

| | | |
|---|---|---|
| Aurora MySQL mendukung enkripsi sisi server dengan (SSE-KMS) AWS KMS keys | Di Aurora MySQL versi 3.05 dan lebih tinggi, Anda dapat menggunakan SSE-KMS, termasuk Kunci yang dikelola AWS dan kunci yang dikelola pelanggan, untuk enkripsi data sisi server yang Anda muat atau simpan ke Amazon S3. Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3 dan Menyimpan data dari klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3. | 25 Oktober 2023 |
| Pengoptimalan Aurora MySQL mengurangi waktu mulai ulang basis data | Di Aurora MySQL versi 3.05 dan yang lebih tinggi, kami telah memperken alkan pengoptimalan yang mengurangi waktu mulai ulang basis data. Pengoptim alan ini menghasilkan periode nonaktif hingga 65% lebih sedikit daripada tanpa pengoptimalan, dan lebih sedikit gangguan pada beban kerja basis data Anda, setelah memulai ulang. Untuk informasi selengkapnya, lihat Pengoptimalan untuk mengurangi waktu mulai ulang basis data. | 25 Oktober 2023 |

| | | |
|---|---|---|
| Memperbarui ke kebijakan AWS terkelola | Kebijakan terkelola `AmazonRDSPerformanceInsightsReadOnly` dan `AmazonRDSPerformanceInsightsFullAccess` sekarang menyertakan `Sid` (ID pernyataan) sebagai pengidentifikasi dalam pernyataan kebijakan. Untuk informasi selengkapnya, lihat Pembaruan Amazon RDS terhadap kebijakan terkelola AWS. | 23 Oktober 2023 |
| Amazon RDS menerbitkan metrik penghitung Performance Insights ke Amazon CloudWatch | Fungsi matematika metrik DB_PERF_INSIGHTS di konsol CloudWatch memungkinkan Anda melakukan kueri Amazon RDS untuk metrik penghitung Performance Insights. Untuk informasi selengkapnya, lihat Membuat CloudWatch alarm untuk memantau Amazon Aurora. | 20 September 2023 |
| Amazon Aurora mendukung point-in-time pemulihan (PITR) dengan AWS Backup | Anda sekarang dapat mengelola pencadangan otomatis (berkelanjutan) Aurora AWS Backup dan mengembalikan ke waktu yang ditentukan dari mereka. Untuk informasi selengkapnya, lihat Memulihkan klaster DB ke waktu tertentu menggunakan AWS Backup. | 7 September 2023 |

| | | |
|---|---|---|
| Dukungan Amazon RDS yang Diperluas | Amazon Aurora mengumumkan kemampuan mendatang untuk terus menjalankan versi mesin utama Aurora MySQL dan Aurora PostgreSQL di instans DB Anda setelah akhir tanggal dukungan standar Aurora. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Amazon RDS yang Diperluas. | 1 September 2023 |
| Amazon Aurora MySQL memperluas dukungan untuk Percona XtraBackup | Anda sekarang dapat melakukan migrasi fisik basis data MySQL 8.0 ke klaster DB Aurora MySQL versi 3. Untuk informasi selengkapnya, lihat Migrasi fisik dari MySQL menggunakan XtraBackup Percona dan Amazon S3. | 24 Agustus 2023 |

| | | |
|---|---|---|
| Basis data global Aurora mendukung failover basis data global | Basis data global Aurora kini mendukung failover global terkelola, yang memungkinkan Anda untuk lebih mudah pulih dari bencana alam regional atau penonaktifan total tingkat layanan. Untuk mempelajari lebih lanjut fitur ini, lihat Melakukan failover terkelola untuk basis data global Aurora. Fitur yang sebelumnya disebut "failover terencana yang dikelola" sekarang disebut "switchover". Untuk informasi tentang switchover, lihat Melakukan switchover untuk basis data global Amazon Aurora. | 21 Agustus 2023 |
| Memperbarui ke izin kebijakan AWS terkelola | Kebijakan terkelola `AmazonRDSFullAccess` memiliki izin baru yang memungkinkan Anda membuat, melihat, dan menghapus laporan analisis performa untuk jangka waktu tertentu. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | 17 Agustus 2023 |

| | | |
|---|---|---|
| Memperbarui ke izin kebijakan AWS terkelola | Penambahan izin baru ke kebijakan terkelola `AmazonRDSPerformanceInsightsReadOnly` dan penambahan kebijakan terkelola baru `AmazonRDSPerformanceInsightsFullAccess` memungkinkan Anda membuat laporan analisis beban DB untuk jangka waktu tertentu. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | 16 Agustus 2023 |
| Amazon RDS mendukung analisis periode waktu muat DB dengan Wawasan Performa | Wawasan Performa memungkinkan Anda membuat laporan analisis performa untuk periode waktu tertentu. Laporan ini memberikan wawasan yang diidentifikasi dan rekomendasi untuk mengatasi masalah performa. Untuk informasi selengkapnya, lihat Menganalisis beban DB untuk jangka waktu tertentu. | 16 Agustus 2023 |
| Amazon Aurora mendukung fitur retensi cadangan otomatis untuk klaster DB | Anda kini dapat mempertahankan cadangan otomatis untuk klaster Aurora yang dihapus dan memulihkannya ke titik waktu tertentu. Untuk informasi selengkapnya, lihat Mempertahankan cadangan otomatis. | 4 Agustus 2023 |

| | | |
|---|---|---|
| Amazon Aurora tersedia di Wilayah Israel (Tel Aviv) | Amazon Aurora kini tersedia di Wilayah Israel (Tel Aviv). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 1 Agustus 2023 |
| Amazon Aurora MySQL mendukung penerusan tulis lokal (dalam klaster) | Anda kini dapat meneruskan operasi tulis dari instans DB pembaca ke instans DB penulis dalam klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Menggunakan fungsi penerusan tulis lokal di klaster DB Amazon Aurora MySQL. | 31 Juli 2023 |
| Amazon Aurora mendukung Aurora Serverless v2 tambahan Wilayah AWS | Anda sekarang dapat membuat cluster Aurora Serverless v2 DB di Asia Pasifik (Melbourne) Wilayah AWS. Untuk informasi selengkapnya tentang Aurora Serverless v2, lihat Menggunakan Aurora Serverless v2. | 28 Juni 2023 |
| Amazon Aurora memperkenalkan integrasi nol-ETL dengan Amazon Redshift (pratinjau) | Integrasi nol-ETL memberikan solusi yang terkelola sepenuhnya untuk menyediakan data transaksional di Amazon Redshift dalam hitungan detik setelah ditulis ke klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift. | 28 Juni 2023 |

| | | |
|---|---|---|
| Amazon RDS menyediakan gabungan Performance Insights CloudWatch dan tampilan metrik di dasbor Performance Insights | Amazon RDS kini menyediakan tampilan konsolidasi Performance Insights CloudWatch dan metrik di dasbor Performance Insights. Untuk informasi selengkapnya, lihat Melihat metrik gabungan dengan dasbor Performance Insights. | 24 Mei 2023 |
| Amazon Aurora mendukung kelas instans db.r7g | Anda kini dapat menggunakan kelas instans db.r7g untuk membuat klaster DB Aurora. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora. | 11 Mei 2023 |
| Amazon Aurora mendukung konfigurasi penyimpanan klaster DB baru | DenganAurora I/O-Optimized, Anda hanya membayar untuk penggunaan dan penyimpanan cluster DB Anda, tanpa biaya tambahan untuk I/O operasi baca dan tulis. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora. | 11 Mei 2023 |

| | | |
|---|---|---|
| Amazon Aurora mendukung tambahan Aurora Serverless v2Wilayah AWS | Anda sekarang dapat membuat cluster Aurora Serverless v2 DB sebagai berikut Wilayah AWS: Asia Pasifik (Hyderabad), Eropa (Spanyol) Eropa (Zurich), dan Timur Tengah (UEA). Untuk informasi selengkap nya tentang Aurora Serverless v2, lihat Menggunakan Aurora Serverless v2. | 4 Mei 2023 |
| Aurora Serverless v1 mendukung konversi ke yang disediakan | Anda dapat mengonversi klaster DB Aurora Serverles s v1 langsung ke klaster DB yang disediakan. Untuk informasi selengkapnya, lihat Mengonversi klaster DB Aurora Serverless v1 ke yang disediakan. | 27 April 2023 |
| Aurora Serverless v1 mendukung Amazon Aurora PostgreSQL versi 13 | Anda kini dapat membuat klaster DB Aurora Serverless v1 yang menjalankan Aurora PostgreSQL versi 13. Untuk informasi selengkapnya, lihat Aurora Serverless v1. | 27 April 2023 |
| Dukungan Amazon Aurora untuk AWS Secrets Manager di Wilayah Tiongkok | Amazon Aurora mendukung Secrets Manager di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan Amazon Aurora dan AWS Secrets Manager. | 20 April 2023 |

| | | |
|---|---|---|
| Amazon Aurora mendukung peristiwa penerbitan dengan tag ke pelanggan topik | Pemberitahuan acara Amazon Aurora yang dikirim ke Amazon Simple Notification Service (Amazon SNS) atau EventBridge Amazon sekarang berisi tag peristiwa di badan pesan. Tag ini menyediakan data sumber daya yang dipengaruhi oleh peristiwa layanan. Untuk informasi selengkapnya, lihat Tag dan atribut notifikasi peristiwa Amazon RDS. | 17 April 2023 |
| Memperbarui ke izin peran yang terkait dengan layanan IAM | `AmazonRDSReadOnlyAccess` Kebijakan `AmazonRDSFullAccess` dan sekarang memberikan izin tambahan untuk memungkinkan tampilan temuan Amazon DevOps Guru di konsol RDS. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | 30 Maret 2023 |
| Amazon Aurora mendukung basis data global di Wilayah Asia Pasifik (Melbourne) | Anda kini dapat membuat basis data global Aurora di Wilayah Asia Pasifik (Melbourne). Untuk informasi tentang basis data global Aurora, lihat Menggunakan basis data global Amazon Aurora. | 22 Maret 2023 |

| | | |
|---|---|---|
| Memperbarui ke izin kebijakan AWS terkelola | `AmazonRDSReadOnlyAccess` Kebijakan `AmazonRDSFullAccess` dan sekarang memberikan izin tambahan ke Amazon. CloudWatch Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | 16 Maret 2023 |
| Proksi RDS tersedia di Wilayah Tiongkok | Proksi RDS sekarang tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 15 Maret 2023 |
| Amazon Aurora mendukung Aurora Serverless v2 di Wilayah Tiongkok | Aurora Serverless v2 kini tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkapnya, lihat Aurora Serverless v2. | 15 Maret 2023 |
| Proksi RDS tersedia di Wilayah Asia Pasifik (Jakarta) | Sekarang Proksi RDS tersedia di Wilayah Asia Pasifik (Jakarta). Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 8 Maret 2023 |

| | | |
|---|---|---|
| Amazon Aurora MySQL mendukung autentikasi Kerberos | Anda kini dapat menggunakan autentikasi Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster DB Aurora MySQL Anda. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos untuk Aurora MySQL. | 8 Maret 2023 |
| Amazon Aurora mendukung database global sebagai tambahan Wilayah AWS | Anda kini dapat membuat basis data global Aurora di Wilayah berikut: Afrika (Cape Town), Asia Pasifik (Hong Kong), Asia Pasifik (Hyderabad), Asia Pasifik (Jakarta), Eropa (Milan), Eropa (Spanyol), Eropa (Zürich), Timur Tengah (Bahrain), dan Timur Tengah (UAE). Untuk informasi tentang basis data global Aurora, lihat Menggunakan basis data global Amazon Aurora. | 6 Maret 2023 |

| | | |
|---|---|---|
| Amazon Aurora mendukung penyalinan snapshot cluster DB sebagai tambahan Wilayah AWS | Anda kini dapat menyalin snapshot klaster DB di Wilayah berikut: Afrika (Cape Town), Asia Pasifik (Hong Kong), Asia Pasifik (Hyderabad), Asia Pasifik (Jakarta), Asia Pasifik (Melbourne), Eropa (Milan), Eropa (Spanyol), Eropa (Zürich), Timur Tengah (Bahrain), dan Timur Tengah (UAE). Untuk informasi tentang penyalinan snapshot klaster DB, lihat Menyalin snapshot klaster DB. | 6 Maret 2023 |
| Amazon DevOps Guru untuk RDS mendukung wawasan proaktif | Amazon DevOps Guru for RDS menerbitkan wawasan proaktif dengan rekomendasi untuk membantu Anda mengatasi masalah di database Aurora Anda sebelum diprediksi terjadi. Untuk informasi selengkapnya, lihat Cara Kerja DevOps Guru for RDS. | 28 Februari 2023 |
| Amazon Aurora MySQL versi 1 tidak digunakan lagi | Aurora MySQL versi 1 (kompatibel dengan MySQL 5.6) tidak digunakan lagi. Untuk informasi selengkapnya, lihat Sisa waktu ketersediaan Amazon Aurora versi utama. | 28 Februari 2023 |

| | | |
|---|---|---|
| Aurora Serverless v1 mendukung pengaturan periode pemeliharaan klaster DB | Anda kini dapat mengatur periode pemeliharaan untuk klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menyesuaikan periode pemeliharaan klaster DB yang diinginkan. | 27 Februari 2023 |
| Amazon Aurora mendukung Aliran Aktivitas Basis Data di Wilayah Asia Pasifik (Hyderabad), Eropa (Spanyol), dan Timur Tengah (UEA). | Untuk informasi selengkapnya, lihat Aliran Aktivitas Basis Data. | 27 Januari 2023 |
| Amazon Aurora tersedia di Wilayah Asia Pasifik (Melbourne) | Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Melbourne). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 23 Januari 2023 |
| Tentukan otoritas sertifikat (CA) selama pembuatan klaster DB | Anda kini dapat menentukan CA mana yang akan digunakan untuk sertifikat server klaster DB selama pembuatan klaster DB. Untuk informasi selengkapnya, lihat Otoritas sertifikat. | 5 Januari 2023 |

| | | |
|---|---|---|
| Dukungan Aurora MySQL 3.* untuk pelacakan mundur | Aurora MySQL versi 3.* kini menawarkan sebuah cara cepat untuk pulih dari kesalahan pengguna, seperti meletakkan tabel yang salah atau menghapus baris yang salah. Pelacakan mundur memungkinkan Anda memindahkan basis data ke titik waktu sebelumnya tanpa harus memulihkan dari cadangan, dan dapat diselesai kan dalam hitungan detik, bahkan untuk basis data yang besar. Untuk perincian, lihat Melacak mundur klaster DB Aurora. | 4 Januari 2023 |
| Gunakan Amazon RDS Blue/ Green Deployment yang tersedia dalam tambahan Wilayah AWS | Fitur Blue/Green Deployment sekarang tersedia di Wilayah Tiongkok (Beijing) dan China (Ningxia). Untuk informasi selengkapnya, lihat Menggunakan Blue/Green Penerapan Amazon RDS untuk pembaruan database. | 22 Desember 2022 |
| Pembaruan terhadap izin peran yang ditautkan layanan IAM | RDSServiceRolePolicy Kebijakan Amazon sekarang memberikan izin tambahan untuk. AWS Secrets Manager Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola. | 22 Desember 2022 |

| | | |
|---|---|---|
| Amazon Aurora terintegrasi dengan AWS Secrets Manager manajemen kata sandi | Aurora dapat mengelola kata sandi pengguna master untuk klaster DB di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan Amazon Aurora dan AWS Secrets Manager. | 22 Desember 2022 |
| Amazon Aurora mendukung tambahan Aurora Serverless v2Wilayah AWS | Aurora Serverless v2 kini tersedia di Wilayah Afrika (Cape Town) dan Eropa (Milan). Untuk informasi selengkapnya, lihat Aurora Serverless v2. | 21 Desember 2022 |
| Aurora PostgreSQL mendukung Proksi RDS dengan PostgreSQL 14 | Anda kini dapat membuat Proksi RDS dengan klaster DB Aurora PostgreSQL 14. Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 13 Desember 2022 |
| Amazon Aurora memberi tahu Anda tentang anomali terbaru yang terdeteksi oleh Amazon Guru DevOps | Halaman detail basis data konsol memberi tahu Anda tentang anomali saat ini dan anomali yang terjadi dalam 24 jam terakhir. Untuk informasi selengkapnya, lihat Cara Kerja DevOps Guru for RDS. | 13 Desember 2022 |

| | | |
|---|---|---|
| Proksi Amazon RDS mendukung basis data global | Anda kini dapat menggunakan Proksi RDS dengan basis data global Aurora. Untuk informasi selengkapnya, lihat Menggunakan Proksi RDS dengan basis data global Aurora. | 7 Desember 2022 |
| Klaster DB Aurora PostgreSQL mendukung Ekstensi Bahasa Tepercaya untuk PostgreSQL | Ekstensi Bahasa Tepercaya untuk PostgreSQL adalah kit pengembangan sumber terbuka yang memungkinkan Anda membuat ekstensi PostgreSQL beperforma tinggi dan menjalankannya dengan aman di klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL. | 30 November 2022 |

| | | |
|---|---|---|
| Amazon GuardDuty RDS Protection memantau ancaman akses | Saat Anda mengaktifkan Perlindungan GuardDuty RDS, GuardDuty mengkonsumsi peristiwa login RDS dari database Aurora Anda, memantau peristiwa ini, dan memprofilkannya untuk potensi ancaman orang dalam atau aktor eksternal. Ketika GuardDuty RDS Protection mendeteksi potensi ancaman, GuardDuty menghasilkan temuan baru dengan rincian tentang database yang berpotensi dikompromikan. Untuk informasi selengkapnya, lihat Memantau ancaman dengan Perlindungan GuardDuty RDS. | 30 November 2022 |
| Menggunakan Amazon RDS Blue/Green Deployment untuk pembaruan database | Anda dapat membuat perubahan pada klaster DB di lingkungan staging dan menguji perubahan tersebut tanpa memengaruhi klaster DB produksi Anda. Jika sudah siap, Anda dapat mempromosikan lingkungan staging menjadi lingkungan produksi baru, dengan periode nonaktif yang minimal. Untuk informasi selengkapnya, lihat Menggunakan Blue/Green Penerapan Amazon RDS untuk pembaruan database. | 27 November 2022 |

| | | |
|---|---|---|
| Amazon Aurora tersedia di Wilayah Asia Pasifik (Hyderabad) | Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Hyderabad). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 22 November 2022 |
| Amazon Aurora tersedia di Wilayah Eropa (Spanyol) | Amazon Aurora kini tersedia di Wilayah Eropa (Spanyol) . Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 16 November 2022 |
| Amazon Aurora tersedia di Wilayah Eropa (Zürich) | Amazon Aurora kini tersedia di Wilayah Eropa (Zürich). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 9 November 2022 |
| Amazon Aurora mendukung pengeksporan data ke Amazon S3 dari klaster DB | Anda kini dapat mengekspor data klaster Aurora langsung ke S3, tanpa harus membuat snapshot terlebih dahulu. Untuk informasi selengkapnya, lihat Mengekspor data klaster DB ke Amazon S3. | 27 Oktober 2022 |
| Amazon Aurora MySQL mendukung ekspor yang lebih cepat ke Amazon S3 | Anda kini dapat menyaksikan performa yang maksimal 10x lebih cepat untuk mengekspor data snapshot klaster DB ke S3 untuk klaster Aurora MySQL yang kompatibel dengan MySQL 5.7 dan 8.0. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3. | 20 Oktober 2022 |

| | | |
|---|---|---|
| Amazon Aurora mendukung pengaturan konektivitas secara otomatis antara cluster Aurora DB dan instans EC2 | Anda dapat menggunakan AWS Management Console untuk mengatur konektivitas antara cluster Aurora DB yang ada dan sebuah EC2 instance. Untuk informasi selengkapnya, lihat Menghubungkan EC2 instans dan klaster Aurora DB secara otomatis. | 14 Oktober 2022 |
| AWS Driver JDBC untuk PostgreSQL umumnya tersedia | Driver AWS JDBC untuk PostgreSQL adalah driver klien yang dirancang untuk Aurora PostgreSQL. Driver AWS JDBC untuk PostgreSQL sekarang tersedia secara umum. Untuk informasi selengkapnya, lihat Menghubungkan dengan Driver AWS JDBC untuk PostgreSQL. | 6 Oktober 2022 |
| Amazon Aurora mendukung peningkatan di tempat untuk Aurora MySQL yang kompatibel dengan MySQL 5.7 | Anda dapat melakukan peningkatan di tempat untuk mengubah klaster Aurora MySQL yang kompatibel dengan MySQL 5.7 menjadi klaster Aurora MySQL yang kompatibel dengan MySQL 8.0. Untuk informasi selengkapnya, lihat Meningkatkan dari Aurora MySQL 2.x ke 3.x. | 26 September 2022 |

| | | |
|---|---|---|
| Wawasan Performa menampilkan 25 kueri SQL teratas | Di dasbor Wawasan Performa, tab SQL Teratas menampilkan 25 kueri SQL yang paling berkontribusi pada beban DB. Untuk informasi selengkapnya, lihat Gambaran umum tab SQL Teratas. | 13 September 2022 |
| Aurora MySQL mendukung kelas instans DB yang baru | Anda kini dapat menggunakan kelas instans db.r6i untuk klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Kelas instans DB. | 13 September 2022 |
| Amazon Aurora tersedia di Wilayah Timur Tengah (UAE) | Amazon Aurora kini tersedia di Wilayah Timur Tengah (UAE). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 30 Agustus 2022 |
| Amazon Aurora mendukung pengaturan konektivitas secara otomatis dengan sebuah instans EC2 | Saat membuat klaster Aurora DB, Anda dapat menggunakannya AWS Management Console untuk mengatur konektivitas antara instans Amazon Elastic Compute Cloud dan cluster DB baru. Untuk informasi selengkapnya, lihat Mengkonfigurasi konektivitas jaringan otomatis dengan sebuah EC2 instance. | 22 Agustus 2022 |

| | | |
|---|---|---|
| Amazon Aurora mendukung mode tumpukan ganda | Klaster DB kini dapat berjalan dalam mode tumpukan ganda. Dalam mode dual-stack, sumber daya dapat berkomunikasi dengan cluster DB over IPv4, IPv6, atau keduanya. Untuk informasi selengkapnya, lihat Pengalamatan IP Amazon Aurora. | 17 Agustus 2022 |
| Amazon Aurora mendukung peningkatan di tempat untuk Aurora Serverless v1 yang kompatibel dengan PostgreSQL | Anda dapat melakukan peningkatan di tempat untuk klaster Aurora Serverless v1 yang kompatibel dengan PostgreSQL 10 untuk mengubah klaster yang ada menjadi klaster Aurora Serverless v1 yang kompatibel dengan PostgreSQL 11. Untuk prosedur peningkatan di tempat, lihat Memodifikasi klaster DB Aurora Serverless v1. | 8 Agustus 2022 |
| Wawasan Performa mendukung Wilayah Asia Pasifik (Jakarta) | Sebelumnya, Anda tidak dapat menggunakan Wawasan Performa di Wilayah Asia Pasifik (Jakarta). Pembatasan ini telah dihapus. Untuk informasi selengkapnya, lihat Dukungan Wilayah AWS untuk Wawasan Performa. | 21 Juli 2022 |

| | | |
|---|---|---|
| Amazon Aurora mendukung kelas instans DB yang baru | Anda kini dapat menggunakan kelas instans DB db.r6i untuk klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Kelas instans DB. | 14 Juli 2022 |
| Wawasan Performa RDS mendukung periode retensi tambahan | Sebelumnya, Wawasan Performa hanya menawarkan dua periode retensi: 7 hari (default) atau 2 tahun (731 hari). Sekarang, jika Anda perlu mempertahankan data performa Anda selama lebih dari 7 hari, Anda dapat menentukan 1–24 bulan. Untuk informasi selengkapnya, lihat Harga dan retensi data untuk Wawasan Performa. | 1 Juli 2022 |
| Amazon Aurora mendukung peningkatan di tempat untuk Aurora Serverless v1 yang kompatibel dengan MySQL | Anda dapat melakukan peningkatan di tempat untuk klaster Aurora Serverless v1 yang kompatibel dengan MySQL 5.6 untuk mengubah klaster yang ada menjadi klaster Aurora Serverless v1 yang kompatibel dengan MySQL 5.7. Untuk prosedur peningkatan di tempat, lihat Memodifikasi klaster DB Aurora Serverless v1. | 16 Juni 2022 |

| | | |
|---|---|---|
| Aurora mendukung menyalakan Amazon DevOps Guru di konsol RDS | Anda dapat mengaktifkan cakupan DevOps Guru untuk database Aurora Anda dari dalam konsol RDS. Untuk informasi selengkapnya, lihat Menyiapkan DevOps Guru untuk RDS. | 9 Juni 2022 |
| Amazon Aurora mendukung publikasi peristiwa ke topik Amazon SNS yang terenkripsi | Amazon Aurora kini dapat memublikasikan peristiwa ke topik Amazon Simple Notification Service (Amazon SNS) yang mengaktifkan enkripsi di sisi server (SSE), untuk perlindungan tambahan terhadap peristiwa yang berisi data sensitif. Untuk informasi selengkapnya, lihat Berlangganan notifikasi peristiwa Amazon RDS. | 1 Juni 2022 |
| Amazon RDS menerbitkan metrik penggunaan ke Amazon CloudWatch | `AWS/Usage` Namespace di Amazon CloudWatch menyertakan metrik penggunaan tingkat akun untuk kuota layanan Amazon RDS Anda. Untuk informasi selengkapnya, lihat metrik CloudWatch penggunaan Amazon untuk Amazon Aurora. | 28 April 2022 |

| | | |
|---|---|---|
| Kumpulan hasil API Data dalam format JSON | Parameter opsional untuk fungsi `ExecuteStatement` menyebabkan hasil kueri ditetapkan untuk ditampilkan sebagai string dalam format JSON. Kumpulan hasil JSON mudah dan praktis untuk diubah menjadi struktur data dalam bahasa aplikasi Anda. Untuk informasi selengkapnya, lihat Memproses hasil kueri dalam format JSON. | 27 April 2022 |
| Amazon Aurora Serverless v2 kini tersedia secara umum | Amazon Aurora Serverless v2 tersedia secara umum untuk semua pengguna. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2. | 21 April 2022 |
| Aurora MySQL mendukung cipher suite yang dapat dikonfigurasi | Dengan Aurora MySQL, Anda kini dapat menggunakan cipher suite yang dapat dikonfigurasi untuk mengontrol enkripsi koneksi yang diterima server basis data Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL. | 15 April 2022 |

| | | |
|---|---|---|
| Aurora PostgreSQL mendukung Proksi RDS dengan PostgreSQL 13 | Anda kini dapat membuat Proksi RDS dengan klaster DB Aurora PostgreSQL 13. Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 4 April 2022 |
| Catatan Rilis untuk Aurora PostgreSQL | Sekarang ada panduan terpisah untuk catatan rilis Amazon Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Catatan rilis Aurora PostgreSQL. | 22 Maret 2022 |
| Catatan Rilis untuk Aurora MySQL | Sekarang ada panduan terpisah untuk catatan rilis Amazon Aurora MySQL. Untuk informasi selengkapnya, lihat Catatan rilis Aurora MySQL. | 22 Maret 2022 |
| Aurora PostgreSQL mendukung peningkatan versi multi-utama | Anda kini dapat melakukan peningkatan versi klaster DB Aurora PostgreSQL di beberapa versi utama. Untuk informasi selengkapnya, lihat Cara melakukan peningkatan versi utama. | 4 Maret 2022 |

| | | |
|---|---|---|
| Aurora PostgreSQL mendukung cipher suite yang dapat dikonfigurasi | Dengan Aurora PostgreSQL versi 11.8 dan yang lebih tinggi, Anda kini dapat menggunakan cipher suite yang dapat dikonfigurasi untuk mengontrol enkripsi koneksi yang diterima server basis data Anda. Untuk informasi tentang penggunaan cipher suite yang dapat dikonfigurasi dengan Aurora PostgreSQL, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL. | 4 Maret 2022 |
| AWS Driver JDBC untuk MySQL umumnya tersedia | Driver AWS JDBC untuk MySQL adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora MySQL. Driver AWS JDBC untuk MySQL sekarang tersedia secara umum. Untuk informasi selengkapnya, lihat Menghubungkan dengan Amazon Web Services JDBC Driver for MySQL. | 2 Maret 2022 |

| | | |
|---|---|---|
| Aurora PostgreSQL 13.5 mendukung Babelfish untuk Babelfish for Aurora PostgreSQL 1.1.0 | Aurora PostgreSQL 13.5 mendukung Babelfish 1.1.0. Untuk daftar fitur baru, lihat 13.5. Untuk daftar fitur yang didukung oleh setiap rilis Babelfish, lihat Fungsionalitas yang didukung di Babelfish menurut versi. Untuk informasi penggunaan, lihat Bekerja dengan Babelfish for Aurora PostgreSQL. | 28 Februari 2022 |
| Amazon Aurora mendukung Aliran Aktivitas Basis Data di Wilayah Asia Pasifik (Jakarta) | Untuk informasi selengkapnya, lihat Support Wilayah AWS untuk aliran aktivitas database. | 16 Februari 2022 |
| Wawasan Performa mendukung operasi API yang baru | Wawasan Performa kini mendukung operasi API berikut: `GetResourceMetadata` , `ListAvailableResourceDimensions` , dan `ListAvailableResourceMetrics` . Untuk informasi selengkapnya, lihat Mengambil metrik dengan API Wawasan Performa dalam panduan ini dan Referensi API Wawasan Performa Amazon RDS. | 12 Januari 2022 |

| | | |
|---|---|---|
| Proksi Amazon RDS mendukung peristiwa | Proxy RDS sekarang menghasilkan acara yang dapat Anda berlangganan dan lihat di CloudWatch Acara atau konfigurasikan untuk dikirim ke Amazon EventBridge. Untuk informasi selengkapnya, lihat Menggunakan peristiwa Proksi RDS. | 11 Januari 2022 |
| Proxy RDS tersedia dalam tambahan Wilayah AWS | Proksi RDS sekarang tersedia di Wilayah berikut: Afrika (Cape Town), Asia Pasifik (Hong Kong), Asia Pasifik (Osaka), Eropa (Milan), Eropa (Paris), Eropa (Stockholm), Timur Tengah (Bahrain), dan Amerika Selatan (Sao Paulo). Untuk informasi selengkap nya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 5 Januari 2022 |
| Amazon Aurora tersedia di Wilayah Asia Pasifik (Jakarta) | Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Jakarta) . Untuk informasi selengkap nya, lihat Wilayah dan Zona Ketersediaan. | 13 Desember 2021 |

| | | |
|---|---|---|
| DevOpsGuru untuk Amazon RDS memberikan wawasan dan rekomendasi terperinci untuk Amazon Aurora | DevOpsGuru untuk RDS menambang Performance Insights untuk data terkait kinerja. Dengan menggunakan data ini, layanan menganalisis performa instans DB Amazon Aurora dan dapat membantu Anda mengatasi masalah performa. Untuk mempelajari lebih lanjut, lihat Menganalisis anomali kinerja dengan DevOps Guru untuk RDS dalam panduan ini dan lihat Ikhtisar DevOps Guru untuk RDS di Panduan Pengguna Amazon DevOps Guru. | 1 Desember 2021 |
| Aurora PostgreSQL mendukung Proksi RDS dengan PostgreSQL 12 | Anda kini dapat membuat Proksi RDS dengan klaster basis data Aurora PostgreSQL 12. Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS. | 22 November 2021 |
| Aurora mendukung kelas instance AWS Graviton2 untuk Aliran Aktivitas Database | Anda dapat menggunakan aliran aktivitas basis data dengan kelas instans db.r6g untuk Aurora MySQL dan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Kelas instans DB yang didukung. | 3 November 2021 |

| | | |
|---|---|---|
| Dukungan Amazon Aurora untuk lintas akun AWS KMS keys | Anda dapat menggunakan kunci KMS dari yang berbeda Akun AWS untuk enkripsi saat mengekspor snapshot DB ke Amazon S3. Untuk informasi selengkapnya, lihat Mengekspor data snapshot DB ke Amazon S3. | 3 November 2021 |
| Amazon Aurora mendukung Babelfish for Aurora PostgreSQL | Babelfish for Aurora PostgreSQL memperlua s basis data Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL Anda dengan kemampuan menerima koneksi basis data dari klien Server SQL Microsoft. Untuk informasi selengkapnya, lihat Bekerja dengan Babelfish for Aurora PostgreSQL. | 28 Oktober 2021 |
| Aurora Serverless v1 mungkin memerlukan SSL untuk koneksi | Parameter klaster Aurora `force_ssl` untuk PostgreSQL dan `require_secure_transport` untuk MySQL kini didukung untuk Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1. | 26 Oktober 2021 |

| | | |
|---|---|---|
| Amazon Aurora mendukung Performance Insights sebagai tambahan Wilayah AWS | Wawasan Performa tersedia di Wilayah Timur Tengah (Bahrain), Afrika (Cape Town), Eropa (Milan), dan Asia Pasifik (Osaka). Untuk informasi selengkapnya, lihat Dukungan Wilayah AWS untuk Wawasan Performa. | 5 Oktober 2021 |
| Batas waktu penskalaan otomatis yang dapat dikonfigurasi untuk Aurora Serverless v1 | Anda dapat memilih durasi tunggu Aurora Serverless v1 sebelum menemukan titik penskalaan otomatis. Jika tidak ditemukan titik penskalaan otomatis selama periode tersebut, Aurora Serverless v1 membatalkan peristiwa penskalaan atau memaksa perubahan kapasitas, bergantung pada tindakan batas waktu yang Anda pilih. Untuk informasi selengkapnya, lihat Penskalaan otomatis untuk v1Aurora Serverless. | 10 September 2021 |

| | | |
|---|---|---|
| Aurora mendukung kelas instans X2g dan T4g | Aurora MySQL dan Aurora PostgreSQL sekarang dapat menggunakan kelas instans X2g dan T4g. Kelas instans yang dapat Anda gunakan bergantung pada versi Aurora MySQL atau Aurora PostgreSQL. Untuk informasi selengkapnya tentang jenis instans yang didukung, lihat Kelas instans DB. | 10 September 2021 |
| Amazon RDS mendukung Proksi RDS di VPC bersama | Anda kini dapat membuat Proksi RDS di cloud privat virtual (VPC) bersama. Untuk informasi selengkapnya tentang Proksi RDS, lihat "Mengelola Koneksi dengan Proksi Amazon RDS" dalam Panduan Pengguna Amazon RDS atau Panduan Pengguna Aurora. | 6 Agustus 2021 |
| Halaman kebijakan versi Aurora | Panduan Pengguna Amazon Aurora sekarang menyertakan bagian berisi informasi umum tentang versi Aurora dan kebijakan terkait. Untuk detailnya, lihat versi Amazon Aurora. | 14 Juli 2021 |

| | | |
|---|---|---|
| Kecualikan peristiwa API Data dari AWS CloudTrail jejak | Anda dapat mengecualikan peristiwa Data API dari CloudTrail jejak. Untuk informasi selengkapnya, lihat Mengecualikan peristiwa API Data dari AWS CloudTrail jejak. | 2 Juli 2021 |
| Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL mendukung ekstensi tambahan | Ekstensi yang baru didukung termasuk pg_bigm, pg_cron, pg_partman, dan pg_procta b. Untuk informasi selengkapnya, lihat Versi ekstensi untuk Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL. | 17 Juni 2021 |
| Mengklonakan klaster Aurora Serverless | Anda kini dapat membuat klaster klona yang Aurora Serverless. Untuk informasi selengkapnya tentang pengklonaan, lihat Mengklonakan volume untuk klaster DB Aurora. | 16 Juni 2021 |
| Basis data global Aurora tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia) | Anda kini dapat membuat basis data global Aurora di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora. | 19 Mei 2021 |

| | | |
|---|---|---|
| Dukungan FIPS 140-2 untuk API Data | Data API mendukung Federal Information Processing Standard Publication 140-2 (FIPS 140-2) untuk koneksi. SSL/TLS Untuk informasi selengkapnya, lihat Ketersediaan API Data. | 14 Mei 2021 |
| AWS Driver JDBC untuk PostgreSQL (pratinjau) | Driver AWS JDBC untuk PostgreSQL, sekarang tersedia dalam pratinjau , adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menghubungkan Driver JDBC Layanan Web Amazon untuk PostgreSQL (pratinjau). | 27 April 2021 |
| Data API tersedia dalam tambahan Wilayah AWS | API Data kini tersedia di Wilayah Asia Pasifik (Seoul) dan Kanada (Pusat). Untuk informasi selengkapnya, lihat Ketersediaan API Data. | 9 April 2021 |
| Amazon Aurora mendukung kelas instans DB Graviton2 | Anda kini dapat menggunakan kelas instans DB Graviton2 db.r6g.x untuk membuat klaster DB yang menjalankan MySQL atau PostgreSQL. Untuk informasi selengkapnya, lihat Jenis kelas instans DB. | 12 Maret 2021 |

| | | |
|---|---|---|
| Peningkatan titik akhir Proksi RDS | Anda dapat membuat titik akhir tambahan yang terkait dengan setiap Proksi RDS. Membuat titik akhir dalam VPC yang berbeda memungkin kan akses lintas VPC untuk proksi tersebut. Proksi untuk klaster Aurora MySQL juga dapat memiliki titik akhir hanya baca. Titik akhir pembaca ini terhubung ke instans DB pembaca di dalam klaster dan dapat meningkatkan skalabili tas dan ketersediaan baca untuk aplikasi intensif kueri. Untuk informasi selengkap nya tentang Proksi RDS, lihat “Mengelola Koneksi dengan Proksi Amazon RDS” dalam Panduan Pengguna Amazon RDS atau Panduan Pengguna Aurora. | 8 Maret 2021 |
| Amazon Aurora tersedia di Wilayah Asia Pasifik (Osaka) | Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Osaka). Untuk informasi selengkap nya, lihat Wilayah dan Zona Ketersediaan. | 1 Maret 2021 |
| Aurora PostgreSQL mendukung pengaktifan autentikasi IAM dan Kerberos pada klaster DB yang sama | Aurora PostgreSQL kini mendukung pengaktifan autentikasi IAM dan Kerberos pada klaster DB yang sama. Untuk informasi selengkapnya, lihat Autentikasi basis data dengan Amazon Aurora. | 24 Februari 2021 |

| | | |
|---|---|---|
| Basis data global Aurora kini mendukung failover terencana yang terkelola | Basis data global Aurora kini mendukung failover terencana yang terkelola sehingga Anda dapat lebih mudah mengubah Wilayah AWS utama dari basis data global Aurora Anda. Anda dapat menggunakan failover terencana yang terkelola dengan basis data global Aurora yang berkondisi baik saja. Untuk mempelajari lebih lanjut, lihat Pemulihan bencana dan basis data global Amazon Aurora. Untuk informasi referensi, lihat `FailoverGlobalCluster` dalam Referensi API Amazon RDS. | 11 Februari 2021 |
| API Data untuk Aurora Serverless kini mendukung lebih banyak jenis data | Dengan API Data untuk Aurora Serverless, Anda kini dapat menggunakan jenis data `UUID` dan `JSON` sebagai input ke basis data Anda. Termasuk dengan API Data untuk Aurora Serverless, Anda kini dapat menampilkan nilai jenis `LONG` dari basis data Anda sebagai nilai `STRING`. Untuk mempelajari lebih lanjut, lihat Memanggil API Data. Untuk informasi referensi tentang jenis data yang didukung, lihat `SqlParameter` dalam Referensi API Layanan Data Amazon RDS. | 2 Februari 2021 |

| | | |
|---|---|---|
| Aurora PostgreSQL mendukung peningkatan versi utama ke PostgreSQL 12 | Dengan Aurora PostgreSQL, Anda kini dapat meningkatkan mesin DB ke versi utama 12. Untuk informasi selengkapnya, lihat Meningkatkan mesin DB PostgreSQL untuk Aurora PostgreSQL. | 28 Januari 2021 |
| Aurora MySQL mendukung peningkatan di tempat | Anda dapat meningkatkan klaster Aurora MySQL 1.x ke Aurora MySQL 2.x, yang mempertahankan instans DB, titik akhir, dan sebagainya dari klaster asli. Teknik peningkatan di tempat ini menghindari ketidaknyamanan penyiapan klaster baru dengan memulihkan snapshot. Hal ini juga menghindari overhead untuk penyalinan semua data tabel ke klaster baru. Untuk informasi selengkapnya, lihat Meningkatkan versi utama klaster DB Aurora MySQL dari 1.x ke 2.x. | 11 Januari 2021 |
| AWS Driver JDBC untuk MySQL (pratinjau) | Driver AWS JDBC untuk MySQL, sekarang tersedia dalam pratinjau, adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora MySQL. Untuk informasi selengkapnya, lihat Menghubungkan Driver JDBC Layanan Web Amazon untuk MySQL (pratinjau). | 7 Januari 2021 |

| | | |
|---|---|---|
| Aurora mendukung aliran aktivitas basis data pada klaster sekunder basis data global | Anda dapat memulai aliran aktivitas basis data pada klaster primer atau sekunder dari Aurora PostgreSQL atau Aurora MySQL. Untuk versi mesin yang didukung, lihat Batasan basis data global Aurora. | 22 Desember 2020 |
| Klaster multi-master dengan 4 instans DB | Jumlah maksimal instans DB pada klaster multi-master Aurora MySQL sekarang adalah empat. Sebelumnya, jumlah maksimalnya adalah dua instans DB. Untuk informasi selengkapnya, lihat Bekerja dengan Klaster Multi-Master Aurora. | 17 Desember 2020 |
| Aurora PostgreSQL mendukung fungsi AWS Lambda | Anda sekarang dapat memanggil AWS Lambda fungsi untuk cluster Aurora PostgreSQL DB Anda. Untuk informasi selengkapnya, lihat Menginvokasi fungsi Lambda dari klaster DB Aurora PostgreSQL. | 11 Desember 2020 |
| Amazon Aurora mendukung kelas instans DB Graviton2 dalam pratinjau | Anda kini dapat menggunakan kelas instans DB Graviton2 db.r6g.x dalam pratinjau untuk membuat klaster DB yang menjalankan MySQL atau PostgreSQL. Untuk informasi selengkapnya, lihat Jenis kelas instans DB. | 11 Desember 2020 |

| | | |
|---|---|---|
| Amazon Aurora Serverless v2 kini tersedia dalam pratinjau. | Amazon Aurora Serverless v2 tersedia dalam pratinjau. Untuk menggunakan Amazon Aurora Serverless v2, ajukan permohonan akses. Untuk informasi selengkapnya, lihat halaman Aurora Serverless v2. | 1 Desember 2020 |

| | | |
|---|---|---|
| Aurora PostgreSQL kini tersedia untuk Aurora Serverless di lebih banyak Wilayah AWS. | Aurora PostgreSQL kini tersedia untuk Aurora Serverless di lebih banyak Wilayah AWS. Anda sekarang dapat memilih untuk menjalankan Aurora PostgreSQL Serverless v1 penawaran Wilayah AWS yang samaAurora MySQL Serverless v1. Tambahan Wilayah AWS dengan Aurora Serverless dukungan termasuk AS Barat (Californ ia N.), Asia Pasifik (Singapur a) Asia Pasifik (Sydney) Asia Pasifik (Seoul) Asia Pasifik (Mumbai) Kanada (Tengah) Eropa (London) dan Eropa (Paris). Untuk daftar semua Wilayah dan mesin Aurora DB yang didukungAurora Serverless, lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Tanpa Server v1. API Data Amazon RDS untuk Aurora Serverless juga tersedia di Wilayah AWS yang sama ini. Untuk daftar semua Wilayah dengan dukungan untuk API DataAurora Serverless, lihat Data API dengan Aurora MySQL Serverless v1 | 24 November 2020 |

| | | |
|---|---|---|
| Wawasan Performa Amazon RDS memperkenalkan dimensi baru | Anda dapat mengelompokkan beban basis data berdasarkan grup dimensi untuk basis data, aplikasi (PostgreSQL), dan jenis sesi (PostgreSQL). Amazon RDS juga mendukung dimensi db.name, db.application.name (PostgreSQL), dan db.session_type.name (PostgreSQL). Untuk informasi selengkapnya, lihat Tabel beban teratas. | 24 November 2020 |
| Aurora Serverless mendukung Aurora PostgreSQL versi 10.12 | Aurora PostgreSQL untuk Aurora Serverless telah ditingkatkan ke Aurora PostgreSQL versi 10.12 di seluruh Wilayah AWS tempat Aurora PostgreSQL untuk Aurora Serverless didukung. Untuk informasi selengkapnya, lihat Daerah yang Didukung dan mesin Aurora DB untuk Aurora Tanpa Server v1. | 4 November 2020 |
| API Data kini mendukung otorisasi berbasis tag | API Data kini mendukung otorisasi berbasis tag. Jika Anda sudah melabeli sumber daya klaster RDS dengan tag, Anda dapat menggunakan tag ini dalam pernyataan kebijakan Anda untuk mengontrol akses melalui API Data. Untuk informasi selengkapnya, lihat Mengotorisasi akses ke API Data. | 27 Oktober 2020 |

| | | |
|---|---|---|
| Amazon Aurora memperluas dukungan untuk mengekspor snapshot ke Amazon S3 | Anda kini dapat mengekspor data snapshot DB ke Amazon S3 di semua Wilayah AWS komersial. Untuk informasi selengkapnya, lihat Mengekspor data snapshot DB ke Amazon S3. | 22 Oktober 2020 |
| Basis data global Aurora mendukung pengklonaan | Anda kini dapat menciptakan klona klaster DB primer dan sekunder dari basis data global Aurora Anda. Anda dapat melakukannya dengan menggunakan AWS Management Console dan memilih opsi menu Create clone. Anda juga dapat menggunakan AWS CLI dan menjalankan `restore-db-cluster-to-point-in-time` perintah dengan `--restore-type copy-on-write` opsi. Menggunakan AWS Management Console atau AWS CLI, Anda juga dapat mengkloning cluster DB dari database global Aurora Anda di seluruh akun. AWS Untuk informasi selengkapnya tentang pengklonaan, lihat Mengklonakan volume klaster DB Aurora. | 19 Oktober 2020 |

| | | |
|---|---|---|
| Amazon Aurora mendukung pengubahan ukuran dinamis untuk volume klaster | Mulai dari Aurora MySQL 1.23 dan 2.09, dan Aurora PostgreSQL 3.3.0 dan Aurora PostgreSQL 2.6.0, Aurora mengurangi ukuran volume klaster setelah Anda menghapus data melalui pengoperasian seperti `DROP TABLE`. Untuk memanfaatkan peningkatan ini, tingkatkan ke salah satu versi yang sesuai, bergantung pada mesin basis data yang digunakan klaster Anda. Untuk informasi tentang fitur ini dan cara memeriksa ruang penyimpanan yang masih ada dan telah digunakan untuk klaster Aurora, lihat Mengelola Performa dan Menskalakan Klaster DB Aurora. | 13 Oktober 2020 |
| Amazon Aurora mendukung ukuran volume hingga 128 TiB | Volume klaster Aurora yang baru dan lama kini bertambah ke ukuran maksimum 128 tebibyte (TiB). Untuk informasi selengkapnya, lihat Cara penyimpanan Aurora bertambah. | 22 September 2020 |

| | | |
|---|---|---|
| Aurora PostgreSQL mendukung kelas instans DB db.r5 dan db.t3 untuk Wilayah Tiongkok (Ningxia) | Anda kini dapat membuat klaster DB Aurora PostgreSQL di Wilayah Tiongkok (Ningxia) yang menggunakan kelas instans DB db.r5 dan db.t3. Untuk informasi selengkapnya, lihat Kelas instans DB. | 3 September 2020 |

| | | |
|---|---|---|
| Peningkatan kueri paralel Aurora | Mulai dengan Aurora MySQL 2.09 dan 1.23, Anda dapat memanfaatkan peningkatan pada fitur kueri paralel. Pembuatan klaster kueri paralel tidak lagi membutuhkan mode mesin khusus. Anda kini dapat mengaktifkan dan menonaktifkan kueri paralel menggunakan opsi konfigurasi `aurora_parallel_query` untuk klaster apa pun yang telah disediakan yang menjalankan versi Aurora MySQL yang kompatibel. Anda dapat meningkatkan klaster yang sudah ada ke versi Aurora MySQL yang kompatibel dan menggunakan kueri paralel, bukan membuat klaster baru serta mengimpor data ke klaster. Anda dapat menggunakan Wawasan Performa untuk klaster kueri paralel. Anda dapat menghentikan dan memulai klaster kueri paralel. Anda dapat membuat klaster kueri paralel Aurora yang kompatibel dengan MySQL 5.7. Kueri paralel berfungsi untuk tabel yang menggunakan format baris `DYNAMIC`. Cluster query paralel dapat menggunakan otentikasi AWS Identity and Access Management (IAM). | 2 September 2020 |

| | | |
|---|---|---|
| | Instans DB pembaca dalam klaster kueri paralel dapat memanfaatkan tingkat isolasi `READ COMMITTED` . Anda kini juga dapat membuat klaster kueri paralel di Wilayah AWS tambahan. Untuk informasi selengkapnya tentang fitur query paralel dan penyempur naan ini, lihat Parallel query untuk Aurora MySQL. | |
| Mengubah ke parameter Aurora MySQL binlog_ro ws_query_log_events | Anda kini dapat mengubah nilai parameter konfigurasi Aurora MySQL `binlog_ro ws_query_log_event s` . Sebelumnya, parameter ini tidak dapat diubah. | 26 Agustus 2020 |

| | | |
|---|---|---|
| Dukungan untuk peningkatan otomatis versi minor untuk Aurora MySQL | Dengan Aurora MySQL, pengaturan Aktifkan peningkatan otomatis versi minor kini akan diterapkan saat Anda menentukannya untuk klaster DB Aurora MySQL. Saat Anda mengaktifkan peningkatan otomatis versi minor, Aurora secara otomatis meningkatkan ke versi minor baru saat versi baru tersebut dirilis. Peningkatan otomatis terjadi selama periode pemeliharaan untuk basis data. Untuk Aurora MySQL, fitur ini hanya berlaku untuk Aurora MySQL versi 2, yang kompatibel dengan MySQL 5.7. Pertama-tama, prosedur peningkatan otomatis akan mengarahkan klaster DB Aurora MySQL ke versi 2.07.2. Untuk informasi selengkapnya tentang cara kerja fitur ini pada Aurora MySQL, lihat Peningkatan dan Patch Basis Data untuk Amazon Aurora MySQL. | 3 Agustus 2020 |
| Aurora PostgreSQL mendukung peningkatan versi utama ke PostgreSQL versi 11 | Dengan Aurora PostgreSQL, Anda kini dapat meningkatkan mesin DB ke versi utama 11. Untuk informasi selengkapnya, lihat Meningkatkan mesin DB PostgreSQL untuk Aurora PostgreSQL. | 28 Juli 2020 |

| | | |
|---|---|---|
| Amazon Aurora mendukung AWS PrivateLink | Amazon Aurora sekarang mendukung pembuatan titik akhir Amazon VPC untuk panggilan Amazon RDS API untuk menjaga lalu lintas antara aplikasi dan Aurora di jaringan. AWS Untuk informasi selengkapnya, lihat titik akhir VPC antarmuka dan Amazon Aurora (AWS PrivateLink). | 9 Juli 2020 |
| Proksi RDS tersedia secara umum | Proksi RDS kini tersedia secara umum. Anda dapat menggunakan Proksi RDS dengan RDS untuk MySQL, Aurora MySQL, RDS untuk PostgreSQL, dan Aurora PostgreSQL untuk beban kerja produksi. Untuk informasi selengkapnya tentang Proksi RDS, lihat "Mengelola Koneksi dengan Proksi Amazon RDS" dalam Panduan Pengguna Amazon RDS atau Panduan Pengguna Aurora. | 30 Juni 2020 |

| | | |
|---|---|---|
| Penerusan tulis basis data global Aurora | Anda kini dapat mengaktif kan kemampuan tulis pada klaster sekunder di basis data global. Dengan penerusan tulis, Anda mengeluarkan pernyataan DML di klaster sekunder, Aurora akan meneruskan tulisan tersebut ke klaster primer, dan data yang diperbarui akan direplika si ke semua klaster sekunder. Untuk informasi selengkap nya, lihat Menulis penerusan untuk sekunder Wilayah AWS dengan database global Aurora. | 18 Juni 2020 |
| Aurora mendukung integrasi dengan AWS Backup | Anda dapat menggunakan AWS Backup untuk mengelola backup cluster Aurora DB. Untuk informasi selengkap nya, lihat Ikhtisar pencadang an dan pemulihan klaster DB Aurora. | 10 Juni 2020 |
| Aurora PostgreSQL mendukung kelas instans DB db.t3.large | Anda kini dapat membuat klaster DB Aurora PostgreSQ L yang menggunakan kelas instans DB db.t3.large. Untuk informasi selengkapnya, lihat Kelas instans DB. | 5 Juni 2020 |

| | | |
|---|---|---|
| Basis data global Aurora mendukung PostgreSQL versi 11.7 dan sasaran titik pemulihan (RPO) yang terkelola | Anda kini dapat membuat basis data global Aurora untuk mesin basis data PostgreSQL versi 11.7. Anda juga dapat mengelola bagaimana basis data global PostgreSQL dipulihkan dari kegagalan menggunakan sasaran titik pemulihan (RPO). Untuk informasi selengkapnya, lihat Pemulihan Bencana Lintas Wilayah untuk basis data global Aurora. | 4 Juni 2020 |
| Aurora MySQL mendukung pemantauan basis data dengan aliran aktivitas basis data | Aurora MySQL sekarang menyertakan aliran aktivitas database, yang menyediakan aliran near-real-time data aktivitas database dalam database relasional Anda. Untuk informasi selengkapnya, lihat Menggunakan aliran aktivitas basis data. | 2 Juni 2020 |
| Editor kueri tersedia dalam tambahan Wilayah AWS | Editor kueri untuk Aurora Serverless sekarang tersedia dalam tambahan. Wilayah AWS Untuk informasi selengkapnya, lihat Ketersediaan editor kueri. | 28 Mei 2020 |
| Data API tersedia dalam tambahan Wilayah AWS | Data API sekarang tersedia dalam tambahan Wilayah AWS. Untuk informasi selengkapnya, lihat Ketersediaan API Data. | 28 Mei 2020 |

| | | |
|---|---|---|
| Proksi RDS tersedia di Wilayah Kanada (Pusat) | Anda kini dapat menggunakan pratinjau Proksi RDS di Wilayah Kanada (Pusat). Untuk informasi selengkapnya tentang Proksi RDS, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau). | 28 Mei 2020 |
| Basis data global Aurora dan replika baca lintas wilayah | Untuk basis data global Aurora, Anda kini dapat membuat replika baca lintas Wilayah Aurora MySQL dari klaster primer di wilayah yang sama seperti klaster sekunder. Untuk informasi selengkapnya tentang Basis Data Global Aurora dan replika baca lintas Wilayah, lihat Bekerja dengan basis data global Amazon Aurora dan Mereplikasi DB Amazon Aurora MySQL. | 18 Mei 2020 |

| | | |
|---|---|---|
| Proxy RDS tersedia di lebih banyak Wilayah AWS | Anda kini dapat menggunakan pratinjau Proksi RDS di Wilayah AS Barat (California Utara), Wilayah Eropa (London), Wilayah Eropa (Frankfurt), Wilayah Asia Pasifik (Seoul), Wilayah Asia Pasifik (Mumbai), Wilayah Asia Pasifik (Singapura), dan Wilayah Asia Pasifik (Sydney). Untuk informasi selengkapnya tentang Proksi RDS, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau). | 13 Mei, 2020 |
| Edisi yang Kompatibel dengan Aurora PostgreSQL mendukung direktori aktif Microsoft on-premise atau yang di-host sendiri | Anda kini dapat menggunakan Direktori Aktif on-premise atau yang di-host sendiri untuk autentikasi pengguna Kerberos saat mereka terhubung ke klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL. | 7 Mei 2020 |

| | | |
|---|---|---|
| Cluster multi-master Aurora MySQL tersedia di lebih banyak Wilayah AWS | Anda kini dapat membuat klaster multi-master Aurora di Wilayah Asia Pasifik (Seoul), Wilayah Asia Pasifik (Tokyo), Wilayah Asia Pasifik (Mumbai), dan Wilayah Eropa (Frankfurt). Untuk informasi selengkapnya tentang klaster multi-master, lihat Bekerja dengan klaster multi-master Aurora. | 7 Mei 2020 |
| Wawasan Performa mendukung analisis statistik kueri Aurora MySQL yang sedang berjalan | Anda kini dapat menganalisis statistik kueri yang sedang berjalan dengan Wawasan Performa untuk instans DB Aurora MySQL. Untuk informasi selengkapnya, lihat Menganalisis statistik kueri yang sedang berjalan. | 5 Mei 2020 |
| Pustaka klien Java untuk API Data tersedia secara umum | Pustaka klien Java untuk API Data kini tersedia secara umum. Anda dapat mengunduh dan menggunakan pustaka klien Java untuk API Data. Pustaka ini memungkinkan Anda untuk memetakan kelas sisi klien terhadap permintaan dan respons dari API Data. Untuk informasi selengkapnya, lihat Menggunakan pustaka klien Java untuk API Data. | 30 April 2020 |

| | | |
|---|---|---|
| Amazon Aurora tersedia di Wilayah Eropa (Milan) | Amazon Aurora kini tersedia di Wilayah Eropa (Milan). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 28 April 2020 |
| Amazon Aurora tersedia di Wilayah Eropa (Milan) | Amazon Aurora kini tersedia di Wilayah Eropa (Milan). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 27 April 2020 |
| Amazon Aurora tersedia di Wilayah Afrika (Cape Town) | Amazon Aurora kini tersedia di Wilayah Afrika (Cape Town). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan. | 22 April 2020 |
| Aurora PostgreSQL kini mendukung kelas instans DB db.r5.16xlarge dan db.r5.8xlarge | Anda kini dapat membuat klaster DB Aurora PostgreSQL yang menjalankan PostgreSQL dan menggunakan kelas instans DB db.r5.16xlarge dan db.r5.8xlarge. Untuk informasi selengkapnya, lihat Spesifikasi perangkat keras untuk kelas instans DB untuk Aurora. | 8 April 2020 |

| | | |
|---|---|---|
| Proksi Amazon RDS untuk PostgreSQL | Proksi Amazon RDS kini tersedia untuk PostgreSQL. Anda dapat menggunakan Proksi RDS untuk mengurangi overhead manajemen koneksi pada klaster Anda dan juga kemungkinan kesalahan "terlalu banyak koneksi". Proksi RDS saat ini dalam pratinjau publik untuk PostgreSQL. Untuk informasi selengkapnya, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau). | 8 April 2020 |
| Basis data global Aurora kini mendukung Aurora PostgreSQL | Anda kini dapat membuat basis data global Aurora untuk mesin basis data PostgreSQL. Basis data global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh wilayah. Untuk informasi selengkapnya, lihat Bekerja dengan basis data global Amazon Aurora. | 10 Maret 2020 |

| | | |
|---|---|---|
| Dukungan untuk peningkatan versi utama untuk Aurora PostgreSQL | Dengan Aurora PostgreSQL, Anda kini dapat meningkatkan mesin DB ke versi utama. Dengan demikian, Anda dapat langsung beralih ke versi utama terbaru saat meningkatkan versi mesin PostgreSQL tertentu. Untuk informasi selengkapnya, lihat Meningkatkan mesin DB PostgreSQL untuk Aurora PostgreSQL. | 4 Maret 2020 |
| Aurora PostgreSQL mendukung autentikasi Kerberos | Anda kini dapat menggunakan autentikasi Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL. | 28 Februari 2020 |
| Data API mendukung AWS PrivateLink | Data API sekarang mendukung pembuatan titik akhir Amazon VPC untuk panggilan API Data untuk menjaga lalu lintas antara aplikasi dan API Data di jaringan. AWS Untuk informasi selengkapnya, lihat Membuat titik akhir VPC Amazon (AWS PrivateLink) untuk API Data. | 6 Februari 2020 |

| | | |
|---|---|---|
| Dukungan machine learning Aurora di Aurora PostgreSQL | Ekstensi `aws_ml` Aurora PostgreSQL menyediakan fungsi yang Anda gunakan dalam kueri database Anda untuk memanggil Amazon Comprehend untuk analisis sentimen dan AI untuk menjalankan model pembelajaran mesin Anda sendiri. SageMaker Untuk informasi selengkapnya, lihat Menggunakan kemampuan machine learning (ML) dengan Aurora. | 5 Februari 2020 |
| Aurora PostgreSQL mendukung pengeksporan data ke Amazon S3 | Anda dapat membuat kueri data dari klaster DB Aurora PostgreSQL dan mengekspornya langsung ke file yang tersimpan dalam bucket Amazon S3. Untuk informasi selengkapnya, lihat Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3. | 5 Februari 2020 |
| Dukungan untuk mengekspor data snapshot DB ke Amazon S3 | Amazon Aurora mendukung pengeksporan data snapshot DB ke Amazon S3 for MySQL dan PostgreSQL. Untuk informasi selengkapnya, lihat Mengekspor data snapshot DB ke Amazon S3. | 9 Januari 2020 |

| | | |
|---|---|---|
| Catatan rilis Aurora MySQL dalam riwayat dokumen | Bagian ini sekarang menyertakan entri riwayat untuk catatan rilis Edisi yang Kompatibel dengan Aurora MySQL untuk versi yang dirilis setelah 31 Agustus 2018. Untuk catatan rilis lengkap dari versi tertentu, pilih tautan di kolom pertama entri riwayat. | 13 Desember 2019 |
| Proksi Amazon RDS | Anda dapat mengurangi overhead manajemen koneksi pada klaster, dan mengurangi kemungkinan kesalahan "terlalu banyak koneksi", dengan menggunakan Proksi Amazon RDS. Anda mengaitkan setiap proksi dengan instans DB RDS atau klaster DB Aurora. Kemudian, Anda menggunakan titik akhir proksi dalam string koneksi untuk aplikasi Anda. Proksi Amazon RDS saat ini dalam status pratinjau publik. Proksi ini mendukung mesin basis data Aurora MySQL. Untuk informasi selengkapnya, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau). | 3 Desember 2019 |

| | | |
|---|---|---|
| API Data untuk Aurora Serverless v1 mendukung petunjuk pemetaan jenis data | Anda kini dapat menggunakan petunjuk untuk menginstruksikan API Data untuk Aurora Serverless v1 agar mengirim nilai `String` ke basis data sebagai jenis yang berbeda. Untuk informasi selengkapnya, Memanggil API Data. | 26 November 2019 |
| API Data untuk Aurora Serverless v1 mendukung pustaka klien Java (pratinjau) | Anda dapat mengunduh dan menggunakan pustaka klien Java untuk API Data. Pustaka ini memungkinkan Anda untuk memetakan kelas sisi klien terhadap permintaan dan respons dari API Data. Untuk informasi selengkapnya, lihat Menggunakan pustaka klien Java untuk API Data. | 26 November 2019 |
| Aurora PostgreSQL memenuhi syarat FedRAMP HIGH | Aurora PostgreSQL memenuhi syarat FedRAMP HIGH. Untuk detail tentang AWS dan upaya kepatuhan, lihat AWS layanan dalam cakupan berdasarkan program kepatuhan. | 26 November 2019 |

| | | |
|---|---|---|
| Tingkat isolasi READ COMMITTED diaktifkan untuk replika Amazon Aurora MySQL | Anda kini dapat mengaktifkan tingkat isolasi `READ COMMITTED` pada Replika Aurora MySQL. Untuk melakukan hal ini, Anda harus mengaktifkan pengaturan konfigurasi `aurora_read_replica_read_committed_isolation_enabled` di tingkat sesi. Menggunakan tingkat isolasi `READ COMMITTED` untuk kueri yang membutuhkan waktu lama pada klaster OLTP dapat membantu mengatasi masalah terkait panjang daftar riwayat. Sebelum mengaktifkan pengaturan ini, pastikan untuk memahami perbedaan perilaku isolasi pada Replika Aurora dengan penerapan MySQL standar dari `READ COMMITTED`. Untuk informasi selengkapnya, lihat Tingkat isolasi Aurora MySQL. | 25 November 2019 |
| Wawasan Performa mendukung analisis statistik kueri Aurora PostgreSQL yang sedang berjalan | Anda kini dapat menganalisis statistik kueri yang sedang berjalan dengan Wawasan Performa untuk instans DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menganalisis statistik kueri yang sedang berjalan. | 25 November 2019 |

| | | |
|---|---|---|
| Ada lebih banyak klaster dalam basis data global Aurora | Anda kini dapat menambahkan beberapa wilayah sekunder ke basis data global Aurora. Anda dapat memanfaatkan pembacaan global latensi rendah dan pemulihan bencana di area geografis yang lebih luas. Untuk informasi selengkapnya tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora. | 25 November 2019 |
| Dukungan machine learning Aurora di Aurora MySQL | Di Aurora MySQL 2.07 dan lebih tinggi, Anda dapat menghubungi Amazon Comprehend untuk analisis sentimen dan AI untuk berbagai macam algoritma pembelajaran mesin. SageMaker Anda dapat menggunakan hasilnya secara langsung dalam aplikasi basis data dengan menyematkan panggilan ke fungsi yang tersimpan dalam kueri Anda. Untuk informasi selengkapnya, lihat Menggunakan kemampuan machine learning (ML) dengan Aurora. | 25 November 2019 |

| | | |
|---|---|---|
| Basis data global Aurora tidak lagi membutuhkan pengaturan mode mesin | Anda tidak perlu lagi menentukan `--engine-mode=global` saat membuat klaster yang ditujukan untuk menjadi bagian dari basis data global Aurora. Semua klaster Aurora yang memenuhi persyaratan kompatibilitas layak untuk menjadi bagian dari basis data global. Misalnya, klaster saat ini harus menggunakan Aurora MySQL versi 1 dengan kompatibilitas MySQL 5.6. Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora. | 25 November 2019 |
| Basis data global Aurora tersedia untuk Aurora MySQL versi 2 | Mulai dari Aurora MySQL 2.07, Anda dapat membuat basis data global Aurora dengan kompatibilitas MySQL 5.7. Anda tidak perlu menentukan mode mesin `global` untuk klaster primer atau sekunder. Anda dapat menambahkan klaster yang baru disediakan dengan Aurora MySQL 2.07 atau yang lebih tinggi ke Basis Data Global Aurora. Untuk informasi tentang Basis Data Global Aurora, lihat Bekerja dengan basis data global Amazon Aurora. | 25 November 2019 |

| | | |
|---|---|---|
| Pengoptimalan pertentangan hot row Aurora MySQL tersedia tanpa mode lab | Pengoptimalan pertentangan hot row kini sudah tersedia secara umum untuk Aurora MySQL dan mode lab Aurora tidak perlu ditetapkan ke ON. Fitur ini meningkatkan secara signifikan throughput untuk beban kerja dengan banyak transaksi yang bersaing untuk baris di halaman yang sama. Peningkatan ini melibatkan perubahan algoritma pembukaan kunci yang digunakan oleh Aurora MySQL. | 19 November 2019 |
| Hash join Aurora MySQL tersedia tanpa mode lab | Fitur hash join sudah tersedia secara umum untuk Aurora MySQL dan mode lab Aurora tidak perlu ditetapkan ke ON. Fitur ini dapat meningkatkan performa kueri saat Anda harus menggabungkan data dalam jumlah besar menggunakan equijoin. Untuk informasi selengkapnya tentang cara menggunakan fitur ini, lihat Bekerja dengan hash join di Aurora MySQL. | 19 November 2019 |

| | | |
|---|---|---|
| Dukungan Aurora MySQL 2.* untuk lebih banyak kelas instans db.r5 | Klaster Aurora MySQL kini mendukung jenis instans db.r5.8xlarge, db.r5.16x large, dan db.r5.24xlarge. Untuk informasi selengkapnya tentang jenis instans untuk klaster Aurora MySQL, lihat Memilih kelas instans DB. | 19 November 2019 |
| Dukungan Aurora MySQL 2.* untuk pelacakan mundur | Aurora MySQL versi 2.* kini menawarkan sebuah cara cepat untuk pulih dari kesalahan pengguna, seperti meletakkan tabel yang salah atau menghapus baris yang salah. Pelacakan mundur memungkinkan Anda memindahkan basis data ke titik waktu sebelumnya tanpa harus memulihkan dari cadangan, dan dapat diselesai kan dalam hitungan detik, bahkan untuk basis data yang besar. Untuk perincian, lihat Melacak mundur klaster DB Aurora. | 19 November 2019 |

| | | |
|---|---|---|
| Dukungan tag penagihan untuk Aurora | Anda kini dapat menggunakan tag untuk terus melacak alokasi biaya untuk sumber daya seperti klaster Aurora, instans DB dalam klaster Aurora, I/O, cadangan, snapshot, dan sebagainya. Anda dapat melihat biaya yang terkait dengan setiap tag menggunakan AWS Cost Explorer. Untuk informasi selengkapnya tentang cara menggunakan tag dengan Aurora, lihat Memberikan Tag pada sumber daya Amazon RDS. Untuk informasi umum tentang tag dan cara menggunakannya untuk analisis biaya, lihat Menggunakan tag alokasi biaya dan Tag alokasi dana yang ditentukan pengguna. | 23 Oktober 2019 |
| API Data untuk Aurora PostgreSQL | Aurora PostgreSQL kini mendukung penggunaan API Data dengan klaster DB Amazon Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan API Data untuk Aurora Serverless v1. | 23 September 2019 |

| | | |
|---|---|---|
| Aurora PostgreSQL mendukung pengunggahan log database ke log CloudWatch | Anda dapat mengonfigurasi klaster DB PostgreSQL Aurora untuk mempublikasikan data log ke grup log di Amazon Logs. CloudWatch Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Anda dapat menggunakan CloudWatch Log untuk menyimpan catatan log Anda dalam penyimpanan yang sangat tahan lama. Untuk informasi selengkapnya, lihat Menerbitkan log PostgreSQL Aurora ke Amazon Logs. CloudWatch | 9 Agustus 2019 |
| Klaster multi-master untuk Aurora MySQL | Anda dapat menyiapkan klaster multi-master Aurora MySQL. Dalam klaster ini, setiap instans DB memiliki kemampuan baca/tulis. Untuk informasi selengkapnya, lihat Bekerja dengan klaster multi-master Aurora. | 8 Agustus 2019 |

| | | |
|---|---|---|
| Aurora PostgreSQL mendukung Aurora Serverless v1 | Anda kini dapat menggunakan Amazon Aurora Serverless v1 dengan Aurora PostgreSQL. Klaster DB Aurora Nirserver akan secara otomatis aktif, nonaktif, dan menaikkan atau menurunkan skala kapasitas komputasinya berdasarkan kebutuhan aplikasi Anda. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1. | 9 Juli 2019 |
| Pengklonaan lintas akun untuk Aurora MySQL | Anda sekarang dapat mengkloning volume cluster untuk cluster Aurora MySQL DB antar akun. AWS Anda mengizinkan berbagi melalui AWS Resource Access Manager (AWS RAM). Volume kloning kloning menggunakan copy-on-write mekanisme, yang hanya membutuhkan penyimpanan tambahan untuk data baru atau yang diubah. Untuk informasi selengkapnya tentang pengklonaan untuk Aurora, lihat Mengklona basis data di klaster DB Aurora. | 2 Juli 2019 |
| Aurora PostgreSQL mendukung kelas instans DB db.t3 | Anda kini dapat membuat klaster DB Aurora PostgreSQL yang menggunakan kelas instans DB db.t3. Untuk informasi selengkapnya, lihat Kelas instans DB. | 20 Juni 2019 |

| | | |
|---|---|---|
| Dukungan untuk pengimpor an data dari Amazon S3 untuk Aurora PostgreSQL | Anda kini dapat mengimpor data dari file Amazon S3 ke tabel di klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Mengekspor data Amazon S3 ke klaster DB Aurora PostgreSQL. | 19 Juni 2019 |
| Aurora PostgreSQL kini menyediakan pemulihan failover cepat dengan manajemen cache klaster | Aurora PostgreSQL kini menyediakan manajemen cache klaster untuk memastika n pemulihan instans DB primer yang cepat jika terjadi failover. Untuk informasi selengkapnya, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster. | 11 Juni 2019 |
| API Data untuk Aurora Serverless v1 tersedia secara umum | Anda dapat mengakses klaster Aurora Serverless v1 dengan aplikasi berbasis layanan web menggunakan Data API. Untuk informasi selengkapnya, lihat Menggunakan API Data untuk Aurora Serverless v1. | 30 Mei 2019 |
| Aurora PostgreSQL mendukung pemantauan basis data dengan aliran aktivitas basis data | Aurora PostgreSQL sekarang menyertakan aliran aktivitas database, yang menyediakan aliran data aktivitas database dalam near-real-time database relasional Anda. Untuk informasi selengkapnya, lihat Menggunakan aliran aktivitas basis data. | 30 Mei 2019 |

| | | |
|---|---|---|
| Rekomendasi Amazon Aurora | Saat ini, Amazon Aurora menyediakan rekomendasi otomatis untuk sumber daya Aurora. Untuk informasi selengkapnya, lihat Menggunakan rekomendasi Amazon Aurora. | 22 Mei 2019 |
| Dukungan Wawasan Performa untuk basis data global Aurora | Anda kini dapat menggunakan Wawasan Performa dengan Basis Data Global Aurora. Untuk informasi tentang Wawasan Performa Aurora, lihat Menggunakan wawasan performa Amazon RDS. Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Aurora. | 13 Mei 2019 |
| Wawasan Performa tersedia untuk Aurora MySQL 5.7 | Wawasan Performa Amazon RDS kini tersedia untuk Aurora MySQL versi 2.x, yang kompatibel dengan MySQL 5.7. Untuk informasi selengkapnya, lihat Menggunakan wawasan performa Amazon RDS. | 3 Mei 2019 |

| | | |
|---|---|---|
| Database global Aurora tersedia di lebih banyak Wilayah AWS | Anda sekarang dapat membuat database global Aurora di sebagian besar tempat Wilayah AWS Aurora tersedia. Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora. | 30 April 2019 |
| Kapasitas minimum adalah sebesar 1 untuk Aurora Serverless v1 | Pengaturan kapasitas minimum yang dapat Anda gunakan untuk sebuah klaster Aurora Serverles s v1 adalah 1. Sebelumny a, jumlah minimumnya adalah 2. Untuk informasi tentang cara menentukan nilai kapasitas Aurora Nirserver, lihat Menetapkan kapasitas klaster DB Aurora Serverless v1. | 29 April 2019 |
| Tindakan waktu tunggu habis Aurora Serverless v1 | Anda kini dapat menentukan tindakan yang harus diambil saat perubahan kapasitas Aurora Serverless v1 habis waktu tunggunya. Untuk informasi selengkapnya, lihat Tindakan waktu tunggu habis untuk perubahan kapasitas. | 29 April 2019 |

| | | |
|---|---|---|
| Penagihan per detik | Amazon RDS sekarang ditagih dalam kenaikan 1 detik di semua Wilayah AWS kecuali AWS GovCloud (AS) untuk instans sesuai permintaan. Untuk informasi selengkapnya, lihat Penagihan instans DB untuk Aurora. | 25 April 2019 |
| Berbagi Aurora Serverless v1 snapshot di seluruh Wilayah AWS | Dengan Aurora Serverless v1, snapshot selalu dienkripsi. Jika Anda mengenkripsi snapshot dengan milik Anda sendiri AWS KMS key, Anda sekarang dapat menyalin atau membagikan snapshot tersebut. Wilayah AWS Untuk informasi tentang snapshot klaster DB Aurora Serverless v1, lihat Aurora Serverless v1 dan snapshot. | 17 April 2019 |
| Memulihkan cadangan MySQL 5.7 dari Amazon S3 | Anda kini dapat membuat sebuah cadangan basis data MySQL versi 5.7, menyimpannya di Amazon S3, lalu memulihkan file cadangan ke sebuah klaster DB Aurora MySQL yang baru. Untuk informasi selengkapnya, lihat Memigrasikan data dari basis data MySQL eksternal ke klaster DB Aurora MySQL. | 17 April 2019 |

| | | |
|---|---|---|
| Berbagi snapshot Aurora Serverless v1 antar-wilayah | Dengan Aurora Serverless v1, snapshot selalu dienkripsi. Jika Anda mengenkripsi snapshot dengan milik Anda sendiri AWS KMS key, Anda sekarang dapat menyalin atau membagikan snapshot di seluruh wilayah. Untuk informasi tentang snapshot klaster DB Aurora Serverless v1, lihat Aurora Nirserver dan snapshot. | 16 April 2019 |
| Tutorial Aurora proof-of-concept | Anda dapat mempelajari cara melakukan pembuktian konsep untuk mencoba aplikasi dan beban kerja Anda dengan Aurora. Untuk tutorial lengkap, lihat Melakukan pembuktian konsep Aurora. | 16 April 2019 |
| Aurora Serverless v1 mendukung pemulihan dari cadangan Amazon S3 | Anda kini dapat mengimpor backup dari Amazon S3 ke klaster Aurora Serverless. Untuk detail tentang prosedur tersebut, lihat Memigrasikan data dari MySQL dengan menggunakan bucket Amazon S3. | 16 April 2019 |

| | | |
|---|---|---|
| Parameter baru yang dapat dimodifikasi untuk Aurora Serverless v1 | Anda kini dapat memodifikasi parameter DB berikut untuk klaster Aurora Serverless v1: `innodb_file_format` , `innodb_file_per_table` , `innodb_large_prefix` , `innodb_lock_wait_timeout` , `innodb_monitor_disable` , `innodb_monitor_enable` , `innodb_monitor_reset` , `innodb_monitor_reset_all` , `innodb_print_all_deadlocks` , `log_warnings` , `net_read_timeout` , `net_retry_count` , `net_write_timeout` , `sql_mode`, dan `tx_isolation` . Untuk informasi selengkapnya tentang parameter konfigurasi untuk klaster Aurora Serverless v1, lihat Aurora Serverless v1 dan grup parameter. | 4 April 2019 |
| Aurora PostgreSQL mendukung kelas instans DB db.r5 | Anda kini dapat membuat klaster DB Aurora PostgreSQL yang menggunakan kelas instans DB db.r5. Untuk informasi selengkapnya, lihat Kelas instans DB. | 4 April 2019 |

| | | |
|---|---|---|
| Replikasi logis Aurora PostgreSQL | Anda kini dapat menggunakan replikasi logis PostgreSQL untuk mereplikasi bagian dari basis data untuk klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menggunakan replikasi logis PostgreSQL. | 28 Maret 2019 |
| Dukungan GTID untuk Aurora MySQL 2.04 | Anda kini dapat menggunakan replikasi dengan fitur ID transaksi global (GTID) dari MySQL 5.7. Fitur ini menyederhanakan proses replikasi log biner (binlog) antara Aurora MySQL dan basis data eksternal yang kompatibel dengan MySQL 5.7. Replikasi tersebut dapat menggunakan klaster Aurora MySQL sebagai sumber atau tujuan. Fitur ini tersedia untuk Aurora MySQL 2.04 dan yang lebih tinggi. Untuk informasi selengkapnya tentang replikasi berbasis GTID dan Aurora MySQL, lihat Menggunakan replikasi berbasis GTID untuk Aurora MySQL. | 25 Maret 2019 |

| | | |
|---|---|---|
| Mengunggah Aurora Serverless v1 log ke Amazon CloudWatch | Anda sekarang dapat meminta Aurora mengunggah log database CloudWatch untuk sebuah Aurora Serverless v1 cluster. Untuk informasi selengkapnya, lihat Melihat klaster DB Aurora Nirserver. Sebagai bagian dari peningkatan ini, Anda kini dapat menentukan nilai untuk parameter tingkat instans di grup parameter klaster DB, dan nilai tersebut berlaku untuk semua instans DB di klaster tersebut kecuali jika Anda menggantinya di grup parameter DB. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter DB dan grup parameter klaster DB. | 25 Februari 2019 |
| Aurora MySQL mendukung kelas instans DB db.t3 | Anda kini dapat membuat klaster DB Aurora MySQL yang menggunakan kelas instans DB db.t3. Untuk informasi selengkapnya, lihat Kelas instans DB. | 25 Februari 2019 |
| Aurora MySQL mendukung kelas instans DB db.r5 | Anda kini dapat membuat klaster DB Aurora MySQL yang menggunakan kelas instans DB db.r5. Untuk informasi selengkapnya, lihat Kelas instans DB. | 25 Februari 2019 |

| | | |
|---|---|---|
| Penghitung Wawasan Performa untuk Aurora MySQL | Anda kini dapat menambahkan penghitung performa ke diagram Wawasan Performa untuk instans DB Aurora MySQL. Untuk informasi selengkapnya, lihat Komponen dasbor Wawasan Performa. | 19 Februari 2019 |
| Wawasan Performa Amazon RDS mendukung tampilan lebih banyak teks SQL untuk Aurora MySQL | Wawasan Performa Amazon RDS kini mendukung tampilan lebih banyak teks SQL di dasbor Wawasan Performa untuk instans DB Aurora MySQL. Untuk informasi selengkapnya, lihat Menampilkan lebih banyak teks SQL di dasbor Wawasan Performa. | 6 Februari 2019 |
| Wawasan Performa Amazon RDS mendukung tampilan lebih banyak teks SQL untuk Aurora PostgreSQL | Wawasan Performa Amazon RDS kini mendukung tampilan lebih banyak teks SQL di dasbor Wawasan Performa untuk instans DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menampilkan lebih banyak teks SQL di dasbor Wawasan Performa. | 24 Januari 2019 |

| | | |
|---|---|---|
| Penagihan cadangan Aurora | Anda dapat menggunakan CloudWatch metrik `AmazonTotalBackupStorageBilled`, `SnapshotStorageUsed`, dan `BackupRetentionPeriodStorageUsed` untuk memantau penggunaan ruang cadangan Aurora Anda. Untuk informasi selengkapnya tentang cara menggunakan CloudWatch metrik, lihat Ikhtisar pemantauan. Untuk informasi selengkapnya tentang cara mengelola penyimpanan untuk data cadangan, lihat Memahami penggunaan penyimpanan cadangan Aurora. | 3 Januari 2019 |
| Penghitung Wawasan Performa | Anda kini dapat menambahkan penghitung performa ke diagram Wawasan Performa. Untuk informasi selengkapnya, lihat Komponen dasbor Wawasan Performa. | 6 Desember 2018 |

| | | |
|---|---|---|
| Basis data global Aurora | Anda kini dapat membuat basis data global Aurora. Basis data global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh wilayah. Untuk informasi selengkapnya, lihat Bekerja dengan basis data global Amazon Aurora. | 28 November 2018 |
| Manajemen rencana kueri di Aurora PostgreSQL | Aurora PostgreSQL kini menyediakan manajemen rencana kueri yang dapat Anda gunakan untuk mengelola rencana eksekusi kueri PostgreSQL. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL. | 20 November 2018 |
| Editor kueri untuk Aurora Serverless v1 (beta) | Anda dapat menjalankan pernyataan SQL dalam konsol Amazon RDS pada klaster Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan editor kueri untuk Aurora Serverless v1. | 20 November 2018 |

| | | |
|---|---|---|
| API Data untuk Aurora Serverless v1 (beta) | Anda dapat mengakses klaster Aurora Serverless v1 dengan aplikasi berbasis layanan web menggunakan API Data. Untuk informasi selengkapnya, lihat Menggunakan API Data untuk Aurora Nirserver. | 20 November 2018 |
| Dukungan TLS untuk Aurora Serverless v1 | Aurora Serverless v1cluster mendukung TLS/SSL enkripsi. Untuk informasi selengkapnya, lihat TLS/SSL untuk Aurora Nirserver. | 19 November 2018 |
| Titik akhir kustom | Anda kini dapat membuat titik akhir yang dikaitkan dengan kumpulan instans DB arbitrer. Fitur ini membantu penyeimbangan beban dan ketersediaan yang tinggi untuk klaster Aurora dengan beberapa instans DB memiliki kapasitas atau konfigurasi yang berbeda dibandingkan yang lain. Anda dapat menggunakan titik akhir kustom, bukannya terhubung ke instans DB tertentu melalui titik akhir instansnya. Untuk informasi selengkapnya, lihat Titik akhir Amazon Aurora. | 12 November 2018 |
| Dukungan autentikasi IAM di Aurora PostgreSQL | Aurora PostgreSQL kini mendukung autentikasi IAM. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM. | 8 November 2018 |

| | | |
|---|---|---|
| Grup parameter kustom untuk restorasi dan pemulihan titik waktu | Anda kini dapat menentukan grup parameter kustom saat memulihkan snapshot atau melakukan operasi pemulihan titik waktu. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB dan Memulihkan klaster DB ke waktu yang ditentukan. | 15 Oktober 2018 |
| Perlindungan penghapusan untuk klaster DB Aurora | Saat Anda mengaktifkan perlindungan penghapusan untuk klaster DB, basis data tidak dapat dihapus oleh pengguna. Untuk informasi selengkapnya, lihat Menghapus klaster DB. | 26 September 2018 |
| Fitur Stop/Start Aurora | Anda kini dapat menonaktifkan atau mengaktifkan keseluruhan klaster Aurora dengan satu pengoperasian. Untuk informasi selengkapnya, lihat Menonaktifkan dan mengaktifkan klaster Aurora. | 24 September 2018 |

| | | |
|---|---|---|
| Fitur kueri paralel untuk Aurora MySQL | Aurora MySQL sekarang menawarkan opsi untuk memparalelkan I/O pekerjaan untuk kueri di seluruh infrastruktur penyimpanan Aurora. Fitur ini mempercepat kueri analitik intensif data, yang sering kali merupakan pengoperasian yang paling banyak memakan waktu dalam beban kerja. Untuk informasi selengkapnya, lihat Kueri paralel untuk Aurora MySQL. | 20 September 2018 |
| Panduan baru | Ini merupakan rilisan pertama dari Panduan Pengguna Amazon Aurora. | 31 Agustus 2018 |

# AWS Glosarium

Untuk AWS terminologi terbaru, lihat AWS glosarium di Referensi.Glosarium AWS

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.