aws

Panduan Pengguna

Amazon Aurora DSQL

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Aurora DSQL: Panduan Pengguna

Copyright © 2026 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 pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon Aurora DSQL? 1
Kapan harus digunakan 1
Fitur utama 1
Wilayah AWS ketersediaan 3
Multi-Region kluster 5
Harga 6
Apa selanjutnya? 6
Memulai 7
Prasyarat 7
Buat kluster Single-region 7
Connect ke sebuah cluster 8
Jalankan perintah SQL 9
Buat klaster Multi-wilayah 10
Pemecahan masalah 12
Cara kerja penagihan 13
Cara kerja metering 13
Pengukuran komponen DPU 14
ComputedPU 14
DitulisPU 14
bacaDPU 15
Contoh penagihan 16
Multi-Region penagihan 20
Memantau penggunaan DPU 21
Metrik DPU yang tersedia 21
Melihat metrik DPU 21
Metrik observabilitas tambahan 22
JELASKAN ANALISIS VERBOSE untuk kesadaran biaya 22
Contoh 1: SELECT query 23
Contoh 2: Insert query 24
Menggunakan informasi DPU untuk optimasi 24
Menafsirkan informasi DPU 25
Praktik terbaik estimasi biaya 26
Autentikasi dan otorisasi 27
Mengelola klaster Anda 27

Menghubungkan ke cluster Anda 27
PostgreSQL dan peran IAM 28
Menggunakan tindakan kebijakan IAM dengan Aurora DSQL 29
Menggunakan tindakan kebijakan IAM untuk terhubung ke cluster 29
Menggunakan tindakan kebijakan IAM untuk mengelola cluster 30
Mencabut otorisasi menggunakan IAM dan PostgreSQL 31
Menghasilkan token otentikasi 32
Konsol 32
AWS CloudShell 33
AWS CLI 34
Aurora DSQL SDK 35
Peran database dan otentikasi IAM 44
Peran IAM 44
Pengguna IAM: 45
Hubungkan 45
Kueri 45
Lihat pemetaan 46
Mencabut 46
Aurora DSQL dan PostgreSQL 48
Sorotan kompatibilitas 48
Manfaat arsitektur terdistribusi 49
Kompatibilitas SQL 49
Jenis data yang didukung 50
Fitur SQL yang didukung 56
Subset perintah SQL yang didukung 60
Panduan migrasi 83
Kontrol konkurensi 90
Respons kontrol konkurensi 90
Pedoman untuk mengoptimalkan kinerja transaksi 91
DDL dan transaksi terdistribusi 91
Kunci primer 93
Struktur dan penyimpanan data 93
Pedoman untuk memilih kunci utama 93
Urutan dan kolom identitas 94
Fungsi manipulasi urutan 95
Kolom identitas 98

Bekerja dengan urutan dan kolom identitas 99
Indeks asinkron 101
Sintaksis 102
Parameter 102
Catatan penggunaan 103
Membuat Indeks 103
Mengueri sebuah indeks 104
Kegagalan pembuatan indeks unik 106
Pelanggaran keunikan 106
Tabel dan perintah sistem 108
Tabel sistem 108
Kueri sistem yang berguna 119
Perintah ANALYSIS 120
JELASKAN rencana 121
PostgreSQL JELASKAN rencana 121
Elemen kunci 122
Pemfilteran 122
Membaca rencana JELASKAN 123
DPU dalam EXPLY ANALYSIS 127
Mengelola klaster Aurora DSQL 131
Single-Region kluster 131
Penggunaan AWS SDK 131
Penggunaan AWS CLI 170
Multi-Region kluster 173
Penggunaan AWS SDK 174
Penggunaan AWS CLI 228
CloudFormation 234
Konfigurasi awal 234
Temukan kluster 235
Perbarui konfigurasi 235
Siklus hidup Aurora DSQL Cluster 236
Status cluster 236
Melihat status cluster 238
Pemrograman dengan Aurora DSQL 240
Konektor 241
Konektor JDBC 241

Konektor Python ... 246
Konektor Go ... 257
Node.js konektor ... 264
Konektor Ruby ... 272
Konektor PHP ... 278
Konektor .NET ... 283
Konektor karat ... 288
Mengakses Aurora DSQL ... 295
Parameter sesi ... 296
Klien SQL ... 299
DBeaver ... 300
JetBrains DataGrip ... 303
Psql ... 304
VSCode ... 305
Pemecahan masalah ... 307
Alat Konektivitas Database ... 307
Adaptor Aurora DSQL ... 307
Sampel Driver Database ... 308
ORM dan Sampel Kerangka Kerja ... 310
Memuat data ... 311
Memilih pendekatan pemuatan ... 311
Aurora DSQL Loader ... 312
Jalur migrasi ... 317
Menggunakan PostgreSQL\ copy ... 319
Sumber daya tambahan ... 320
AI Generatif ... 320
AWS Labs Aurora DSQL MCP Server ... 320
Aurora DSQL Steering: Keterampilan dan Kekuatan ... 329
Editor Kueri ... 334
Prasyarat ... 335
Bekerja dengan Query Editor ... 335
Editor Kueri: Menggunakan JupyterLab dengan Aurora DSQL ... 337
Memulai ... 337
Contoh buku catatan ... 340
Sumber bacaan lebih lanjut ... 340
Pencadangan dan pemulihan ... 341

Memulai dengan AWS Backup 341
Memulihkan backup 341
Memulihkan kluster wilayah tunggal 342
Memulihkan klaster Multi-wilayah 342
Pemantauan dan kepatuhan 342
Sumber daya tambahan 343
Aliran CDC (Pratinjau) 344
Cara kerjanya 344
Topik di halaman ini 345
Topik terkait 345
Pemesanan dan pengiriman semantik 345
Jaminan pengiriman 345
Memesan 346
Strategi konsumen 346
Multi-Region Konfigurasi aliran CDC 347
Memproses catatan CDC di hilir 347
Memulai 348
Prasyarat 349
Langkah 1: Buat aliran data Amazon Kinesis 349
Langkah 2: Buat peran IAM untuk Aurora DSQL 351
Langkah 3: Buat aliran CDC 353
Langkah 4: Verifikasi bahwa catatan mengalir 354
Langkah 5: Konsumsi catatan dengan skrip Python 355
Mengelola aliran CDC 359
Mengkonfigurasi IAM 360
Izin penelepon 360
Peran layanan 361
Kebijakan kepercayaan peran layanan 362
Kebijakan izin peran layanan 363
Perlindungan data 364
Memahami catatan CDC 365
Bagaimana catatan memetakan ke Amazon Kinesis 365
Kunci utama dalam muatan 365
Rekam muatan 366
Bidang payload 368
Rincian format 370

Catatan besar ... 371
Serialisasi tipe data ... 373
Evolusi skema dalam catatan CDC ... 378
Pemantauan aliran ... 378
Siklus hidup streaming ... 379
Memecahkan masalah aliran yang terganggu atau gagal ... 380
Referensi kode kesalahan ... 380
Memulihkan aliran yang terganggu ... 385
Pemantauan kesehatan aliran ... 385
CloudWatch metrik untuk aliran CDC ... 385
Memantau praktik terbaik ... 387
Pencatatan dan pemantauan ... 390
Pemantauan CloudWatch dengan ... 390
Observabilitas ... 390
Penggunaan ... 392
Aliran CDC ... 394
Logging dengan CloudTrail ... 396
Peristiwa manajemen ... 397
Peristiwa data ... 399
Keamanan ... 401
AWS kebijakan terkelola ... 402
AmazonAuroraDSQLFullAccess ... 402
AmazonAuroraDSQLReadOnlyAccess ... 404
AmazonAuroraDSQLConsoleFullAccess ... 404
AuroraDSQLServiceRolePolicy ... 406
Pembaruan kebijakan ... 406
Perlindungan data ... 414
Enkripsi data ... 415
Perlindungan Data di Daerah Saksi ... 417
SSL/TLS sertifikat ... 417
Enkripsi data ... 415
Jenis kunci KMS ... 423
Enkripsi saat diam ... 424
Menggunakan KMS dan kunci data ... 426
Mengotorisasi kunci KMS Anda ... 428
Konteks enkripsi ... 430

Pemantauan AWS KMS 431
Membuat cluster terenkripsi 433
Menghapus atau memperbarui kunci 435
Pertimbangan-pertimbangan 438
Manajemen identitas dan akses 438
Audiens 439
Mengautentikasi dengan identitas 439
Mengelola akses menggunakan kebijakan 441
Bagaimana Aurora DSQL bekerja dengan IAM 442
Identity-based contoh kebijakan 448
Pemecahan masalah 454
Resource-based kebijakan 456
Kapan Menggunakan 457
Buat dengan kebijakan 458
Menambahkan dan mengedit kebijakan 461
Lihat Kebijakan 464
Hapus Kebijakan 465
Contoh kebijakan 467
Blokir akses publik 471
Operasi API 474
Menggunakan peran terkait layanan 477
Service-linked izin peran untuk Aurora DSQL 477
Buat peran tertaut layanan 478
Edit peran tertaut layanan 478
Hapus peran tertaut layanan 478
Wilayah yang Didukung untuk peran terkait layanan Aurora DSQL 479
Menggunakan tombol kondisi IAM 479
Buat cluster di Wilayah tertentu 479
Buat klaster Multi-wilayah di Wilayah tertentu 480
Buat klaster Multi-wilayah dengan Wilayah saksi tertentu 481
Respons insiden 482
Validasi kepatuhan 482
Ketahanan 483
Pencadangan dan pemulihan 483
Replikasi 483
Ketersediaan tinggi 484

Pengujian injeksi kesalahan 485
Keamanan Infrastruktur 485
Mengelola cluster menggunakan AWS PrivateLink 486
Konfigurasi dan analisis kerentanan 497
Cross-service pencegahan wakil bingung 498
Peran layanan aliran CDC 499
Praktik terbaik keamanan 500
Praktik terbaik keamanan detektif 500
Praktik terbaik keamanan pencegahan 501
Memberikan tag ke sumber daya 503
Tag nama 503
Persyaratan penandaan 503
Tag aliran CDC 504
Menandai catatan penggunaan 504
Pertimbangan-pertimbangan 505
Kuota dan batas 507
Kuota cluster 507
Batas basis data 508
Referensi API 513
Pemecahan masalah 301
Kesalahan koneksi 514
Kesalahan autentikasi 515
Kesalahan otorisasi 516
Kesalahan SQL 517
Respons kontrol konkurensi 517
SSL/TLS koneksi 517
Memberikan umpan balik 519
Saluran umpan balik 519
Permintaan fitur yang efektif 519
Catatan rilis 520
2026 520
Juni 2026 520
Mei 2026 520
April 2026 521
Maret 2026 522
Februari 2026 523

Januari 2026 525
2025 525
Desember 2025 525
November 2025 526
Oktober 2025 527
Agustus 2025 527
Juli 2025 527
Mei 2025 528
2024 528
Desember 2024 528
Riwayat dokumen 529
..... dl

Apa itu Amazon Aurora DSQL?

Amazon Aurora DSQL adalah layanan database relasional terdistribusi tanpa server yang dioptimalkan untuk beban kerja transaksional. Aurora DSQL menawarkan skala yang hampir tidak terbatas dan tidak mengharuskan Anda untuk mengelola infrastruktur. Arsitektur aktif-aktif yang sangat tersedia menyediakan 99,99% Single-region dan 99,999% ketersediaan Multi-region.

Kapan menggunakan Aurora DSQL

Aurora DSQL dioptimalkan untuk beban kerja transaksional yang mendapat manfaat dari transaksi ACID dan model data relasional. Karena tanpa server, Aurora DSQL sangat ideal untuk pola aplikasi arsitektur microservice, serverless, dan event-driven. Aurora DSQL adalah PostgreSQL-compatible, sehingga Anda dapat menggunakan driver yang sudah dikenal, pemetaan relasional objek (ORM), kerangka kerja, dan fitur SQL.

Aurora DSQL secara otomatis mengelola infrastruktur sistem dan skala komputasi I/O, dan penyimpanan berdasarkan beban kerja Anda. Karena Anda tidak memiliki server untuk menyediakan atau mengelola, Anda tidak perlu khawatir tentang downtime pemeliharaan yang terkait dengan penyediaan, penambalan, atau peningkatan infrastruktur.

Aurora DSQL membantu Anda membangun dan memelihara aplikasi perusahaan yang selalu tersedia dalam skala apa pun. Desain tanpa server aktif mengotomatiskan pemulihan kegagalan, jadi Anda tidak perlu khawatir tentang failover database tradisional. Aplikasi Anda mendapat manfaat dari Multi-AZ dan ketersediaan Multi-wilayah, dan Anda tidak perlu khawatir tentang konsistensi akhirnya atau data yang hilang terkait dengan failover.

Fitur utama di Aurora DSQL

Fitur utama berikut membantu Anda membuat database terdistribusi tanpa server untuk mendukung aplikasi ketersediaan tinggi Anda:

Arsitektur terdistribusi

Aurora DSQL terdiri dari komponen-komponen multi-tenant berikut:

- Relay dan konektivitas
- Komputasi dan database

- Log transaksi, kontrol konkurensi, dan isolasi
- Penyimpanan

Bidang kontrol mengoordinasikan komponen sebelumnya. Setiap komponen menyediakan redundansi di tiga Availability Zone (AZ), dengan penskalaan cluster otomatis dan penyembuhan diri jika terjadi kegagalan komponen. Untuk mempelajari lebih lanjut tentang bagaimana arsitektur ini mendukung ketersediaan tinggi, lihatKetahanan di Amazon Aurora DSQL.

Single-Region dan klaster Multi-wilayah

Cluster Aurora DSQL memberikan manfaat sebagai berikut:

- Replikasi data sinkron
- Operasi baca yang konsisten
- Pemulihan kegagalan otomatis
- Konsistensi data di beberapa AZ atau Wilayah

Jika komponen infrastruktur gagal, Aurora DSQL secara otomatis merutekan permintaan ke infrastruktur yang sehat tanpa intervensi manual. Aurora DSQL menyediakan transaksi atomisitas, konsistensi, isolasi, dan daya tahan (ACID) dengan konsistensi yang kuat, isolasi snapshot, atomisitas, dan daya tahan lintas-AZ dan lintas wilayah.

Multi-Region cluster peered memberikan ketahanan dan konektivitas yang sama dengan cluster Single-region. Tetapi mereka meningkatkan ketersediaan dengan menawarkan dua titik akhir Regional, satu di setiap Region cluster peered. Kedua titik akhir dari cluster peered menyajikan database logis tunggal. Mereka tersedia untuk operasi baca dan tulis bersamaan, dan memberikan konsistensi data yang kuat. Anda dapat membangun aplikasi yang berjalan di beberapa Wilayah secara bersamaan untuk kinerja dan ketahanan—dan tahu bahwa pembaca selalu melihat data yang sama.

Kompatibilitas dengan PostgreSQL

Lapisan database terdistribusi (komputasi) di Aurora DSQL didasarkan pada versi utama PostgreSQL saat ini. Anda dapat terhubung ke Aurora DSQL dengan driver dan alat PostgreSQL yang sudah dikenal, seperti. `psql` Aurora DSQL saat ini kompatibel dengan PostgreSQL versi 16 dan mendukung berbagai fitur PostgreSQL, ekspresi, dan tipe data. Untuk informasi selengkapnya tentang fitur SQL yang didukung, lihatKompatibilitas fitur SQL di Aurora DSQL.

Ketersediaan wilayah untuk Aurora DSQL

Dengan Amazon Aurora DSQL, Anda dapat menerapkan instans database di beberapa Wilayah AWS untuk mendukung aplikasi global dan memenuhi persyaratan residensi data. Ketersediaan wilayah menentukan di mana Anda dapat membuat dan mengelola cluster database Aurora DSQL. Administrator database dan arsitek aplikasi yang perlu merancang sistem basis data yang sangat tersedia dan terdistribusi secara global sering kali perlu memahami dukungan Wilayah untuk beban kerja mereka. Kasus penggunaan umum termasuk menyiapkan pemulihan bencana lintas wilayah, melayani pengguna dari instans basis data yang lebih dekat secara geografis untuk mengurangi latensi, dan memelihara salinan data di lokasi tertentu untuk kepatuhan.

Tabel berikut menunjukkan di Wilayah AWS mana Aurora DSQL saat ini tersedia dan titik akhir untuk masing-masing. Wilayah AWS

Nama Wilayah	Wilayah	Titik Akhir	Protokol
AS Timur (Ohio)	us–east-2	dsql.us-east-2.api.aws dsql-fips.us-east-2.api.aws	HTTPS HTTPS
AS Timur (Virginia Utara)	us-east-1	dsql.us-east-1.api.aws dsql-fips.us-east-1.api.aws	HTTPS HTTPS
US West (Oregon)	us-west-2	dsql.us-west-2.api.aws dsql-fips.us-west-2.api.aws	HTTPS HTTPS
Asia Pasifik (Hong Kong)	ap-east-1	dsql.ap-east-1.api.aws	HTTPS
Asia Pasifik (Melbourne)	ap-southeast-4	dsql.ap-southeast-4.api.aws	HTTPS

Nama Wilayah	Wilayah	Titik Akhir	Protokol
Asia Pasifik (Mumbai)	ap-south-1	dsql.ap-south-1.api.aws	HTTPS
Asia Pasifik (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS
Asia Pasifik (Seoul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asia Pasifik (Singapura)	ap-southeast-1	dsql.ap-southeast-1.api.aws	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	dsql.ap-southeast-2.api.aws	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Canada (Central)	ca-central-1	dsql.ca-central-1.api.aws dsql-fips.ca-central-1.api.aws	HTTPS HTTPS
Kanada Barat (Calgary)	ca-west-1	dsql.ca-west-1.api.aws dsql-fips.ca-west-1.api.aws	HTTPS HTTPS
Eropa (Frankfurt)	eu-central-1	dsql.eu-central-1.api.aws	HTTPS

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
Eropa (Irlandia)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS	
Europe (London)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS	
Eropa (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS	
Eropa (Stockholm)	eu-north-1	dsql.eu-north-1.api.aws	HTTPS	
Amerika Selatan (Sao Paulo)	sa-east-1	dsql.sa-east-1.api.aws	HTTPS	

Multi-Region ketersediaan cluster untuk Aurora DSQL

Anda dapat membuat klaster Multi-wilayah Aurora DSQL dalam set Wilayah tertentu. AWS Setiap Wilayah menetapkan kelompok Wilayah yang terkait secara geografis yang dapat bekerja sama dalam klaster Multi-wilayah.

Wilayah AS

- Timur AS (N. Virginia)
- AS Timur (Ohio)
- AS Barat (Oregon)

Wilayah Asia Pasifik

- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)

- Asia Pasifik (Tokyo)

Wilayah Eropa

- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)

Batasan Penting

Multi-Region cluster harus dibuat dalam satu set Region. Misalnya, Anda tidak dapat membuat klaster yang mencakup Wilayah AS Timur (Virginia Utara) dan Eropa (Irlandia).

> **Important**
>
> Aurora DSQL saat ini tidak mendukung klaster Multi-wilayah lintas benua.

Harga untuk Aurora DSQL

Untuk informasi biaya, lihat harga Aurora DSQL.

Apa selanjutnya?

Untuk informasi tentang komponen inti di Aurora DSQL dan untuk memulai layanan, lihat berikut ini:

- Memulai dengan Aurora DSQL
- Kompatibilitas fitur SQL di Aurora DSQL
- Mengakses Aurora DSQL dengan klien PostgreSQL-compatible
- Aurora DSQL dan PostgreSQL

Memulai dengan Aurora DSQL

Amazon Aurora DSQL adalah database relasional tanpa server, dikelola sepenuhnya, dan terdistribusi yang dioptimalkan untuk beban kerja transaksional. Di bagian berikut, Anda akan belajar cara membuat kluster DSQL Aurora wilayah tunggal dan Multi-wilayah, menghubungkannya, dan membuat serta memuat skema sampel. Anda akan mengakses cluster dengan AWS Konsol dan secara opsional berinteraksi dengan database Anda menggunakan klien PostgreSQL lainnya. Pada akhirnya, Anda akan memiliki cluster Aurora DSQL yang berfungsi yang siap digunakan untuk beban kerja pengujian atau produksi.

Topik

- Prasyarat
- Langkah 1: Buat cluster Aurora DSQL Single-region
- Langkah 2: Connect ke cluster Aurora DSQL Anda
- Langkah 3: Jalankan contoh perintah SQL di Aurora DSQL
- Langkah 4 (Opsional): Buat cluster Multi-region
- Pemecahan masalah

Prasyarat

Sebelum Anda dapat mulai menggunakan Aurora DSQL, pastikan Anda memenuhi prasyarat berikut:

- Identitas IAM Anda harus memiliki izin untuk masuk ke konsol.
- Identitas IAM Anda harus memenuhi kriteria berikut:
  - Akses untuk melakukan tindakan apa pun pada sumber daya apa pun di Akun AWS
  - `AmazonAuroraDSQLConsoleFullAccess` AWS kebijakan terkelola terlampir.

Langkah 1: Buat cluster Aurora DSQL Single-region

Unit dasar Aurora DSQL adalah cluster, yang merupakan tempat Anda menyimpan data Anda. Dalam tugas ini, Anda membuat cluster dalam satu Wilayah AWS.

Untuk membuat cluster Single-region di Aurora DSQL

1. Masuk ke Konsol Manajemen AWS dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Pilih Buat cluster dan kemudian Single-Region.
3. (Opsional) mengubah nilai tag Nama default.
4. (Opsional) Tambahkan Tag tambahan untuk cluster ini.
5. (Opsional) Dalam pengaturan Cluster, pilih salah satu opsi berikut:
   - Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk memilih atau membuat AWS KMS key. Jika Anda menggunakan kunci yang dikelola pelanggan, pastikan bahwa kebijakan kunci memberikan Aurora DSQL izin yang diperlukan. Untuk informasi selengkapnya, lihat Kebijakan kunci untuk kunci yang dikelola pelanggan.
   - Pilih Aktifkan perlindungan penghapusan untuk mencegah operasi penghapusan klaster Anda. Secara default, perlindungan penghapusan dipilih.
   - Pilih Resource-based kebijakan (lanjutan) untuk menentukan kebijakan kontrol akses untuk klaster ini.
6. Pilih Buat klaster.
7. Konsol mengembalikan Anda ke halaman Clusters. Spanduk notifikasi muncul yang menunjukkan bahwa cluster sedang dibuat. Pilih ID Cluster untuk membuka tampilan detail cluster.

Langkah 2: Connect ke cluster Aurora DSQL Anda

Aurora DSQL mendukung beberapa cara untuk terhubung ke cluster Anda, termasuk DSQL Query Editor, klien psql lokal AWS CloudShell, dan alat lainnya. PostgreSQL-compatible Pada langkah ini, Anda terhubung menggunakan Aurora DSQL Query Editor, yang menyediakan cara cepat untuk mulai berinteraksi dengan cluster baru Anda.

Untuk terhubung menggunakan Query Editor

1. Di Aurora DSQL Console (https://console.aws.amazon.com/dsql), buka halaman Clusters dan konfirmasikan bahwa pembuatan cluster Anda telah selesai dan statusnya Aktif.
2. Pilih klaster Anda dari daftar, atau pilih ID Cluster untuk membuka halaman detail Cluster.
3. Pilih Connect with Query editor.

4. Pilih Connect sebagai admin untuk cluster yang baru saja dibuat.

   - Secara opsional Anda dapat terhubung dengan peran khusus, lihatMenggunakan peran database dan otentikasi IAM.

Langkah 3: Jalankan contoh perintah SQL di Aurora DSQL

Uji cluster Aurora DSQL Anda dengan menjalankan pernyataan SQL. Setelah membuka cluster di Query Editor, pilih dan jalankan setiap contoh query langkah demi langkah.

Jalankan contoh perintah SQL di Aurora DSQL

1. Buat skema bernama`test`.

```
CREATE SCHEMA IF NOT EXISTS test;
```

2. Buat tabel hello_world yang menggunakan UUID yang dihasilkan secara otomatis sebagai kunci utama.

```
CREATE TABLE IF NOT EXISTS test.hello_world (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    message VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Masukkan baris sampel.

```
INSERT INTO test.hello_world (message)
VALUES ('Hello, World!!');
```

4. Baca nilai yang dimasukkan.

```
SELECT * FROM test.hello_world;
```

5. Bersihkan secara opsional.

```
DROP TABLE test.hello_world;
DROP SCHEMA test;
```

Langkah 4 (Opsional): Buat cluster Multi-region

Saat Anda membuat cluster Multi-region, Anda menentukan Wilayah berikut:

Wilayah Terpencil

Ini adalah Wilayah tempat Anda membuat cluster kedua. Anda membuat cluster kedua di Region ini dan mengintip ke cluster awal Anda. Aurora DSQL mereplikasi semua penulisan pada cluster awal ke cluster jarak jauh. Anda dapat membaca dan menulis di cluster mana pun.

Wilayah Saksi

Wilayah ini menerima semua data yang ditulis ke cluster Multi-region. Namun, Wilayah saksi tidak meng-host titik akhir klien dan tidak menyediakan akses data pengguna. Jendela terbatas dari log transaksi terenkripsi dipertahankan di Wilayah saksi. Log ini memfasilitasi pemulihan dan mendukung kuorum transaksional jika suatu Wilayah menjadi tidak tersedia.

Gunakan prosedur berikut untuk membuat cluster awal, membuat cluster kedua di Region yang berbeda, dan kemudian mengintip dua cluster untuk membuat cluster Multi-region. Ini juga menunjukkan replikasi penulisan lintas wilayah dan pembacaan yang konsisten dari kedua titik akhir Regional.

Untuk membuat cluster Multi-region

1. Masuk ke konsol Aurora DSQL.
2. Pada panel navigasi, silakan pilih Klaster.
3. Pilih Buat cluster dan kemudian Multi-Region.
4. (Opsional) mengubah nilai tag Nama default.
5. (Opsional) Tambahkan Tag tambahan untuk cluster ini.
6. Di Multi-Region pengaturan, pilih opsi berikut untuk klaster awal Anda:
   - Di Wilayah Saksi, pilih Wilayah.
   - (Opsional) Di ARN cluster Remote Region, masukkan ARN untuk cluster yang ada di Wilayah lain. Jika tidak ada klaster yang berfungsi sebagai klaster kedua di klaster Multi-region Anda, selesaikan penyiapan setelah Anda membuat klaster awal.
7. (Opsional) Dalam pengaturan Cluster, pilih salah satu opsi berikut untuk klaster awal Anda:

- Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk memilih atau membuat AWS KMS key. Jika Anda menggunakan kunci yang dikelola pelanggan, pastikan bahwa kebijakan kunci memberikan Aurora DSQL izin yang diperlukan. Untuk informasi selengkapnya, lihat Kebijakan kunci untuk kunci yang dikelola pelanggan.
- Pilih Aktifkan perlindungan penghapusan untuk mencegah operasi penghapusan klaster Anda. Secara default, perlindungan penghapusan dipilih.
- Pilih Resource-based kebijakan (lanjutan) untuk menentukan kebijakan kontrol akses untuk klaster ini.

8. Pilih Buat cluster untuk membuat klaster awal Anda. Jika Anda tidak memasukkan ARN pada langkah sebelumnya, konsol akan menampilkan pemberitahuan tertunda pengaturan Cluster.
9. Dalam pemberitahuan tertunda penyiapan klaster, pilih Selesaikan pengaturan klaster Multi-wilayah. Tindakan ini memulai pembuatan cluster kedua di Wilayah lain.
10. Pilih salah satu opsi berikut untuk cluster kedua Anda:

    - Tambahkan ARN cluster Region jarak jauh — Pilih opsi ini jika ada cluster, dan Anda ingin itu menjadi cluster kedua di cluster Multi-region Anda.
    - Buat cluster di Wilayah lain - Pilih opsi ini untuk membuat cluster kedua. Di Remote Region, pilih Region untuk cluster kedua ini.

11. Pilih Buat cluster di ***your-second-region***, di *your-second-region* mana lokasi cluster kedua Anda. Konsol terbuka di Wilayah kedua Anda.
12. (Opsional) Pilih pengaturan cluster untuk cluster kedua Anda. Misalnya, Anda dapat memilih AWS KMS key. Jika Anda menggunakan kunci yang dikelola pelanggan, pastikan bahwa kebijakan kunci memberikan Aurora DSQL izin yang diperlukan. Untuk informasi selengkapnya, lihat Kebijakan kunci untuk kunci yang dikelola pelanggan.
13. Pilih Buat cluster untuk membuat cluster kedua Anda.
14. Pilih Peer in ***initial-cluster-region***, di *initial-cluster-region* mana Region yang menjadi tuan rumah cluster pertama yang Anda buat.
15. Saat diminta, pilih Konfirmasi. Langkah ini melengkapi pembuatan cluster Multi-region Anda.

Untuk terhubung ke cluster kedua Anda

1. Buka konsol Aurora DSQL dan pilih Region untuk cluster kedua Anda.
2. Pilih Klaster.
3. Pilih baris untuk cluster kedua di cluster Multi-region Anda.

4. Pilih Connect with Query editor.
5. Pilih Connect sebagai admin.
6. Buat skema dan tabel sampel, dan masukkan data dengan mengikuti langkah-langkahnya. Langkah 3: Jalankan contoh perintah SQL di Aurora DSQL

Untuk melakukan kueri data di klaster kedua dari Region yang menghosting klaster awal Anda

1. Di konsol Aurora DSQL, pilih Region untuk cluster awal Anda.
2. Pilih Klaster.
3. Pilih baris untuk cluster kedua di cluster Multi-region Anda.
4. Pilih Connect with Query editor.
5. Pilih Connect sebagai admin.
6. Kueri data yang Anda masukkan ke dalam cluster kedua.

   Example

   ```
   SELECT * FROM test.hello_world;
   ```

Pemecahan masalah

Lihat bagian Pemecahan Masalah pada dokumentasi Aurora DSQL.

Cara kerja penagihan di Aurora DSQL

Dengan Amazon Aurora DSQL, Anda hanya membayar untuk apa yang Anda gunakan tanpa biaya di muka. Bagian ini menjelaskan bagaimana Aurora DSQL mengukur aktivitas database Anda dan menerjemahkannya ke dalam tagihan Anda. AWS Untuk harga saat ini menurut Wilayah, lihat halaman harga Aurora DSQL.

Topik

- Cara kerja metering
- Pengukuran komponen DPU dijelaskan
- Multi-Region penagihan
- Memantau penggunaan DPU dengan CloudWatch
- Menggunakan EXPLORE ANALYZE VERBOSE untuk kesadaran biaya
- Praktik terbaik estimasi biaya

Cara kerja metering

Tidak seperti database tradisional yang mengenakan biaya untuk kapasitas yang disediakan, Aurora DSQL mengenakan biaya hanya untuk pekerjaan aktual yang dilakukan. Aurora DSQL meter dua komponen utama: aktivitas database, diukur dalam Unit Pemrosesan Terdistribusi (DPU), dan penyimpanan, diukur dalam. GiB-month

DPU mengukur berapa banyak pekerjaan yang dilakukan sistem untuk menjalankan beban kerja SQL Anda dan terdiri dari tiga komponen untuk cluster Single-region: Compute DPU, Read DPU, dan Write DPU. Multi-Region cluster mengeluarkan komponen MultiRegion Write DPU tambahan. Untuk detailnya, lihat Multi-Region penagihan.

Tabel berikut merangkum komponen yang Aurora DSQL gunakan untuk mengukur aktivitas database Anda. Pada tagihan Anda, Anda hanya melihat dua item baris: satu untuk penyimpanan dan satu untuk DPU, yang merupakan jumlah dari masing-masing komponen.

Unit pengukuran	Jenis aktivitas	Pengukuran
Hitung DPU	Pemrosesan kueri	Waktu CPU
Baca DPU	Membaca data dari database Anda	Byte dibaca dari penyimpanan

Unit pengukuran	Jenis aktivitas	Pengukuran
Tulis DPU	Menulis data ke database Anda	Byte ditulis ke penyimpanan
Penyimpanan	Penyimpanan meja	GiB-month

Pengukuran komponen DPU dijelaskan

Untuk setiap transaksi, Aurora DSQL menghitung total DPU sebagai jumlah dari tiga komponen: Compute DPU, Read DPU, dan Write DPU. Bagian berikut menjelaskan bagaimana Aurora DSQL meter setiap komponen.

```
Total DPU = ComputeDPU + ReadDPU + WriteDPU
```

ComputedPU

DPU komputasi diukur menggunakan total waktu pemrosesan yang dihabiskan untuk mengeksekusi kueri Anda, termasuk gabungan, fungsi, agregasi, penyortiran, dan perencanaan kueri. Karena bagian dari kueri Anda dapat diproses secara paralel, DPU Komputasi mencerminkan jumlah semua waktu pemrosesan — bukan waktu jam dinding kueri.

Rumus berikut merangkum cara menghitung DPU Komputasi:

```
ComputeDPU = Total Compute time (in seconds)
```

DitulisPU

Untuk setiap transaksi, Aurora DSQL mengukur Write DPU dengan total byte yang ditulis ke penyimpanan. Tulis DPU mencakup total data yang ditulis ke tabel dasar Anda serta indeks sekunder apa pun. Aurora DSQL menagih setiap baris yang ditulis ke tabel dasar Anda dan indeks sekunder yang lebih kecil dari 128 byte seolah-olah itu adalah 128 byte. Aurora DSQL menagih transaksi tulis yang menulis kurang dari 1.024 byte seolah-olah menulis 1.024 byte.

> **Note**
>
> Operasi tulis juga dikenakan biaya readDPU karena Aurora DSQL membaca indeks kunci utama untuk memverifikasi keunikan sebelum menulis.

Rumus berikut menunjukkan langkah-langkah untuk menghitung Write DPU:

Langkah 1: Hitung byte tertulis

```
Bytes Written = Sum of max(size of each row, 128 bytes) for all rows written
```

Langkah 2: Hitung WritedPU

```
WriteDPU = max(Bytes Written, 1024) × 0.00004883
```

bacaDPU

Untuk setiap transaksi, Aurora DSQL mengukur Read DPU dengan total byte yang dibaca dari penyimpanan. Baca DPU mencakup data yang dibaca dari tabel dasar Anda serta indeks sekunder apa pun.

Per-partition minimum: Aurora DSQL mengukur byte baca per partisi penyimpanan, bukan per baris. Jika permintaan baca ke partisi penyimpanan mengembalikan kurang dari 128 byte, Aurora DSQL membulatkannya hingga 128 byte. Misalnya, jika kueri Anda membaca dari 4 partisi — 200 byte dari satu partisi dan 50 byte dari masing-masing tiga partisi lainnya — tiga pembacaan 50 byte masing-masing dibulatkan menjadi 128 byte, menghasilkan total 200 + 128 + 128 + 128 = 584 byte yang ditagih.

Minimum transaksi: Aurora DSQL menagih transaksi baca yang berbunyi kurang dari 2.048 byte secara total seolah-olah membaca 2.048 byte.

Rumus berikut menunjukkan langkah-langkah untuk menghitung Baca DPU:

Langkah 1: Hitung byte yang dibaca

```
Bytes Read = # of rows read × size of each row
```

> **Note**
>
> Pembacaan byte sebenarnya tergantung pada bagaimana data Anda didistribusikan di seluruh partisi penyimpanan, karena minimum 128-byte per partisi diterapkan per partisi. Jika semua ukuran baris Anda di atas 128 byte, Anda cukup mengalikan jumlah baris yang dibaca dengan ukuran setiap baris.

Langkah 2: Hitung ReadDPU

```
ReadDPU = max(Bytes Read, 2048) × 0.00000183105
```

Contoh penagihan

Contoh berikut menunjukkan bagaimana Aurora DSQL menghitung DPU untuk operasi umum. Nilai biaya dalam contoh ini menggunakan harga Wilayah us-east-1. Untuk harga di Wilayah lain, lihat halaman harga Aurora DSQL.

Contoh: Pencarian titik sederhana (baca)

Contoh ini menunjukkan pencarian titik perhitungan ReadDPU di mana minimum transaksi berlaku.

Skema:

```
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id VARCHAR(50) NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(20)
);
-- Average row size: ~100 bytes
```

Permintaan:

```
SELECT * FROM orders WHERE customer_id = 'cust-12345';
```

Skenario: Query mengembalikan 5 baris, masing-masing sekitar 100 byte. Dengan asumsi semua baris berada dalam satu partisi penyimpanan, total byte yang dibaca adalah 5 × 100 = 500 byte. Karena 500 byte melebihi minimum 128-byte per partisi, tidak ada minimum per partisi yang berlaku.

Hitung ReadDPU:

```
ReadDPU = max(500, 2048) × 0.00000183105 = 2048 × 0.00000183105 = 0.00375
```

Minimal transaksi 2.048 byte berlaku sejak 500 < 2.048.

Total biaya transaksi:

Dengan asumsi waktu eksekusi kueri 3 ms (0,003 detik):

```
ComputeDPU: 0.003
ReadDPU:    0.00375
WriteDPU:   0.0
-------------------
Total DPU:  0.00675
```

Contoh: Pemindaian rentang yang difilter (baca)

Skema:

```
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id VARCHAR(50) NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(20)
);
-- Average row size: ~100 bytes
-- Table contains 100 orders for customer 'cust-12345'
```

Permintaan:

```
SELECT * FROM orders
WHERE customer_id = 'cust-12345'
  AND total_amount > 500.00;
```

Skenario: Kueri memindai 100 baris untuk pelanggan 'cust-12345', tetapi `total_amount > 500.00` filter mengurangi hasilnya menjadi hanya 10 baris yang dikembalikan. Tagihan Aurora DSQL untuk semua 100 baris yang dipindai. Dengan asumsi semua baris berada dalam satu partisi penyimpanan, total byte yang dibaca adalah 100 × 100 = 10.000 byte.

Hitung ReadDPU:

```
ReadDPU = max(10000, 2048) × 0.00000183105 = 10000 × 0.00000183105 = 0.01831
```

Karena 10.000 byte melebihi minimum transaksi 2.048-byte, byte yang sebenarnya dibaca digunakan.

Total biaya transaksi:

Dengan asumsi waktu eksekusi kueri 8 ms (0,008 detik):

```
ComputeDPU: 0.008
ReadDPU:    0.01831
WriteDPU:   0.0
-------------------
Total DPU:  0.02631
```

> **Important**
>
> Untuk meminimalkan biaya readDPU, desain kueri dan indeks untuk memindai hanya baris yang Anda butuhkan. Dalam contoh ini, menambahkan indeks pada `(customer_id, total_amount)` dapat memungkinkan kueri untuk memindai lebih sedikit baris.

Contoh: Sisipan tunggal (baca dan tulis)

Skema:

```
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id VARCHAR(50) NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(20)
);
-- Average row size: ~100 bytes
```

Permintaan:

```
INSERT INTO orders (customer_id, total_amount, status)
VALUES ('cust-67890', 150.00, 'pending');
```

Skenario: Sisipkan 1 baris, sekitar 100 byte.

Perhitungan WritedPU:

Langkah 1 - Hitung byte yang ditulis:

```
1 row × max(100 bytes, 128 bytes) = 1 × 128 = 128 bytes
```

Langkah 2 - Hitung WritedPU:

```
WriteDPU = max(128, 1024) × 0.00004883 = 1024 × 0.00004883 = 0.05
```

Minimal transaksi 1.024 byte berlaku sejak 128 < 1.024.

ReadDPU (pemeriksaan kunci utama):

Aurora DSQL membaca indeks kunci utama untuk memverifikasi keunikan sebelum menulis. Ini menimbulkan biaya baca minimum transaksi.

```
ReadDPU = 0.00375 (transaction minimum)
```

Total biaya transaksi:

Dengan asumsi waktu eksekusi kueri 8 ms (0,008 detik):

```
ComputeDPU: 0.008
ReadDPU:    0.00375
WriteDPU:   0.05
-------------------
Total DPU:  0.06175
```

Contoh: Sisipan massal (baca dan tulis)

Skema:

```
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id VARCHAR(50) NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(20)
);
-- Average row size: ~100 bytes
```

Permintaan:

```
INSERT INTO orders (customer_id, total_amount, status)
VALUES
    ('cust-001', 100.00, 'pending'),
    ('cust-002', 150.00, 'pending'),
```

```
    ... -- 100 rows total
    ('cust-100', 200.00, 'pending');
```

Skenario: Masukkan 100 baris, masing-masing sekitar 100 byte.

Perhitungan WritedPU:

Langkah 1 - Hitung byte yang ditulis:

```
100 rows × max(100 bytes, 128 bytes) = 100 × 128 = 12,800 bytes
```

Langkah 2 - Hitung WritedPU:

```
WriteDPU = max(12800, 1024) × 0.00004883 = 12800 × 0.00004883 = 0.625
```

ReadDPU (pemeriksaan kunci utama):

Aurora DSQL membaca indeks kunci utama untuk setiap baris untuk memverifikasi keunikan. Dengan asumsi semua 100 pencarian kunci berada dalam satu partisi penyimpanan, total byte yang dibaca adalah 100 × 16 byte (UUID) = 1.600 byte:

```
ReadDPU = max(1600, 2048) × 0.00000183105 = 2048 × 0.00000183105 = 0.00375
```

Minimal transaksi 2.048 byte berlaku sejak 1.600 < 2.048.

Total biaya transaksi:

Dengan asumsi waktu eksekusi kueri 80 ms (0,08 detik):

```
ComputeDPU: 0.08
ReadDPU:    0.00375
WriteDPU:   0.625
-------------------
Total DPU:  0.70875
```

Multi-Region penagihan

Multi-Region cluster mengeluarkan komponen MultiRegion Write DPU tambahan di atas DPU Compute, Read, dan Write standar. Bagian ini hanya berlaku untuk klaster Multi-wilayah. Single-Region cluster tidak dikenakan biaya ini.

MultiRegion Tulis DPU mengukur total byte yang ditulis ke Wilayah peered. Karena Aurora DSQL secara sinkron mereplikasi data yang Anda tulis ke Wilayah peered, nilai Write DPU setara dengan MultiRegion Write DPU. Aurora DSQL membebankan DPU ini di Wilayah tempat penulisan berasal, bukan di Wilayah yang diintip.

```
MultiRegionWriteDPU = WriteDPU
```

Memantau penggunaan DPU dengan CloudWatch

Aurora DSQL menerbitkan metrik penggunaan ke Amazon CloudWatch, memungkinkan Anda memantau konsumsi dalam waktu dekat.

Metrik DPU yang tersedia

Metrik DPU

CloudWatch metrik	Deskripsi	Dimensi
`WriteDPU`	Tulis komponen penggunaan	ClusterId
`ReadDPU`	Baca komponen penggunaan	ClusterId
`ComputeDPU`	Komponen pemrosesan kueri	ClusterId
`MultiRegionWriteDPU`	Multi-Region replikasi (Cluster multi-wil ayah saja)	ClusterId
`TotalDPU`	Jumlah semua komponen DPU	ClusterId

Melihat metrik DPU

Untuk melihat metrik DPU di CloudWatch

1. Buka konsol CloudWatch.
2. Arahkan ke Metrik, lalu AuroraDSql. ClusterId
3. Pilih klaster Anda dan metrik DPU yang ingin Anda pantau.

> **Tip**
>
> Gunakan statistik Jumlah untuk metrik DPU untuk melihat penggunaan total selama periode waktu tertentu. Tambahkan label LAST untuk melihat nilai terbaru.

Metrik observabilitas tambahan

Untuk daftar lengkap metrik Aurora DSQL dan kemampuan pemantauan, lihat. Pencatatan dan pemantauan

Metrik observabilitas

Metrik	Deskripsi
`ClusterStorageSize`	Ukuran penyimpanan saat ini dalam byte
`TotalTransactions`	Total transaksi yang dieksekusi
`ReadOnlyTransactions`	Read-only transaksi dieksekusi
`QueryTimeouts`	Kueri yang melebihi batas waktu
`OccConflicts`	Transaksi dibatalkan karena konflik OCC
`BytesWritten`	Byte mentah ditulis ke penyimpanan
`BytesRead`	Byte mentah dibaca dari penyimpanan

Menggunakan EXPLORE ANALYZE VERBOSE untuk kesadaran biaya

Aurora DSQL diperluas `EXPLAIN ANALYZE VERBOSE` untuk menyertakan perkiraan penggunaan DPU tingkat pernyataan pada akhir output. Ini memberikan visibilitas langsung ke biaya kueri, membantu Anda mengidentifikasi driver biaya beban kerja, menyesuaikan kinerja kueri, dan perkiraan penggunaan sumber daya yang lebih baik.

> **Note**
>
> Anda harus menggunakan `EXPLAIN ANALYZE VERBOSE` (dengan VERBOSE) untuk melihat perkiraan DPU. Polos `EXPLAIN ANALYZE` tanpa VERBOSE tidak menampilkan informasi DPU.

Contoh 1: SELECT query

```
EXPLAIN ANALYZE VERBOSE SELECT * FROM test_table;
```

```
QUERY PLAN
-----------------------------------------------------
Index Only Scan using test_table_pkey on public.test_table  (cost=125100.05..171100.05
 rows=1000000 width=36) (actual time=2.973..4.482 rows=120 loops=1)
  Output: id, context
  -> Storage Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000 width=36)
 (actual rows=120 loops=1)
       Projections: id, context
       -> B-Tree Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000
 width=36) (actual rows=120 loops=1)
Query Identifier: qymgw1m77maoe
Planning Time: 11.415 ms
Execution Time: 4.528 ms
Statement DPU Estimate:
  Compute: 0.01607 DPU
  Read: 0.04312 DPU
  Write: 0.00000 DPU
  Total: 0.05919 DPU
```

Dalam contoh ini, pernyataan SELECT melakukan pemindaian indeks saja, sehingga sebagian besar biaya berasal dari Baca DPU (0,04312), mewakili data yang diambil dari penyimpanan dan Compute DPU (0,01607), yang mencerminkan sumber daya komputasi yang digunakan untuk memproses dan mengembalikan hasilnya. Tidak ada Write DPU karena kueri tidak mengubah data. Total DPU (0,05919) adalah jumlah dari Compute + Read+Write.

Contoh 2: Insert query

```
EXPLAIN ANALYZE VERBOSE INSERT INTO test_table VALUES (1, 'name1'), (2, 'name2'), (3,
 'name3');
```

```
QUERY PLAN
-----------------------------------------------------
Insert on public.test_table  (cost=0.00..0.04 rows=0 width=0) (actual time=0.055..0.056
 rows=0 loops=1)
  ->  Values Scan on "*VALUES*"  (cost=0.00..0.04 rows=3 width=122) (actual
 time=0.003..0.008 rows=3 loops=1)
       Output: "*VALUES*".column1, "*VALUES*".column2
Query Identifier: jtkjkexhjotbo
Planning Time: 0.068 ms
Execution Time: 0.543 ms
Statement DPU Estimate:
  Compute: 0.01550 DPU
  Read: 0.00307 DPU (Transaction minimum: 0.00375)
  Write: 0.01875 DPU (Transaction minimum: 0.05000)
  Total: 0.03732 DPU
```

Pernyataan ini terutama melakukan penulisan, sehingga sebagian besar biaya dikaitkan dengan Tulis DPU. Compute DPU (0,01550) mewakili pekerjaan yang dilakukan untuk memproses dan menyisipkan nilai. DPU Baca (0,00307) mencerminkan pembacaan sistem minor (untuk pencarian katalog atau pemeriksaan indeks).

Perhatikan minimum transaksi yang ditunjukkan dalam tanda kurung di sebelah Baca dan Tulis DPU. Minimum ini berlaku pada tingkat transaksi, artinya total DPU Baca atau Tulis untuk seluruh transaksi tidak pernah kurang dari nilai-nilai ini. Jika Anda menggunakan `EXPLAIN ANALYZE VERBOSE` untuk memperkirakan biaya dan ini adalah satu-satunya pernyataan dalam transaksi, gunakan nilai minimum transaksi daripada perkiraan pernyataan mentah. Jika transaksi berisi beberapa pernyataan, minimum berlaku untuk agregat di semua pernyataan. Karena `EXPLAIN ANALYZE VERBOSE` melaporkan perkiraan tingkat pernyataan saat penagihan berlaku minimum tingkat transaksi, nilainya mungkin tidak sama persis dengan metrik atau data penagihan. CloudWatch

Menggunakan informasi DPU untuk optimasi

Per-statement Perkiraan DPU memberi Anda cara yang ampuh untuk mengoptimalkan kueri di luar waktu eksekusi. Kasus penggunaan umum meliputi:

- Kesadaran biaya: Memahami seberapa mahal kueri relatif terhadap orang lain.
- Optimasi skema: Bandingkan dampak indeks atau perubahan skema pada kinerja dan efisiensi sumber daya.
- Perencanaan anggaran: Perkirakan biaya beban kerja berdasarkan penggunaan DPU yang diamati.
- Perbandingan kueri: Evaluasi pendekatan kueri alternatif dengan konsumsi DPU relatifnya.

Menafsirkan informasi DPU

Ingatlah praktik terbaik berikut saat menggunakan data DPU dari`EXPLAIN ANALYZE VERBOSE`:

- Gunakan secara terarah: Perlakukan DPU yang dilaporkan sebagai cara untuk memahami biaya relatif kueri daripada kecocokan persis dengan CloudWatch metrik atau data penagihan. Perbedaan diharapkan karena `EXPLAIN ANALYZE VERBOSE` melaporkan biaya tingkat pernyataan, sementara CloudWatch agregat aktivitas tingkat transaksi. CloudWatch juga mencakup operasi latar belakang (seperti ANALISIS asinkron atau pemadatan) dan overhead transaksi (`BEGIN`/`COMMIT`) yang sengaja dikecualikan. `EXPLAIN ANALYZE VERBOSE`
- Uji dengan data representatif untuk bukti konsep: Saat menjalankan bukti konsep untuk mengevaluasi biaya, pastikan tabel Anda berisi volume data dan distribusi yang serupa dengan beban kerja produksi yang Anda harapkan. Perkiraan DPU — baik dari `EXPLAIN ANALYZE VERBOSE` atau CloudWatch metrik — yang didasarkan pada tabel kosong atau jarang penduduknya tidak akan mencerminkan biaya dunia nyata.
- Variabilitas DPU di seluruh proses normal dalam sistem terdistribusi dan tidak menunjukkan kesalahan. Faktor-faktor seperti caching, perubahan rencana eksekusi, konkurensi, operasi latar belakang seperti analisis asinkron, atau pergeseran dalam distribusi data semuanya dapat menyebabkan kueri yang sama mengkonsumsi sumber daya yang berbeda dari satu proses ke yang berikutnya.
- Operasi kecil Batch: Jika beban kerja Anda mengeluarkan banyak pernyataan kecil, pertimbangkan untuk mengelompokkannya ke dalam operasi penulisan yang lebih besar dalam satu transaksi (modifikasi tidak boleh melebihi 10 MB per transaksi, meskipun pembacaan hanya dibatasi oleh batas waktu transaksi 5 menit). Ini mengamortisasi minimum transaksi di lebih banyak pekerjaan dan menghasilkan perkiraan biaya yang lebih berarti.
- Gunakan untuk penyetelan, bukan penagihan: Data DPU `EXPLAIN ANALYZE VERBOSE` dirancang untuk kesadaran biaya, penyetelan kueri, dan pengoptimalan. Ini bukan metrik tingkat penagihan.

Selalu mengandalkan CloudWatch metrik atau laporan penagihan bulanan untuk biaya otoritatif dan data penggunaan.

Praktik terbaik estimasi biaya

- Pantau sebelum mengoptimalkan: Gunakan CloudWatch metrik untuk memahami pola penggunaan Anda saat ini sebelum membuat keputusan pengoptimalan. Lihat perinciannya di the section called “Memantau penggunaan DPU”.
- Fokus pada efisiensi transaksi: Karena minimum berlaku pada tingkat transaksi, operasi terkait batch bersama-sama untuk mengamortisasi biaya minimum.
- Gunakan EXPLORE ANALYZE VERBOSE selama pengembangan: Jalankan `EXPLAIN ANALYZE VERBOSE` pada kueri kritis selama pengembangan untuk memahami karakteristik biayanya. Saat menjalankan bukti konsep untuk mengevaluasi biaya, uji terhadap tabel dengan volume dan distribusi data yang representatif — perkiraan berdasarkan tabel kosong atau jarang penduduknya tidak akan mencerminkan biaya produksi. Lihat perinciannya di the section called “JELASKAN ANALISIS VERBOSE untuk kesadaran biaya”.
- Setel CloudWatch alarm: Buat alarm pada metrik DPU untuk mendapatkan pemberitahuan tentang lonjakan penggunaan yang tidak terduga.

Otentikasi dan otorisasi untuk Aurora DSQL

Aurora DSQL menggunakan peran dan kebijakan IAM untuk otorisasi klaster. Anda mengaitkan peran IAM dengan peran database PostgreSQL untuk otorisasi database. Pendekatan ini menggabungkan manfaat dari IAM dengan hak istimewa PostgreSQL. Aurora DSQL menggunakan fitur-fitur ini untuk memberikan otorisasi komprehensif dan kebijakan akses untuk cluster, database, dan data Anda.

Mengelola cluster Anda menggunakan IAM

Untuk mengelola klaster Anda, gunakan IAM untuk otentikasi dan otorisasi:

Autentikasi IAM

Untuk mengautentikasi identitas IAM Anda ketika Anda mengelola cluster Aurora DSQL, Anda harus menggunakan IAM. Anda dapat memberikan otentikasi menggunakan Konsol Manajemen AWS, AWS CLI, atau AWS SDK.

Otorisasi IAM

Untuk mengelola cluster Aurora DSQL, berikan otorisasi menggunakan tindakan IAM untuk Aurora DSQL. Misalnya, untuk mendeskripsikan klaster, pastikan identitas IAM Anda memiliki izin untuk tindakan IAM`dsql:GetCluster`, seperti pada contoh tindakan kebijakan berikut.

```
{
  "Effect": "Allow",
  "Action": "dsql:GetCluster",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

Untuk informasi selengkapnya, lihat Menggunakan tindakan kebijakan IAM untuk mengelola cluster.

Menghubungkan ke klaster Anda menggunakan IAM

Untuk terhubung ke cluster Anda, gunakan IAM untuk otentikasi dan otorisasi:

Autentikasi IAM

Buat token otentikasi sementara menggunakan identitas IAM dengan otorisasi untuk terhubung ke cluster Anda. Untuk mempelajari selengkapnya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

Otorisasi IAM

Berikan tindakan kebijakan IAM berikut ke identitas IAM yang Anda gunakan untuk membuat koneksi ke titik akhir klaster Anda:

- Gunakan `dsql:DbConnectAdmin` jika Anda menggunakan `admin` peran. Aurora DSQL membuat dan mengelola peran ini untuk Anda. Contoh tindakan kebijakan IAM berikut memungkinkan `admin` untuk terhubung ke. *`my-cluster`*

```
{
  "Effect": "Allow",
  "Action": "dsql:DbConnectAdmin",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

- Gunakan `dsql:DbConnect` jika Anda menggunakan peran database kustom. Anda membuat dan mengelola peran ini dengan menggunakan perintah SQL dalam database Anda. Contoh tindakan kebijakan IAM berikut memungkinkan peran database kustom untuk terhubung hingga *`my-cluster`* satu jam.

```
{
  "Effect": "Allow",
  "Action": "dsql:DbConnect",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

Setelah Anda membuat koneksi, peran Anda diizinkan hingga satu jam untuk koneksi.

Berinteraksi dengan database Anda menggunakan peran database PostgreSQL dan peran IAM

PostgreSQL mengelola izin akses database menggunakan konsep peran. Peran dapat dianggap sebagai pengguna database, atau sekelompok pengguna database, tergantung pada bagaimana

peran tersebut diatur. Anda membuat peran PostgreSQL menggunakan perintah SQL. Untuk mengelola otorisasi tingkat database, berikan izin PostgreSQL ke peran database PostgreSQL Anda.

Aurora DSQL mendukung dua jenis peran database: peran dan `admin` peran khusus. Aurora DSQL secara otomatis membuat `admin` peran yang telah ditentukan untuk Anda di cluster Aurora DSQL Anda. Anda tidak dapat mengubah `admin` peran. Ketika Anda terhubung ke database Anda sebagai`admin`, Anda dapat mengeluarkan SQL untuk membuat peran tingkat database baru untuk dikaitkan dengan peran IAM Anda. Untuk membiarkan peran IAM terhubung ke database Anda, kaitkan peran database kustom Anda dengan peran IAM Anda.

Autentikasi

Gunakan `admin` peran untuk terhubung ke cluster Anda. Setelah Anda menghubungkan database Anda, gunakan perintah `AWS IAM GRANT` untuk mengaitkan peran database kustom dengan identitas IAM yang diotorisasi untuk terhubung ke cluster, seperti pada contoh berikut.

```
AWS IAM GRANT custom-db-role TO 'arn:aws:iam::account-id:role/iam-role-name';
```

Untuk mempelajari selengkapnya, lihat Mengotorisasi peran database untuk terhubung ke klaster Anda.

Otorisasi

Gunakan `admin` peran untuk terhubung ke cluster Anda. Jalankan perintah SQL untuk mengatur peran database kustom dan memberikan izin. Untuk mempelajari selengkapnya, lihat peran database PostgreSQL dan hak istimewa PostgreSQL dalam dokumentasi PostgreSQL.

Menggunakan tindakan kebijakan IAM dengan Aurora DSQL

Tindakan kebijakan IAM yang Anda gunakan bergantung pada peran yang Anda gunakan untuk menyambung ke klaster: salah satu `admin` atau peran basis data kustom. Kebijakan ini juga tergantung pada tindakan IAM yang diperlukan untuk peran ini.

Menggunakan tindakan kebijakan IAM untuk terhubung ke cluster

Saat Anda terhubung ke klaster Anda dengan peran database default`admin`, gunakan identitas IAM dengan otorisasi untuk melakukan tindakan kebijakan IAM berikut.

```
"dsql:DbConnectAdmin"
```

Saat Anda terhubung ke klaster Anda dengan peran database kustom, pertama-tama kaitkan peran IAM dengan peran database. Identitas IAM yang Anda gunakan untuk terhubung ke klaster Anda harus memiliki otorisasi untuk melakukan tindakan kebijakan IAM berikut.

```
"dsql:DbConnect"
```

Untuk mempelajari lebih lanjut tentang peran basis data kustom, lihatMenggunakan peran database dan otentikasi IAM.

Menggunakan tindakan kebijakan IAM untuk mengelola cluster

Saat mengelola klaster Aurora DSQL Anda, tentukan tindakan kebijakan hanya untuk tindakan yang perlu dilakukan peran Anda. Misalnya, jika peran Anda hanya perlu mendapatkan informasi kluster, Anda dapat membatasi izin peran hanya untuk `ListClusters` izin `GetCluster` dan, seperti dalam kebijakan contoh berikut

JSON

```
{
"Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
    }
  ]
}
```

Contoh kebijakan berikut menunjukkan semua tindakan kebijakan IAM yang tersedia untuk mengelola cluster.

JSON

```
{
```

```
"Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:UpdateCluster",
        "dsql:DeleteCluster",
        "dsql:ListClusters",
        "dsql:TagResource",
        "dsql:ListTagsForResource",
        "dsql:UntagResource"
      ],
      "Resource" : "*"
    }
  ]
}
```

Mencabut otorisasi menggunakan IAM dan PostgreSQL

Anda dapat mencabut izin untuk peran IAM untuk mengakses peran tingkat database Anda:

Mencabut otorisasi admin untuk terhubung ke cluster

Untuk mencabut otorisasi untuk terhubung ke klaster Anda dengan `admin` peran tersebut, cabut akses identitas IAM ke. `dsql:DbConnectAdmin` Edit kebijakan IAM atau lepaskan kebijakan dari identitas.

Setelah mencabut otorisasi koneksi dari identitas IAM, Aurora DSQL menolak semua upaya koneksi baru dari identitas IAM tersebut. Koneksi aktif apa pun yang menggunakan identitas IAM mungkin tetap diotorisasi selama durasi koneksi. Untuk informasi selengkapnya tentang durasi koneksi, lihat Kuota dan batas.

Mencabut otorisasi peran khusus untuk terhubung ke cluster

Untuk mencabut akses ke peran database selain`admin`, cabut akses identitas IAM ke. `dsql:DbConnect` Edit kebijakan IAM atau lepaskan kebijakan dari identitas.

Anda juga dapat menghapus asosiasi antara peran database dan IAM dengan menggunakan perintah `AWS IAM REVOKE` dalam database Anda. Untuk mempelajari lebih lanjut tentang mencabut akses dari peran database, lihat. Mencabut otorisasi database dari peran IAM

Anda tidak dapat mengelola izin dari peran `admin` database yang telah ditentukan sebelumnya. Untuk mempelajari cara mengelola izin untuk peran database kustom, lihat hak istimewa PostgreSQL. Modifikasi hak istimewa berlaku pada transaksi berikutnya setelah Aurora DSQL berhasil melakukan transaksi modifikasi.

Menghasilkan token otentikasi di Amazon Aurora DSQL

Untuk terhubung ke Amazon Aurora DSQL dengan klien SQL, buat token otentikasi untuk digunakan sebagai kata sandi. Token ini hanya digunakan untuk mengautentikasi koneksi. Setelah koneksi dibuat, koneksi tetap valid bahkan jika token otentikasi kedaluwarsa.

Jika Anda membuat token otentikasi menggunakan AWS konsol, SDK, atau SDK AWS CLI, token secara otomatis kedaluwarsa dalam 15 menit secara default. Durasi maksimum adalah 604.800 detik, yaitu satu minggu. Untuk terhubung ke Aurora DSQL dari klien Anda lagi, Anda dapat menggunakan token otentikasi yang sama jika belum kedaluwarsa, atau Anda dapat membuat yang baru.

Untuk memulai membuat token, buat kebijakan IAM dan cluster di Aurora DSQL. Kemudian gunakan AWS konsol, AWS CLI, atau AWS SDK untuk menghasilkan token.

Minimal, Anda harus memiliki izin IAM yang tercantumMenghubungkan ke klaster Anda menggunakan IAM, tergantung pada peran database yang Anda gunakan untuk terhubung.

Topik

- Gunakan AWS konsol untuk menghasilkan token otentikasi di Aurora DSQL
- Gunakan AWS CloudShell untuk menghasilkan token otentikasi di Aurora DSQL
- Gunakan AWS CLI untuk menghasilkan token otentikasi di Aurora DSQL
- Gunakan SDK untuk menghasilkan token di Aurora DSQL

Gunakan AWS konsol untuk menghasilkan token otentikasi di Aurora DSQL

Aurora DSQL mengautentikasi pengguna dengan token daripada kata sandi. Anda dapat menghasilkan token dari konsol.

Untuk menghasilkan token otentikasi

1. Masuk ke Konsol Manajemen AWS dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql

2. Pilih ID cluster cluster yang ingin Anda buat token otentikasi. Jika Anda belum membuat cluster, ikuti langkah-langkah di Langkah 1: Buat cluster Aurora DSQL Single-region atauLangkah 4 (Opsional): Buat cluster Multi-region.
3. Pilih Connect dan kemudian pilih Get Token.
4. Pilih apakah Anda ingin terhubung sebagai `admin` atau dengan peran database kustom.
5. Salin token otentikasi yang dihasilkan dan gunakan untukAkses Aurora DSQL menggunakan klien SQL.

Untuk mempelajari lebih lanjut tentang peran basis data kustom dan IAM di Aurora DSQL, lihat. Autentikasi dan otorisasi

Gunakan AWS CloudShell untuk menghasilkan token otentikasi di Aurora DSQL

Sebelum Anda dapat membuat token otentikasi menggunakan AWS CloudShell, pastikan bahwa Anda membuat cluster Aurora DSQL.

Untuk menghasilkan token otentikasi menggunakan AWS CloudShell

1. Masuk ke Konsol Manajemen AWS dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Di kiri bawah AWS konsol, pilih AWS CloudShell.
3. Jalankan perintah berikut untuk menghasilkan token otentikasi untuk `admin` peran tersebut. Ganti *`us-east-1`* dengan Wilayah Anda dan *`your_cluster_endpoint`* dengan titik akhir cluster Anda sendiri.

> **Note**
>
> Jika Anda tidak terhubung sebagai`admin`, gunakan sebagai `generate-db-connect-auth-token` gantinya.

```
aws dsql generate-db-connect-admin-auth-token \
  --expires-in 3600 \
  --region us-east-1 \
  --hostname your_cluster_endpoint
```

Jika Anda mengalami masalah, lihat Memecahkan Masalah IAM dan Bagaimana cara memecahkan masalah akses ditolak atau kesalahan operasi yang tidak sah dengan kebijakan IAM? .

4. Gunakan perintah berikut untuk digunakan `psql` untuk memulai koneksi ke cluster Anda.

```
PGSSLMODE=require \
psql --dbname postgres \
  --username admin \
  --host cluster_endpoint
```

5. Anda akan melihat prompt untuk memberikan kata sandi. Salin token yang Anda buat, dan pastikan Anda tidak menyertakan spasi atau karakter tambahan. Tempelkan ke prompt berikut dari`psql`.

```
Password for user admin:
```

6. Tekan Enter. Anda akan melihat prompt PostgreSQL.

```
postgres=>
```

Jika Anda mendapatkan kesalahan akses ditolak, pastikan identitas IAM Anda memiliki `dsql:DbConnectAdmin` izin. Jika Anda memiliki izin dan terus mendapatkan kesalahan penolakan akses, lihat Memecahkan masalah IAM dan Bagaimana saya bisa memecahkan masalah kesalahan operasi yang ditolak atau tidak sah dengan kebijakan IAM? .

Untuk mempelajari lebih lanjut tentang peran basis data kustom dan IAM di Aurora DSQL, lihat. Autentikasi dan otorisasi

Gunakan AWS CLI untuk menghasilkan token otentikasi di Aurora DSQL

Ketika cluster Anda`ACTIVE`, Anda dapat menghasilkan token otentikasi pada CLI dengan menggunakan `aws dsql` perintah. Gunakan salah satu dari teknik berikut:

> **Note**
>
> Pembuatan Token adalah operasi lokal yang menandatangani permintaan menggunakan kredenal IAM Anda saat ini. Itu tidak menghubungi AWS untuk memvalidasi kredensil. Jika

kredensil Anda kedaluwarsa atau tidak valid, pembuatan token masih berhasil, tetapi upaya koneksi gagal. Pastikan kredensi IAM Anda valid sebelum membuat token.

- Jika Anda terhubung dengan `admin` peran tersebut, gunakan `generate-db-connect-admin-auth-token` opsi.
- Jika Anda terhubung dengan peran database kustom, gunakan `generate-db-connect-auth-token` opsi.

Contoh berikut menggunakan atribut berikut untuk menghasilkan token otentikasi untuk `admin` peran tersebut.

- *`your_cluster_endpoint`*— Titik akhir cluster. Ini mengikuti format*`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, seperti pada contoh`01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*— The Wilayah AWS, seperti `us-east-2` atau`us-east-1`.

Contoh berikut mengatur waktu kedaluwarsa token untuk kedaluwarsa dalam 3600 detik (1 jam).

Linux and macOS

```
aws dsql generate-db-connect-admin-auth-token \
  --region region \
  --expires-in 3600 \
  --hostname your_cluster_endpoint
```

Windows

```
aws dsql generate-db-connect-admin-auth-token ^
  --region=region ^
  --expires-in=3600 ^
  --hostname=your_cluster_endpoint
```

Gunakan SDK untuk menghasilkan token di Aurora DSQL

Anda dapat membuat token otentikasi untuk klaster Anda saat berada dalam `ACTIVE` status. Contoh SDK menggunakan atribut berikut untuk menghasilkan token otentikasi untuk peran tersebut`admin`:

- *your_cluster_endpoint*(atau*yourClusterEndpoint*) — Titik akhir cluster Aurora DSQL Anda. Format penamaan adalah*your_cluster_identifier*.dsql.*region*.on.aws, seperti pada contoh01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws.
- *region*(atau*RegionEndpoint*) — Wilayah AWS Di mana cluster Anda berada, seperti us-east-2 atauus-east-1.

Python SDK

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk Python, yang menangani pembuatan token secara otomatis.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan admin peran, gunakangenerate_db_connect_admin_auth_token.
- Jika Anda terhubung dengan peran basis data kustom, gunakangenerate_connect_auth_token.

```
import boto3

def generate_token(your_cluster_endpoint, region):
    client = boto3.client("dsql", region_name=region)
    # use `generate_db_connect_auth_token` instead if you are not connecting as
 admin.
    token = client.generate_db_connect_admin_auth_token(your_cluster_endpoint,
 region)
    print(token)
    return token
```

C++ SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan admin peran, gunakanGenerateDBConnectAdminAuthToken.

- Jika Anda terhubung dengan peran basis data kustom, gunakan`GenerateDBConnectAuthToken`.

```
#include <aws/core/Aws.h>
#include <aws/dsql/DSQLClient.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;

std::string generateToken(String yourClusterEndpoint, String region) {
    DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQLClient client{clientConfig};
    std::string token = "";

    // If you are not using the admin role to connect, use
 GenerateDBConnectAuthToken instead
    const auto presignedString =
 client.GenerateDBConnectAdminAuthToken(yourClusterEndpoint, region);
    if (presignedString.IsSuccess()) {
        token = presignedString.GetResult();
    } else {
        std::cerr << "Token generation failed." << std::endl;
    }

    std::cout << token << std::endl;
    return token;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    // Replace with your cluster endpoint and region
    std::string token = generateToken("your_cluster_endpoint.dsql.us-east-1.on.aws",
 "us-east-1");
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript SDK

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk Node.js, yang menangani pembuatan token secara otomatis.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`getDbConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`getDbConnectAuthToken`.

```
import { DsqlSigner } from "@aws-sdk/dsql-signer";

async function generateToken(yourClusterEndpoint, region) {
  const signer = new DsqlSigner({
    hostname: yourClusterEndpoint,
    region,
  });
  try {
    // Use `getDbConnectAuthToken` if you are _not_ logging in as the `admin` user
    const token = await signer.getDbConnectAdminAuthToken();
    console.log(token);
    return token;
  } catch (error) {
      console.error("Failed to generate token: ", error);
      throw error;
  }
}
```

Java SDK

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk Java JDBC, yang menangani pembuatan token secara otomatis.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`generateDbConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`generateDbConnectAuthToken`.

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.dsql.DsqlUtilities;
import software.amazon.awssdk.regions.Region;

public class GenerateAuthToken {
    public static String generateToken(String yourClusterEndpoint, Region region) {
        DsqlUtilities utilities = DsqlUtilities.builder()
                .region(region)
                .credentialsProvider(DefaultCredentialsProvider.builder().build())
                .build();

        // Use `generateDbConnectAuthToken` if you are _not_ logging in as `admin`
 user
        String token = utilities.generateDbConnectAdminAuthToken(builder -> {
            builder.hostname(yourClusterEndpoint)
                    .region(region);
        });

        System.out.println(token);
        return token;
    }
}
```

Rust SDK

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk Karat SQLx, yang menangani pembuatan token secara otomatis.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`db_connect_admin_auth_token`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`db_connect_auth_token`.

```
use aws_config::{BehaviorVersion, Region};
use aws_sdk_dsql::auth_token::{AuthTokenGenerator, Config};

async fn generate_token(your_cluster_endpoint: String, region: String) -> String {
    let sdk_config = aws_config::load_defaults(BehaviorVersion::latest()).await;
    let signer = AuthTokenGenerator::new(
        Config::builder()
            .hostname(&your_cluster_endpoint)
            .region(Region::new(region))
            .build()
            .unwrap(),
    );

    // Use `db_connect_auth_token` if you are _not_ logging in as `admin` user
    let token = signer.db_connect_admin_auth_token(&sdk_config).await.unwrap();
    println!("{}", token);
    token.to_string()
}
```

Ruby SDK

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk Ruby pg, yang menangani pembuatan token secara otomatis.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`generate_db_connect_admin_auth_token`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`generate_db_connect_auth_token`.

```
require 'aws-sdk-dsql'

def generate_token(your_cluster_endpoint, region)
  credentials = Aws::CredentialProviderChain.new.resolve

  token_generator = Aws::DSQL::AuthTokenGenerator.new({
```

```
    :credentials => credentials
  })

  # if you're not using admin role, use generate_db_connect_auth_token instead
  token = token_generator.generate_db_connect_admin_auth_token({
    :endpoint => your_cluster_endpoint,
    :region => region
  })
end
```

PHP SDK

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk PHP PDO_PGSQL, yang menangani pembuatan token secara otomatis.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`generateDbConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`generateDbConnectAuthToken`.

```
<?php
require 'vendor/autoload.php';

use Aws\DSQL\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

function generateToken(string $yourClusterEndpoint, string $region): string {
    $provider = CredentialProvider::defaultProvider();
    $generator = new AuthTokenGenerator($provider);

    // Use generateDbConnectAuthToken if you are not connecting as admin
    $token = $generator->generateDbConnectAdminAuthToken($yourClusterEndpoint,
 $region);

    echo $token . PHP_EOL;
    return $token;
}
```

.NET

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk .NET Npgsql, yang menangani pembuatan token secara otomatis.

> **Note**
>
> SDK resmi untuk .NET tidak menyertakan panggilan API bawaan untuk menghasilkan token otentikasi untuk Aurora DSQL. Sebaliknya, Anda harus menggunakan`DSQLAuthTokenGenerator`, yang merupakan kelas utilitas. Contoh kode berikut menunjukkan cara menghasilkan token otentikasi untuk.NET.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`DbConnectAdmin`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`DbConnect`.

Contoh berikut menggunakan kelas `DSQLAuthTokenGenerator` utilitas untuk menghasilkan token otentikasi untuk pengguna dengan `admin` peran. Ganti *`insert-dsql-cluster-endpoint`* dengan titik akhir cluster Anda.

```
using Amazon;
using Amazon.DSQL.Util;

var yourClusterEndpoint = "insert-dsql-cluster-endpoint";

// Use `DSQLAuthTokenGenerator.GenerateDbConnectAuthToken` if you are _not_ logging
 in as `admin` user
var token =
 DSQLAuthTokenGenerator.GenerateDbConnectAdminAuthToken(RegionEndpoint.USEast1,
 yourClusterEndpoint);

Console.WriteLine(token);
```

Go

> **Tip**
>
> AWS merekomendasikan penggunaanKonektor Aurora DSQL untuk Go pgx, yang menangani pembuatan token secara otomatis.

AWS SDK for Go v2 menyediakan metode bawaan untuk menghasilkan token otentikasi dalam `github.com/aws/aws-sdk-go-v2/feature/dsql/auth`paket.

- Jika Anda terhubung dengan `admin` peran, gunakan`auth.GenerateDBConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`auth.GenerateDbConnectAuthToken`.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dsql/auth"
)

func main() {
 ctx := context.Background()

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion("region"))
 if err != nil {
  panic(err)
 }

 // Use auth.GenerateDbConnectAuthToken for non-admin users
 token, err := auth.GenerateDBConnectAdminAuthToken(ctx, "yourClusterEndpoint",
 "region", cfg.Credentials)
 if err != nil {
  panic(err)
 }

 fmt.Println(token)
}
```

Menggunakan peran database dan otentikasi IAM

Aurora DSQL mendukung otentikasi menggunakan peran IAM dan pengguna IAM. Anda dapat menggunakan salah satu metode untuk mengautentikasi dan mengakses database Aurora DSQL.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin tertentu tetapi tidak terkait dengan orang tertentu. Menggunakan peran IAM memberikan kredensil keamanan sementara. Anda dapat mengambil peran IAM untuk sementara dalam beberapa cara:

- Dengan beralih peran di Konsol Manajemen AWS

- Dengan memanggil operasi AWS CLI atau AWS API
- Dengan menggunakan URL kustom

Setelah mengambil peran, Anda dapat mengakses Aurora DSQL menggunakan kredenal sementara peran. Untuk informasi selengkapnya tentang metode penggunaan peran, lihat Identitas IAM di panduan pengguna IAM.

Pengguna IAM:

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus dan dikaitkan dengan satu orang atau aplikasi. Pengguna IAM memiliki kredensi jangka panjang seperti kata sandi dan kunci akses yang dapat digunakan untuk mengakses Aurora DSQL.

> **Note**
>
> Untuk menjalankan perintah SQL dengan autentikasi IAM, Anda dapat menggunakan ARN peran IAM atau ARN pengguna IAM dalam contoh di bawah ini.

Mengotorisasi peran database untuk terhubung ke klaster Anda

Buat peran IAM dan berikan otorisasi koneksi dengan tindakan kebijakan IAM:. `dsql:DbConnect`

Kebijakan IAM juga harus memberikan izin untuk mengakses sumber daya klaster. Gunakan wildcard (*) atau ikuti petunjuk di Menggunakan kunci kondisi IAM dengan Amazon Aurora DSQL.

Mengotorisasi peran database untuk menggunakan SQL dalam database Anda

Anda harus menggunakan peran IAM dengan otorisasi untuk terhubung ke cluster Anda.

1. Connect ke cluster Aurora DSQL Anda menggunakan utilitas SQL.

   Gunakan peran `admin` database dengan identitas IAM yang diotorisasi untuk tindakan IAM untuk terhubung `dsql:DbConnectAdmin` ke cluster Anda.
2. Buat peran database baru, pastikan untuk menentukan `WITH LOGIN` opsi.

   ```
   CREATE ROLE example WITH LOGIN;
   ```

3. Kaitkan peran database dengan peran IAM ARN.

```
AWS IAM GRANT example TO 'arn:aws:iam::012345678912:role/example';
```

4. Berikan izin tingkat database ke peran database

   Contoh berikut menggunakan GRANT perintah untuk memberikan otorisasi dalam database.

```
GRANT USAGE ON SCHEMA myschema TO example;
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA myschema TO example;
```

Untuk informasi selengkapnya, lihat PostgreSQL dan GRANT PostgreSQL Privileges dalam dokumentasi PostgreSQL.

Melihat IAM ke pemetaan peran database

Untuk melihat pemetaan antara peran IAM dan peran database, kueri tabel sistem. `sys.iam_pg_role_mappings`

```
SELECT * FROM sys.iam_pg_role_mappings;
```

Contoh output:

```
 iam_oid |                  arn                   | pg_role_oid | pg_role_name |
 grantor_pg_role_oid | grantor_pg_role_name
---------+----------------------------------------+-------------+--------------
+---------------------+----------------------
   26398 | arn:aws:iam::012345678912:role/example |       26396 | example      |
       15579 | admin
(1 row)
```

Tabel ini menunjukkan semua pemetaan antara peran IAM (diidentifikasi oleh ARN mereka) dan peran database PostgreSQL.

Mencabut otorisasi database dari peran IAM

Untuk mencabut otorisasi database, gunakan operasi. `AWS IAM REVOKE`

```
AWS IAM REVOKE example FROM 'arn:aws:iam::012345678912:role/example';
```

Untuk mempelajari lebih lanjut tentang mencabut otorisasi, lihat. Mencabut otorisasi menggunakan IAM dan PostgreSQL

Aurora DSQL dan PostgreSQL

Aurora DSQL adalah database relasional terdistribusi yang PostgreSQL-compatible dirancang untuk beban kerja transaksional. Aurora DSQL menggunakan komponen PostgreSQL inti seperti parser, planner, optimizer, dan sistem tipe.

Desain Aurora DSQL memastikan bahwa semua sintaks PostgreSQL yang didukung memberikan perilaku yang kompatibel dan menghasilkan hasil kueri yang identik. Misalnya, Aurora DSQL menyediakan konversi tipe, operasi aritmatika, dan presisi numerik dan skala yang identik dengan PostgreSQL. Setiap penyimpangan didokumentasikan.

Aurora DSQL juga memperkenalkan kemampuan canggih seperti kontrol konkurensi optimis dan manajemen skema terdistribusi. Dengan fitur-fitur ini, Anda dapat menggunakan perkakas PostgreSQL yang sudah dikenal sambil memanfaatkan kinerja dan skalabilitas aplikasi terdistribusi modern, cloud-native.

Sorotan kompatibilitas PostgreSQL

Aurora DSQL saat ini didasarkan pada PostgreSQL versi 16. Sorotan utama meliputi:

Protokol kawat

Aurora DSQL menggunakan protokol kabel PostgreSQL v3 standar. Ini memungkinkan integrasi dengan klien PostgreSQL standar, driver, dan alat. Misalnya, Aurora DSQL kompatibel dengan`psql`,, dan. `pgjdbc` `psycopg`

Sintaks SQL

Aurora DSQL mendukung berbagai ekspresi dan fungsi PostgreSQL standar yang biasa digunakan dalam beban kerja transaksional. Ekspresi SQL yang didukung menghasilkan hasil yang identik dengan PostgreSQL, termasuk yang berikut:

- Penanganan nulls
- Urutkan perilaku urutan
- Skala dan presisi untuk operasi numerik
- Kesetaraan untuk operasi string

Untuk informasi selengkapnya, lihat Kompatibilitas fitur SQL di Aurora DSQL.

Manajemen transaksi

Aurora DSQL mempertahankan karakteristik utama PostgreSQL, seperti transaksi ACID dan tingkat isolasi yang setara dengan PostgreSQL Repeatable Read. Untuk informasi selengkapnya, lihat Kontrol konkurensi di Aurora DSQL.

Manfaat arsitektur terdistribusi

Desain Aurora DSQL yang terdistribusi dan tidak dibagikan memberikan manfaat kinerja dan skalabilitas di luar basis data simpul tunggal tradisional. Kemampuan utama meliputi:

Kontrol Konkurensi Optimis (OCC)

Aurora DSQL menggunakan model kontrol konkurensi yang optimis. Pendekatan bebas kunci ini mencegah transaksi memblokir satu sama lain, menghilangkan kebuntuan, dan memungkinkan eksekusi paralel throughput tinggi. Fitur-fitur ini membuat Aurora DSQL sangat berharga untuk aplikasi yang membutuhkan kinerja yang konsisten pada skala besar. Untuk contoh lebih lanjut, lihatKontrol konkurensi di Aurora DSQL.

Operasi DDL asinkron

Aurora DSQL menjalankan operasi DDL secara asinkron, yang memungkinkan pembacaan dan penulisan tanpa gangguan selama perubahan skema. Arsitektur terdistribusi memungkinkan Aurora DSQL untuk melakukan tindakan berikut:

- Jalankan operasi DDL sebagai tugas latar belakang, meminimalkan gangguan.
- Mengkoordinasikan perubahan katalog sebagai transaksi terdistribusi yang sangat konsisten. Ini memastikan visibilitas atom di semua node, bahkan selama kegagalan atau operasi bersamaan.
- Beroperasi dengan cara yang terdistribusi penuh dan tanpa pemimpin di beberapa Availability Zone dengan lapisan komputasi dan penyimpanan terpisah.

Untuk lebih lanjut tentang menggunakan perintah EXPLORE di PostgreSQL, lihat. DDL dan transaksi terdistribusi di Aurora DSQL

Kompatibilitas fitur SQL di Aurora DSQL

Pada bagian berikut, pelajari tentang dukungan Aurora DSQL untuk tipe data PostgreSQL dan perintah SQL.

Topik

- Tipe data yang didukung di Aurora DSQL
- SQL yang didukung untuk Aurora DSQL
- Subset yang didukung dari perintah SQL di Aurora DSQL
- Migrasi dari PostgreSQL ke Aurora DSQL

Tipe data yang didukung di Aurora DSQL

Aurora DSQL mendukung subset dari jenis PostgreSQL umum.

Topik

- Jenis data numerik
- Tipe data karakter
- Jenis data tanggal dan waktu
- Jenis data lain-lain
- Jenis data runtime kueri

Jenis data numerik

Aurora DSQL mendukung tipe data numerik PostgreSQL berikut.

Nama	Alias	Rentang dan presisi	Ukuran penyimpanan	Dukungan indeks
`smallint`	`int2`	-32768 ke +32767	2 byte	Ya
`integer`	`int`, `int4`	-2147483648 ke +2147483647	4 byte	Ya
`bigint`	`int8`	-9223372036854775808 ke +9223372036854775807	8 byte	Ya
`real`	`float4`	6 digit desimal presisi	4 byte	Ya

Nama	Alias	Rentang dan presisi	Ukuran penyimpanan	Dukungan indeks
`double precision`	`float8`	15 digit desimal presisi	8 byte	Ya
`numeric [ (p, s) ]`	`decimal [ (p, s) ]` `dec[ (p,s)]`	Numerik yang tepat dari presisi yang dapat dipilih. Presisi maksimum adalah 38 dan skala maksimum adalah 37. [1] Defaultnya adalah`numeric (18,6)`.	8 byte+2 byte per digit presisi. Ukuran maksimum adalah 27 byte.	Ya

[1]— Jika Anda tidak secara eksplisit menentukan ukuran saat Anda menjalankan `CREATE TABLE` atau, `ALTER TABLE ADD COLUMN` Aurora DSQL memberlakukan default. Aurora DSQL menerapkan batasan saat Anda menjalankan atau pernyataan. `INSERT UPDATE`

Tipe data karakter

Aurora DSQL mendukung tipe data karakter PostgreSQL berikut.

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`character [ (n) ]`	`char [ (n) ]`	Fixed-length string karakter	4096 byte [1]	Variabel hingga 4100 byte	Ya
`character varying [ (n) ]`	`varchar [ (n) ]`	Variable-length string karakter	65535 byte [1]	Variabel hingga 65539 byte	Ya
`bpchar [ (n) ]`		Jika panjang tetap, ini adalah alias untuk`char`. Jika panjang variabel, ini adalah alias	4096 byte [1]	Variabel hingga 4100 byte	Ya

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
		untuk`varchar`, di mana spasi tambahan secara semantik tidak signifikan.			
`text`		Variable-length string karakter	1 MiB [1]	Variabel hingga 1 MiB	Ya

[1]— Jika Anda tidak secara eksplisit menentukan ukuran ketika Anda menjalankan `CREATE TABLE` atau`ALTER TABLE ADD COLUMN`, maka Aurora DSQL memberlakukan default. Aurora DSQL menerapkan batasan saat Anda menjalankan atau pernyataan. `INSERT` `UPDATE`

Jenis data tanggal dan waktu

Aurora DSQL mendukung tipe data tanggal dan waktu PostgreSQL berikut.

Nama	Alias	Deskripsi	Kisaran	Resolusi	Ukuran penyimpanan	Dukungan indeks
`date`		Tanggal kalender (tahun, bulan, hari)	4713 SM — 5874897 IKLAN	1 hari	4 byte	Ya
`time [ (`*p*`) ] [ without time zone ]`		Waktu hari, tanpa zona waktu	00:00:00 — 24:00:00	1 mikrodetik	8 byte	Ya
`time [ (`*p*`) ] with time zone`	`timet`	waktu hari, termasuk zona waktu	00:00:00 +1559 — 24:00:00 —1559	1 mikrodetik	12 byte	Tidak

Nama	Alias	Deskripsi	Kisaran	Resolusi	Ukuran penyimpanan	Dukungan indeks
`timestamp [(p)][ without time zone]`		Tanggal dan waktu, tanpa zona waktu	4713 SM — 294276 IKLAN	1 mikrodetik	8 byte	Ya
`timestamp [(p)]with time zone`	`timestz`	Tanggal dan waktu, termasuk zona waktu	4713 SM — 294276 IKLAN	1 mikrodetik	8 byte	Ya
`interval [fields][ (p)]`		Rentang waktu	-178000000 tahun — 178000000 tahun	1 mikrodetik	16 byte	Tidak

Jenis data lain-lain

Aurora DSQL mendukung berbagai jenis data PostgreSQL berikut.

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`boolean`	`bool`	Logis Boolean () true/false		1 byte	Ya
`bytea`		Data biner (“array byte”)	1 MiB 1	Variabel hingga batas 1 MiB	Tidak
`UUID`		Pengident ifikasi unik secara universal		16 byte	Ya

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`json`		Data JSON	1 MiB [2]	Variabel hingga batas 1 MiB. [2]	Tidak
`jsonb`		Data JSON biner	1 MiB [2]	Variabel hingga batas 1 MiB. [2]	Tidak

[1]— Jika Anda tidak secara eksplisit menentukan ukuran ketika Anda menjalankan `CREATE TABLE` atau`ALTER TABLE ADD COLUMN`, maka Aurora DSQL memberlakukan default. Aurora DSQL menerapkan batasan saat Anda menjalankan atau pernyataan. `INSERT` `UPDATE`

[2]— Aurora DSQL secara otomatis menerapkan kompresi ke besar `json` dan `jsonb` nilai selama dan operasi. `INSERT` `UPDATE` Batas 1 MiB berlaku untuk ukuran terkompresi, sehingga Anda dapat menyimpan `json` dan `jsonb` nilai secara signifikan lebih besar dari 1 MiB selama mereka mengompres di bawah batas.

Untuk menonaktifkan kompresi, gunakan `STORAGE` kata kunci. Untuk informasi selengkapnya, lihat `CREATE TABLE` dan `ALTER TABLE`.

Fungsi dan operator JSON

Aurora DSQL mendukung semua fungsi dan operator PostgreSQL JSON dari bagian 9.16 Fungsi dan Operator JSON dengan perilaku yang identik.

> **Note**
>
> Fungsi`json_populate_record`,, `json_populate_recordset`, `jsonb_populate_record`, dan `jsonb_populate_recordset` bekerja dengan tipe baris tabel dan tampilan, tetapi tidak dengan tipe komposit khusus karena Aurora DSQL saat ini tidak mendukung. `CREATE TYPE`

Contoh berikut menunjukkan `json_populate_record` dan `jsonb_populate_recordset` digunakan dengan tipe baris tabel:

```
CREATE TABLE tt (c1 INT, c2 INT);
SELECT * FROM json_populate_record(null::tt, '{"c1": 1, "c2": 2}');
```

```
 c1 | c2
----+----
  1 |  2
(1 row)
```

```
SELECT * FROM jsonb_populate_recordset(null::tt, '[{"c1":1,"c2":2}, {"c1":3,"c2":4}]');
```

```
 c1 | c2
----+----
  1 |  2
  3 |  4
(2 rows)
```

Jenis data runtime kueri

Query tipe data runtime adalah tipe data internal yang digunakan pada waktu eksekusi query. Jenis ini berbeda dari PostgreSQL-compatible jenis seperti `varchar` dan `integer` yang Anda tentukan dalam skema Anda. Sebagai gantinya, jenis ini adalah representasi runtime yang digunakan Aurora DSQL saat memproses kueri.

Tipe data berikut hanya didukung selama runtime kueri:

Jenis array

Aurora DSQL mendukung array dari tipe data yang didukung. Misalnya, Anda dapat memiliki array bilangan bulat. Fungsi `string_to_array` membagi string menjadi PostgreSQL-style array dengan pembatas koma (`,`) seperti yang ditunjukkan pada contoh berikut. Anda dapat menggunakan array dalam ekspresi, output fungsi, atau perhitungan sementara selama eksekusi kueri.

```
SELECT string_to_array('1,2', ',');
```

Fungsi mengembalikan respon yang mirip dengan berikut ini:

```
 string_to_array
```

```
-----------------
 {1,2}
(1 row)
```

jenis inet

Tipe data mewakili IPv4, alamat host IPv6, dan subnetnya. Jenis ini berguna saat mengurai log, memfilter pada subnet IP, atau melakukan perhitungan jaringan dalam kueri. Untuk informasi selengkapnya, lihat inet di dokumentasi PostgreSQL.

SQL yang didukung untuk Aurora DSQL

Aurora DSQL mendukung berbagai fitur inti PostgreSQL SQL. Di bagian berikut, Anda dapat mempelajari tentang dukungan ekspresi PostgreSQL umum. Daftar ini bukanlah daftar lengkap.

Pilih perintah

Aurora DSQL mendukung klausul perintah berikut. SELECT

Klausul utama	Klausul yang didukung
FROM	
GROUP BY	ALL, DISTINCT
ORDER BY	ASC, DESC, NULLS
LIMIT	
DISTINCT	
HAVING	
USING	
WITH(ekspresi tabel umum)	
INNER JOIN	ON
OUTER JOIN	LEFT, RIGHT, FULL, ON

Klausul utama	Klausul yang didukung
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`
`OVER`	`RANK ()`, `PARTITION BY`
`FOR UPDATE`	Tentukan predikat kesetaraan pada semua kolom kunci primer (misalnya,`WHERE pk = value`). Jika Anda menggunakan rentang,, `INOR`, atau predikat non-kesetaraan lainnya, kueri akan kembali. `ERROR: locking clause such as FOR UPDATE can be applied only on tables with equality predicates on the key` Tentukan pada kueri tabel tunggal. Jika Anda menggunakan gabungan atau beberapa tabel, kueri akan kembali`ERROR: locking clause such as FOR UPDATE can be applied on a single table`.

Bahasa Definisi Data (DDL)

Aurora DSQL mendukung perintah PostgreSQL DDL berikut.

Perintah	Klausul Utama	Klausul yang Didukung
`CREATE`	`TABLE`	Untuk informasi tentang sintaks `CREATE TABLE` perintah yang didukung, lihat`CREATE TABLE`.
`ALTER`	`TABLE`	Untuk informasi tentang sintaks `ALTER TABLE` perintah yang didukung, lihat`ALTER TABLE`.

Perintah	Klausul Utama	Klausul yang Didukung
DROP	TABLE	
CREATE	[UNIQUE] INDEX ASYNC	Anda dapat menggunakan perintah ini dengan parameter berikut:ON,NULLS FIRST,NULLS LAST. Untuk informasi tentang sintaks CREATE INDEX ASYNC perintah yang didukung, lihatIndeks asinkron di Aurora DSQL.
DROP	INDEX	
CREATE	VIEW	Untuk informasi selengkapnya tentang sintaks CREATE VIEW perintah yang didukung, lihatCREATE VIEW.
ALTER	VIEW	Untuk informasi tentang sintaks ALTER VIEW perintah yang didukung, lihatALTER VIEW.
DROP	VIEW	Untuk informasi tentang sintaks DROP VIEW perintah yang didukung, lihatDROP VIEW.
CREATE	SEQUENCE	Untuk informasi tentang sintaks CREATE SEQUENCE perintah yang didukung, lihatCREATE SEQUENCE.
ALTER	SEQUENCE	Untuk informasi tentang sintaks ALTER SEQUENCE perintah yang didukung, lihatALTER SEQUENCE.
DROP	SEQUENCE	Untuk informasi tentang sintaks DROP SEQUENCE perintah yang didukung, lihatDROP SEQUENCE.
CREATE	ROLE, WITH	

Perintah	Klausul Utama	Klausul yang Didukung
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

Bahasa Manipulasi Data (DML)

Aurora DSQL mendukung perintah PostgreSQL DHTML berikut.

Perintah	Klausul utama	Klausul yang didukung
INSERT	INTO	VALUES SELECT [ON CONFLICT]
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Bahasa Kontrol Data (DCL)

Aurora DSQL mendukung perintah PostgreSQL DCL berikut.

Perintah	Klausul yang didukung
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

Bahasa Kontrol Transaksi (TCL)

Aurora DSQL mendukung perintah PostgreSQL TCL berikut.

Perintah	Klausul yang didukung	Alias
`COMMIT`	`[WORK | TRANSACTION ]` `[AND NO CHAIN]`	`END`

`BEGIN`	`[WORK | TRANSACTION ]`
	`[ISOLATION LEVEL REPEATABLE READ]`
	`[READ WRITE | READ ONLY]`	

`START TRANSACTION`	`[ISOLATION LEVEL REPEATABLE READ]` `[READ WRITE | READ ONLY]`	
`ROLLBACK`	`[WORK | TRANSACTION ]` `[AND NO CHAIN]`	`ABORT`

Perintah utilitas

Aurora DSQL mendukung perintah utilitas PostgreSQL berikut:

- `EXPLAIN`
- `ANALYZE`(nama relasi saja)

Subset yang didukung dari perintah SQL di Aurora DSQL

Bagian ini memberikan informasi terperinci tentang perintah SQL yang didukung, dengan fokus pada perintah dengan set parameter dan subperintah yang luas. Misalnya, CREATE TABLE di PostgreSQL menawarkan banyak klausa dan parameter, subset yang didukung oleh Aurora DSQL. Bagian ini menjelaskan himpunan bagian yang didukung dari perintah SQL umum menggunakan elemen sintaks PostgreSQL yang sudah dikenal yang didukung Aurora DSQL.

Topik

- CREATE TABLE
- ALTER TABLE
- CREATE SEQUENCE
- ALTER SEQUENCE
- DROP SEQUENCE
- CREATE VIEW
- ALTER VIEW
- DROP VIEW

CREATE TABLE

CREATE TABLEmendefinisikan tabel baru.

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN | DEFAULT } ]
 [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression )|
  DEFAULT default_expr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY ( sequence_options ) |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
```

```
and like_option is:

{ INCLUDING | EXCLUDING } { COMMENTS | CONSTRAINTS | DEFAULTS | GENERATED | IDENTITY |
 INDEXES | STATISTICS | ALL }

index_parameters in UNIQUE, and PRIMARY KEY constraints are:
[ INCLUDE ( column_name [, ... ] ) ]
```

Kolom identitas

> **Note**
>
> Saat menggunakan kolom identitas, nilai cache harus dipertimbangkan dengan cermat. Untuk informasi selengkapnya, lihat info penting di `CREATE SEQUENCE` halaman.
> Untuk panduan tentang cara terbaik menggunakan kolom identitas berdasarkan pola beban kerja, lihatBekerja dengan urutan dan kolom identitas.

`GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY ( sequence_options )`Klausa membuat kolom sebagai kolom identitas. Ini akan memiliki urutan implisit yang melekat padanya dan di baris yang baru disisipkan kolom akan secara otomatis memiliki nilai dari urutan yang ditetapkan padanya. Kolom seperti itu secara implisit`NOT NULL`.

Klausa `ALWAYS` dan `BY DEFAULT` menentukan bagaimana nilai yang ditentukan pengguna secara eksplisit ditangani dan perintah. `INSERT` `UPDATE`

Dalam sebuah `INSERT` perintah, jika `ALWAYS` dipilih, nilai yang ditentukan pengguna hanya diterima jika `INSERT` pernyataan menentukan. `OVERRIDING SYSTEM VALUE` Jika `BY DEFAULT` dipilih, maka nilai yang ditentukan pengguna diutamakan.

Dalam sebuah `UPDATE` perintah, jika `ALWAYS` dipilih, setiap pembaruan kolom ke nilai apa pun selain `DEFAULT` akan ditolak. Jika `BY DEFAULT` dipilih, kolom dapat diperbarui secara normal. (Tidak ada `OVERRIDING` klausul untuk `UPDATE` perintah.)

*`sequence_options`*Klausa dapat digunakan untuk mengganti parameter urutan. Opsi yang tersedia termasuk yang ditampilkan untuk`CREATE SEQUENCE`, plus`SEQUENCE NAME` *`name`*. Tanpa`SEQUENCE NAME`, sistem memilih nama yang tidak digunakan untuk urutan.

Mode penyimpanan

`STORAGE`Klausa opsional menetapkan mode penyimpanan untuk kolom. Gunakan opsi ini untuk mengontrol perilaku kompresi untuk tipe data panjang variabel seperti dan. `JSON` `JSONB`

Amazon Aurora DSQL memampatkan beberapa tipe data ketika melebihi ukuran tertentu. Untuk menonaktifkan perilaku ini, gunakan `EXTERNAL` opsi `PLAIN` atau.

`PLAIN`

Aurora DSQL menyimpan data inline tanpa kompresi. Ini adalah satu-satunya pilihan untuk tipe data dengan panjang tetap seperti. `integer` Gunakan opsi ini untuk menonaktifkan kompresi pada beberapa tipe panjang variabel.

`MAIN` | **`EXTENDED`** | **`DEFAULT`**

`MAIN`dan `EXTENDED` memungkinkan kompresi opsional kolom jika tipe data yang mendasarinya mendukung kompresi. `DEFAULT`mengatur mode penyimpanan ke mode default untuk tipe data kolom.

`EXTERNAL`

Aurora DSQL saat ini tidak mendukung tabel TOAST, namun `EXTERNAL` menonaktifkan kompresi pada tipe data yang mendukung kompresi.

`ALTER TABLE`

`ALTER TABLE`mengubah definisi tabel.

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema

where action is one of:
```

```
    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [ STORAGE { PLAIN | EXTERNAL
 | EXTENDED | MAIN | DEFAULT } ]
    ADD table_constraint_using_index
    ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN |
 DEFAULT }
    ALTER [ COLUMN ] column_name { SET GENERATED { ALWAYS | BY DEFAULT } | SET
 sequence_option | RESTART [ [ WITH ] restart ] } [...]
    ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
    OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }

and table_constraint_using_index is:

    [ CONSTRAINT constraint_name ]
    UNIQUE USING INDEX index_name
```

Mode penyimpanan

`SET STORAGE`

Formulir ini mengatur mode penyimpanan untuk kolom. Untuk detail tentang mode penyimpanan yang tersedia, lihat Mode penyimpanan di `CREATE TABLE` halaman.

Tindakan kolom identitas

`SET GENERATED { ALWAYS | BY DEFAULT }` / **`SET`** ***`sequence_option`*** / **`RESTART`**

Formulir ini mengubah apakah kolom adalah kolom identitas atau mengubah atribut generasi dari kolom identitas yang ada. Lihat `CREATE TABLE` untuk detail. Seperti`SET DEFAULT`, bentuk-bentuk ini hanya mempengaruhi perilaku berikutnya `INSERT` dan `UPDATE` perintah; mereka tidak menyebabkan baris yang sudah ada dalam tabel berubah.

`sequence_option`Ini adalah opsi yang didukung oleh `ALTER SEQUENCE` seperti`INCREMENT BY`. Bentuk-bentuk ini mengubah urutan yang mendasari kolom identitas yang ada.

`DROP IDENTITY [ IF EXISTS ]`

Formulir ini menghapus properti identitas dari kolom. Jika `DROP IDENTITY IF EXISTS` ditentukan dan kolom bukan kolom identitas, tidak ada kesalahan yang dilemparkan. Dalam hal ini pemberitahuan dikeluarkan sebagai gantinya.

Tambahkan tindakan kendala

`ADD` *`table_constraint_using_index`*

Formulir ini menambahkan `UNIQUE` kendala baru ke tabel berdasarkan indeks unik yang ada. Semua kolom indeks akan dimasukkan dalam kendala.

Indeks harus dalam `VALID` keadaan; menambahkan kendala unik menggunakan indeks saat indeks sedang dibangun tidak didukung.

Jika nama kendala disediakan maka indeks akan diganti namanya agar sesuai dengan nama kendala. Jika tidak, kendala akan diberi nama sama dengan indeks.

Setelah perintah ini dijalankan, indeks "dimiliki" oleh kendala, dengan cara yang sama seperti jika indeks telah dibangun oleh perintah biasa`CREATE UNIQUE INDEX ASYNC`. Secara khusus, menjatuhkan kendala akan membuat indeks menghilang juga.

`CREATE SEQUENCE`

`CREATE SEQUENCE`— tentukan generator urutan baru.

> **Important**
>
> Dalam PostgreSQL, `CACHE` menentukan adalah opsional dan default ke 1. Dalam sistem terdistribusi seperti Amazon Aurora DSQL, operasi urutan melibatkan koordinasi, dan ukuran cache 1 dapat meningkatkan overhead koordinasi di bawah konkurensi tinggi. Sementara nilai cache yang lebih besar memungkinkan nomor urut dilayani dari rentang yang telah dialokasikan secara lokal, meningkatkan throughput, nilai cadangan yang tidak digunakan dapat hilang, membuat kesenjangan dan efek pemesanan lebih terlihat. Karena aplikasi berbeda dalam sensitivitasnya terhadap pengurutan alokasi versus throughput, Amazon Aurora DSQL `CACHE` harus ditentukan secara eksplisit dan saat ini `CACHE = 1` mendukung `CACHE >= 65536` atau, memberikan perbedaan yang jelas antara perilaku alokasi yang lebih dekat dengan pembuatan sekuensial yang ketat dan alokasi yang dioptimalkan untuk beban kerja yang sangat bersamaan.
> Kapan`CACHE >= 65536`, nilai urutan tetap dijamin unik tetapi mungkin tidak dihasilkan dalam urutan peningkatan yang ketat di seluruh sesi, dan kesenjangan dapat terjadi, terutama ketika nilai yang di-cache tidak sepenuhnya dikonsumsi. Karakteristik ini konsisten dengan semantik PostgreSQL untuk urutan yang di-cache dalam penggunaan bersamaan, di

mana kedua sistem menjamin nilai yang berbeda tetapi tidak menjamin urutan urutan yang ketat di seluruh sesi.
Dalam satu sesi klien, nilai urutan mungkin tidak selalu tampak meningkat secara ketat, terutama transaksi eksplisit di luar. Perilaku ini mirip dengan penerapan PostgreSQL yang menggunakan penyatuan koneksi. Perilaku alokasi lebih dekat ke lingkungan PostgreSQL sesi tunggal dapat dicapai dengan menggunakan `CACHE = 1` atau dengan memperoleh nilai urutan dalam transaksi eksplisit.
Dengan`CACHE = 1`, alokasi urutan mengikuti perilaku urutan non-cache PostgreSQL.
Untuk panduan tentang cara terbaik menggunakan urutan berdasarkan pola beban kerja, lihat. Bekerja dengan urutan dan kolom identitas

Sintaksis yang didukung

```
CREATE SEQUENCE [ IF NOT EXISTS ] name CACHE cache
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ [ NO ] CYCLE ]
    [ START [ WITH ] start ]
    [ OWNED BY { table_name.column_name | NONE } ]

where data_type is BIGINT
      and cache = 1 or cache >= 65536
```

Deskripsi

`CREATE SEQUENCE`membuat generator nomor urut baru. Ini melibatkan pembuatan dan inisialisasi tabel baris tunggal khusus baru dengan nama. *`name`* Generator akan dimiliki oleh pengguna yang mengeluarkan perintah.

Jika nama skema diberikan maka urutan dibuat dalam skema yang ditentukan. Jika tidak, itu dibuat dalam skema saat ini. Nama urutan harus berbeda dari nama relasi lainnya (tabel, urutan, indeks, tampilan, tampilan terwujud, atau tabel asing) dalam skema yang sama.

Setelah urutan dibuat, Anda menggunakan fungsi`nextval`,`currval`, dan `setval` untuk beroperasi pada urutan. Fungsi-fungsi ini didokumentasikan dalamFungsi manipulasi urutan.

Meskipun Anda tidak dapat memperbarui urutan secara langsung, Anda dapat menggunakan kueri seperti:

```
SELECT * FROM name;
```

untuk memeriksa beberapa parameter dan keadaan urutan saat ini. Secara khusus, `last_value` bidang urutan menunjukkan nilai terakhir yang dialokasikan oleh sesi apa pun. (Tentu saja, nilai ini mungkin sudah usang pada saat dicetak, jika sesi lain secara aktif melakukan `nextval` panggilan.) Parameter lain seperti *`increment`* dan *`maxvalue`* dapat diamati dalam `pg_sequences` tampilan.

Parameter

`IF NOT EXISTS`

Jangan melempar kesalahan jika relasi dengan nama yang sama sudah ada. Pemberitahuan dikeluarkan dalam kasus ini. Perhatikan bahwa tidak ada jaminan bahwa relasi yang ada adalah sesuatu seperti urutan yang akan dibuat — bahkan mungkin bukan urutan.

`name`

Nama (opsional schema-qualified) dari urutan yang akan dibuat.

`data_type`

Klausa opsional `AS` *`data_type`* menentukan tipe data dari urutan. Jenis yang valid adalah`bigint`. `bigint`adalah default. Tipe data menentukan nilai minimum dan maksimum default dari urutan.

`increment`

Klausa opsional `INCREMENT BY` *`increment`* menentukan nilai yang ditambahkan ke nilai urutan saat ini untuk membuat nilai baru. Nilai positif akan membuat urutan naik, yang negatif menjadi urutan menurun. Nilai default adalah 1.

***`minvalue`* / `NO MINVALUE`**

Klausa opsional `MINVALUE` *`minvalue`* menentukan nilai minimum yang dapat dihasilkan oleh urutan. Jika klausa ini tidak disediakan atau `NO MINVALUE` ditentukan, maka default akan digunakan. Default untuk urutan naik adalah 1. Default untuk urutan menurun adalah nilai minimum tipe data.

***`maxvalue`* / `NO MAXVALUE`**

Klausa opsional `MAXVALUE` *`maxvalue`* menentukan nilai maksimum untuk urutan. Jika klausa ini tidak disediakan atau `NO MAXVALUE` ditentukan, maka nilai default akan digunakan. Default untuk urutan menaik adalah nilai maksimum tipe data. Default untuk urutan menurun adalah -1.

`CYCLE` / `NO CYCLE`

`CYCLE`Opsi ini memungkinkan urutan untuk membungkus ketika *`maxvalue`* atau *`minvalue`* telah dicapai oleh urutan naik atau turun masing-masing. Jika batas tercapai, nomor berikutnya yang dihasilkan akan menjadi *`minvalue`* atau*`maxvalue`*, masing-masing.

Jika `NO CYCLE` ditentukan, panggilan apa pun `nextval` setelah urutan mencapai nilai maksimumnya akan mengembalikan kesalahan. Jika tidak `NO CYCLE` ada `CYCLE` atau ditentukan, `NO CYCLE` adalah default.

`start`

Klausa opsional `START WITH` *`start`* memungkinkan urutan dimulai di mana saja. Nilai awal default adalah *`minvalue`* untuk urutan naik dan *`maxvalue`* untuk urutan turun.

`cache`

Klausa `CACHE` *`cache`* menentukan berapa banyak nomor urut yang akan dialokasikan sebelumnya dan disimpan dalam memori untuk akses yang lebih cepat. Nilai yang dapat diterima untuk `CACHE` Aurora DSQL adalah 1 atau nomor apa pun >= 65536. Nilai minimum adalah 1 (hanya satu nilai yang dapat dihasilkan pada satu waktu, artinya tidak ada cache).

`OWNED BY` *`table_name.column_name`* / `OWNED BY NONE`

`OWNED BY`Opsi ini menyebabkan urutan dikaitkan dengan kolom tabel tertentu, sehingga jika kolom itu (atau seluruh tabelnya) dijatuhkan, urutan akan secara otomatis turun juga. Tabel yang ditentukan harus memiliki pemilik yang sama dan berada dalam skema yang sama dengan urutannya. `OWNED BY NONE`, default, menentukan bahwa tidak ada asosiasi seperti itu.

Catatan

Gunakan `DROP SEQUENCE` untuk menghapus urutan.

Urutan didasarkan pada `bigint` aritmatika, sehingga kisarannya tidak dapat melebihi kisaran bilangan bulat delapan byte (-9223372036854775808 hingga 9223372036854775807).

Karena `nextval` dan `setval` panggilan tidak pernah diputar kembali, objek urutan tidak dapat digunakan jika penugasan nomor urut “tanpa celah” diperlukan.

Setiap sesi akan mengalokasikan dan menyimpan nilai urutan berturut-turut selama satu akses ke objek urutan dan meningkatkan objek urutan yang sesuai. `last_value` Kemudian, penggunaan *`cache`* -1 berikutnya dari `nextval` dalam sesi itu hanya mengembalikan nilai yang telah dialokasikan

sebelumnya tanpa menyentuh objek urutan. Jadi, angka apa pun yang dialokasikan tetapi tidak digunakan dalam sesi akan hilang ketika sesi itu berakhir, menghasilkan “lubang” dalam urutan.

Selain itu, meskipun beberapa sesi dijamin untuk mengalokasikan nilai urutan yang berbeda, nilai mungkin dihasilkan di luar urutan ketika semua sesi dipertimbangkan. Misalnya, dengan *cache* pengaturan 10, sesi A mungkin mencadangkan nilai 1.. 10 dan kembali `nextval` = 1, maka sesi B mungkin mencadangkan nilai 11.. 20 dan kembali `nextval` = 11 sebelum sesi A dihasilkan `nextval` = 2. Jadi, dengan *cache* pengaturan satu, aman untuk mengasumsikan bahwa `nextval` nilai dihasilkan secara berurutan; dengan *cache* pengaturan yang lebih besar dari satu Anda hanya boleh berasumsi bahwa `nextval` nilainya semuanya berbeda, bukan bahwa mereka dihasilkan murni secara berurutan. Juga, `last_value` akan mencerminkan nilai terbaru yang dicadangkan oleh sesi apa pun, apakah itu belum dikembalikan oleh`nextval`.

Pertimbangan lain adalah bahwa `setval` eksekusi pada urutan seperti itu tidak akan diperhatikan oleh sesi lain sampai mereka menggunakan nilai yang telah dialokasikan sebelumnya yang telah mereka cache.

Contoh

Buat urutan naik yang disebut`serial`, mulai dari 101:

```
CREATE SEQUENCE serial CACHE 65536 START 101;
```

Pilih nomor berikutnya dari urutan ini:

```
SELECT nextval('serial');

 nextval
---------
     101
```

Pilih nomor berikutnya dari urutan ini:

```
SELECT nextval('serial');

 nextval
---------
     102
```

Gunakan urutan ini dalam sebuah `INSERT` perintah:

```
INSERT INTO distributors VALUES (nextval('serial'), 'nothing');
```

Setel ulang urutan ke nilai tertentu menggunakan`setval`:

```
SELECT setval('serial', 200);
SELECT nextval('serial');

 nextval
---------
     201
```

Kompatibilitas

`CREATE SEQUENCE`sesuai dengan standar SQL, dengan pengecualian berikut:

- Memperoleh nilai berikutnya dilakukan dengan menggunakan `nextval()` fungsi alih-alih `NEXT VALUE FOR` ekspresi standar.
- `OWNED BY`Klausul ini adalah ekstensi PostgreSQL.

`ALTER SEQUENCE`

`ALTER SEQUENCE`— mengubah definisi generator urutan.

> **Important**
>
> Saat menggunakan urutan, nilai cache harus dipertimbangkan dengan cermat. Untuk informasi selengkapnya, lihat info penting di `CREATE SEQUENCE` halaman.
> Untuk panduan tentang cara terbaik menggunakan urutan berdasarkan pola beban kerja, lihat. Bekerja dengan urutan dan kolom identitas

Sintaksis yang didukung

```
ALTER SEQUENCE [ IF EXISTS ] name
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ [ NO ] CYCLE ]
    [ START [ WITH ] start ]
    [ RESTART [ [ WITH ] restart ] ]
```

```
    [ CACHE cache ]
    [ OWNED BY { table_name.column_name | NONE } ]
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER |
 SESSION_USER }
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema

where cache is 1 or cache >= 65536
```

Deskripsi

`ALTER SEQUENCE`mengubah parameter generator urutan yang ada. Parameter apa pun yang tidak secara khusus ditetapkan dalam `ALTER SEQUENCE` perintah mempertahankan pengaturan sebelumnya.

Anda harus memiliki urutan untuk digunakan`ALTER SEQUENCE`. Untuk mengubah skema urutan, Anda juga harus memiliki `CREATE` hak istimewa pada skema baru. Untuk mengubah pemilik, Anda harus `SET ROLE` dapat memiliki peran baru, dan peran itu harus memiliki `CREATE` hak istimewa pada skema urutan. (Pembatasan ini memberlakukan bahwa mengubah pemilik tidak melakukan apa pun yang tidak dapat Anda lakukan dengan menjatuhkan dan membuat ulang urutan. Namun, superuser dapat mengubah kepemilikan urutan apa pun.)

Parameter

name

Nama (opsional schema-qualified) dari urutan yang akan diubah.

`IF EXISTS`

Jangan melempar kesalahan jika urutannya tidak ada. Pemberitahuan dikeluarkan dalam kasus ini.

increment

Klausul `INCREMENT BY` *`increment`* ini opsional. Nilai positif akan membuat urutan naik, yang negatif menjadi urutan menurun. Jika tidak ditentukan, nilai kenaikan lama akan dipertahankan.

minvalue / **`NO MINVALUE`**

Klausa opsional `MINVALUE` *`minvalue`* menentukan nilai minimum yang dapat dihasilkan oleh urutan. Jika `NO MINVALUE` ditentukan, default 1 dan nilai minimum tipe data untuk urutan naik dan turun, masing-masing, akan digunakan. Jika tidak ada opsi yang ditentukan, nilai minimum saat ini akan dipertahankan.

***maxvalue* / NO MAXVALUE**

Klausa opsional MAXVALUE *maxvalue* menentukan nilai maksimum untuk urutan. Jika NO MAXVALUE ditentukan, default dari nilai maksimum tipe data dan -1 untuk urutan naik dan turun, masing-masing, akan digunakan. Jika tidak ada opsi yang ditentukan, nilai maksimum saat ini akan dipertahankan.

CYCLE

Kata CYCLE kunci opsional dapat digunakan untuk mengaktifkan urutan untuk membungkus ketika *maxvalue* atau *minvalue* telah dicapai oleh urutan naik atau turun masing-masing. Jika batas tercapai, nomor berikutnya yang dihasilkan akan menjadi *minvalue* atau*maxvalue*, masing-masing.

NO CYCLE

Jika kata NO CYCLE kunci opsional ditentukan, panggilan apa pun nextval setelah urutan mencapai nilai maksimumnya akan mengembalikan kesalahan. Jika tidak NO CYCLE ada CYCLE atau ditentukan, perilaku siklus lama akan dipertahankan.

start

Klausa opsional START WITH *start* mengubah nilai awal urutan yang direkam. Ini tidak berpengaruh pada nilai urutan saat ini; itu hanya menetapkan nilai yang akan digunakan ALTER SEQUENCE RESTART perintah future.

restart

Klausa opsional RESTART [WITH *restart*] mengubah nilai urutan saat ini. Ini mirip dengan memanggil setval fungsi dengan is_called =false: nilai yang ditentukan akan dikembalikan oleh panggilan berikutnya darinextval. Menulis RESTART tanpa *restart* nilai setara dengan memasok nilai awal yang direkam oleh CREATE SEQUENCE atau terakhir ditetapkan olehALTER SEQUENCE START WITH.

Berbeda dengan setval panggilan, RESTART operasi pada urutan bersifat transaksional dan memblokir transaksi bersamaan dari mendapatkan nomor dari urutan yang sama. Jika itu bukan mode operasi yang diinginkan, setval harus digunakan.

cache

Klausul CACHE *cache* ini memungkinkan nomor urut untuk dialokasikan sebelumnya dan disimpan dalam memori untuk akses yang lebih cepat. Nilai harus berupa 1 atau beberapa nilai >=

65536. Jika tidak ditentukan, nilai cache lama akan dipertahankan. Untuk informasi selengkapnya tentang perilaku cache, lihat panduan di bawahCREATE SEQUENCE.

OWNED BY *table_name*.*column_name* / OWNED BY NONE

OWNED BYOpsi ini menyebabkan urutan dikaitkan dengan kolom tabel tertentu, sehingga jika kolom itu (atau seluruh tabelnya) dijatuhkan, urutan akan secara otomatis turun juga. Jika ditentukan, asosiasi ini menggantikan asosiasi yang ditentukan sebelumnya untuk urutan tersebut. Tabel yang ditentukan harus memiliki pemilik yang sama dan berada dalam skema yang sama dengan urutannya. Menentukan OWNED BY NONE menghapus asosiasi yang ada, membuat urutan "berdiri bebas".

new_owner

Nama pengguna dari pemilik baru urutan.

new_name

Nama baru untuk urutan.

new_schema

Skema baru untuk urutan.

Catatan

ALTER SEQUENCEtidak akan segera mempengaruhi nextval hasil di backend, selain yang sekarang, yang memiliki nilai urutan yang telah dialokasikan sebelumnya (di-cache). Mereka akan menggunakan semua nilai yang di-cache sebelum memperhatikan parameter pembuatan urutan yang diubah. Backend saat ini akan segera terpengaruh.

ALTER SEQUENCEtidak mempengaruhi currval status untuk urutan.

ALTER SEQUENCEdapat menyebabkan transaksi lain ke OCC.

Untuk alasan historis, ALTER TABLE dapat digunakan dengan urutan juga; tetapi satu-satunya varian ALTER TABLE yang diizinkan dengan urutan setara dengan bentuk yang ditunjukkan di atas.

Contoh

Mulai ulang urutan yang disebutserial, pada 105:

```
ALTER SEQUENCE serial RESTART WITH 105;
```

Kompatibilitas

`ALTER SEQUENCE`sesuai dengan standar SQL, kecuali untuk,,,,`AS`, dan `SET SCHEMA` klausa `START WITH` `OWNED BY` `OWNER TO``RENAME TO`, yang merupakan ekstensi PostgreSQL.

DROP SEQUENCE

`DROP SEQUENCE`— menghapus urutan.

Sintaksis yang didukung

```
DROP SEQUENCE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Deskripsi

`DROP SEQUENCE`menghapus generator nomor urut. Urutan hanya dapat dijatuhkan oleh pemiliknya atau superuser.

Parameter

`IF EXISTS`

Jangan melempar kesalahan jika urutannya tidak ada. Pemberitahuan dikeluarkan dalam kasus ini.

`name`

Nama (opsional skema-kualifikasi) dari urutan.

`CASCADE`

Secara otomatis menjatuhkan objek yang bergantung pada urutan, dan pada gilirannya semua objek yang bergantung pada objek tersebut.

`RESTRICT`

Menolak untuk menjatuhkan urutan jika ada objek yang bergantung padanya. Ini adalah opsi default.

Contoh

Untuk menghapus urutan`seq`:

```
DROP SEQUENCE seq;
```

Kompatibilitas

`DROP SEQUENCE`sesuai dengan standar SQL, kecuali bahwa standar hanya memungkinkan satu urutan untuk dijatuhkan per perintah, dan terlepas dari `IF EXISTS` opsi, yang merupakan ekstensi PostgreSQL.

`CREATE VIEW`

`CREATE VIEW`mendefinisikan pandangan persisten baru. Aurora DSQL tidak mendukung tampilan sementara; hanya tampilan permanen yang didukung.

Sintaksis yang didukung

```
CREATE [ OR REPLACE ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Deskripsi

`CREATE VIEW`mendefinisikan tampilan kueri. Pandangan itu tidak terwujud secara fisik. Sebagai gantinya, kueri dijalankan setiap kali tampilan direferensikan dalam kueri.

`CREATE or REPLACE VIEW`mirip, tetapi jika tampilan dengan nama yang sama sudah ada, itu diganti. Kueri baru harus menghasilkan kolom yang sama yang dihasilkan oleh kueri tampilan yang ada (yaitu, nama kolom yang sama dalam urutan yang sama dan dengan tipe data yang sama), tetapi mungkin menambahkan kolom tambahan ke akhir daftar. Perhitungan yang memunculkan kolom output mungkin berbeda.

Jika nama skema diberikan, sepertiCREATE VIEW `myschema.myview ...`) maka tampilan dibuat dalam skema yang ditentukan. Jika tidak, itu dibuat dalam skema saat ini.

Nama tampilan harus berbeda dari nama relasi lainnya (tabel, indeks, tampilan) dalam skema yang sama.

Parameter

`CREATE VIEW`mendukung berbagai parameter untuk mengontrol perilaku tampilan yang dapat diperbarui secara otomatis.

RECURSIVE

Menciptakan tampilan rekursif. Sintaks: `CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...;` setara dengan. `CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;`

Daftar nama kolom tampilan harus ditentukan untuk tampilan rekursif.

name

Nama tampilan yang akan dibuat, yang mungkin secara opsional memenuhi syarat skema. Daftar nama kolom harus ditentukan untuk tampilan rekursif.

column_name

Daftar opsional nama yang akan digunakan untuk kolom tampilan. Jika tidak diberikan, nama kolom disimpulkan dari kueri.

WITH (view_option_name [= view_option_value] [, ...])

Klausa ini menentukan parameter opsional untuk tampilan; parameter berikut didukung.

- `check_option (enum)`— Parameter ini mungkin salah satu `local` atau`cascaded`, dan setara dengan menentukan`WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)`—Ini harus digunakan jika tampilan dimaksudkan untuk memberikan keamanan tingkat baris. Aurora DSQL saat ini tidak mendukung keamanan tingkat baris, tetapi opsi ini masih akan memaksa kondisi tampilan (dan `WHERE` kondisi apa pun yang menggunakan operator yang ditandai sebagai`LEAKPROOF`) untuk dievaluasi terlebih dahulu.
- `security_invoker (boolean)`—Opsi ini menyebabkan hubungan dasar yang mendasarinya diperiksa terhadap hak istimewa pengguna tampilan daripada pemilik tampilan. Lihat catatan di bawah ini untuk detail selengkapnya.

Semua opsi di atas dapat diubah pada tampilan yang ada menggunakan`ALTER VIEW`.

query

A `SELECT` atau `VALUES` perintah yang akan menyediakan kolom dan baris tampilan.

WITH [CASCADED | LOCAL] CHECK OPTION

Opsi ini mengontrol perilaku tampilan yang dapat diperbarui secara otomatis. Ketika opsi ini ditentukan, `INSERT` dan `UPDATE` perintah pada tampilan akan diperiksa untuk memastikan bahwa baris baru memenuhi kondisi penentuan tampilan (yaitu, baris baru diperiksa untuk memastikan

bahwa mereka terlihat melalui tampilan). Jika tidak, pembaruan akan ditolak. Jika tidak `CHECK OPTION` ditentukan, `INSERT` dan `UPDATE` perintah pada tampilan diizinkan untuk membuat baris yang tidak terlihat melalui tampilan.

`LOCAL`—Baris baru hanya diperiksa terhadap kondisi yang ditentukan secara langsung dalam tampilan itu sendiri. Setiap kondisi yang ditentukan pada tampilan dasar yang mendasarinya tidak dicentang (kecuali mereka juga menentukan`CHECK OPTION`).

`CASCADED`—Baris baru diperiksa terhadap kondisi tampilan dan semua tampilan dasar yang mendasarinya. Jika `CHECK OPTION` ditentukan, dan tidak `LOCAL` juga `CASCADED` ditentukan, maka `CASCADED` diasumsikan.

> **Note**
>
> Ini tidak `CHECK OPTION` dapat digunakan dengan `RECURSIVE` tampilan. `CHECK OPTION`Ini hanya didukung pada tampilan yang dapat diperbarui secara otomatis.

Catatan

Gunakan `DROP VIEW` pernyataan untuk menghapus tampilan.

Nama dan tipe data kolom tampilan harus dipertimbangkan dengan cermat. Misalnya, CREATE VIEW vista AS SELECT 'Hello World'; tidak disarankan karena nama kolom default ke. `?column?;` Juga, tipe data kolom default k`etext`, yang mungkin bukan yang Anda inginkan.

Pendekatan yang lebih baik adalah secara eksplisit menentukan nama kolom dan tipe data, seperti:. `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`

Secara default, akses ke relasi dasar yang dirujuk dalam tampilan ditentukan oleh izin pemilik tampilan. Dalam beberapa kasus, ini dapat digunakan untuk menyediakan akses yang aman tetapi terbatas ke tabel yang mendasarinya. Namun, tidak semua pandangan aman terhadap gangguan.

- Jika tampilan memiliki `security_invoker` properti yang disetel ke true, akses ke relasi dasar yang mendasarinya ditentukan oleh izin pengguna yang mengeksekusi kueri, bukan pemilik tampilan. Dengan demikian, pengguna tampilan pemanggil keamanan harus memiliki izin yang relevan pada tampilan dan hubungan dasar yang mendasarinya.
- Jika salah satu hubungan dasar yang mendasarinya adalah tampilan pemanggil keamanan, itu akan diperlakukan seolah-olah telah diakses langsung dari kueri asli. Dengan demikian, tampilan

pemanggil keamanan akan selalu memeriksa hubungan dasar yang mendasarinya menggunakan izin pengguna saat ini, bahkan jika itu diakses dari tampilan tanpa properti. `security_invoker`

- Fungsi yang dipanggil dalam tampilan diperlakukan sama seperti jika mereka telah dipanggil langsung dari kueri menggunakan tampilan. Oleh karena itu, pengguna tampilan harus memiliki izin untuk memanggil semua fungsi yang digunakan oleh tampilan. Fungsi dalam tampilan dijalankan dengan hak istimewa pengguna yang mengeksekusi kueri atau pemilik fungsi, tergantung pada apakah fungsi didefinisikan sebagai `SECURITY INVOKER` atau. `SECURITY DEFINER`
- Pengguna yang membuat atau mengganti tampilan harus memiliki `USAGE` hak istimewa pada skema apa pun yang dirujuk dalam kueri tampilan, untuk mencari objek yang direferensikan dalam skema tersebut.
- Ketika `CREATE OR REPLACE VIEW` digunakan pada tampilan yang ada, hanya `SELECT` aturan penentu tampilan, ditambah `WITH ( ... )` parameter apa pun dan `CHECK OPTION` yang diubah. Properti tampilan lainnya, termasuk kepemilikan, izin, dan aturan non-Pilih, tetap tidak berubah. Anda harus memiliki tampilan untuk menggantinya (ini termasuk menjadi anggota peran pemilik).

Tampilan yang dapat diperbarui

Tampilan sederhana dapat diperbarui secara otomatis: sistem akan memungkinkan`INSERT`,`UPDATE`, dan `DELETE` pernyataan yang akan digunakan pada tampilan dengan cara yang sama seperti pada tabel biasa. Tampilan dapat diperbarui secara otomatis jika memenuhi semua kondisi berikut:

- Tampilan harus memiliki tepat satu entri dalam `FROM` daftarnya, yang harus berupa tabel atau tampilan lain yang dapat diperbarui.
- Definisi tampilan tidak boleh berisi`WITH`,,`DISTINCT`,`GROUP BY`, `HAVING`,`LIMIT`, atau `OFFSET` klausa di tingkat atas.
- Definisi tampilan tidak boleh berisi operasi set (`UNION`,`INTERSECT`, atau`EXCEPT`) di tingkat atas.
- Daftar pilih tampilan tidak boleh berisi agregat, fungsi jendela, atau fungsi set-return.

Tampilan yang dapat diperbarui secara otomatis mungkin berisi campuran kolom yang dapat diperbarui dan yang tidak dapat diperbarui. Kolom dapat diperbarui jika itu adalah referensi sederhana ke kolom yang dapat diperbarui dari hubungan dasar yang mendasarinya. Jika tidak, kolom tersebut hanya-baca, dan kesalahan terjadi jika `UPDATE` pernyataan `INSERT` atau mencoba untuk menetapkan nilai untuk itu.

Tampilan yang lebih kompleks yang tidak memenuhi semua kondisi ini adalah hanya-baca secara default: sistem tidak mengizinkan penyisipan, pembaruan, atau penghapusan pada tampilan.

> **Note**
>
> Pengguna yang melakukan penyisipan, pembaruan, atau penghapusan pada tampilan harus memiliki hak sisipan, pembaruan, atau hapus yang sesuai pada tampilan. Secara default, pemilik tampilan harus memiliki hak istimewa yang relevan pada relasi dasar yang mendasarinya, sementara pengguna yang melakukan pembaruan tidak memerlukan izin apa pun pada relasi dasar yang mendasarinya. Namun, jika tampilan memiliki security_invoker disetel ke true, pengguna yang melakukan pembaruan, bukan pemilik tampilan, harus memiliki hak istimewa yang relevan pada relasi dasar yang mendasarinya.

Contoh

Untuk membuat tampilan yang terdiri dari semua film komedi.

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Buat tampilan dengan`LOCAL CHECK OPTION`.

```
CREATE VIEW pg_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'PG'
    WITH CASCADED CHECK OPTION;
```

Buat tampilan rekursif.

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
    VALUES (1)
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

Kompatibilitas

`CREATE OR REPLACE VIEW`adalah ekstensi bahasa PostgreSQL. `WITH ( ... )`Klausul ini juga merupakan perpanjangan, seperti juga pandangan penghalang keamanan dan pandangan pemanggil keamanan. Aurora DSQL mendukung ekstensi bahasa ini.

ALTER VIEW

`ALTER VIEW`Pernyataan ini memungkinkan mengubah berbagai properti dari tampilan yang ada, dan Aurora DSQL mendukung semua sintaks PostgreSQL untuk perintah ini.

Sintaksis yang didukung

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER VIEW [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER |
 SESSION_USER }
ALTER VIEW [ IF EXISTS ] name RENAME [ COLUMN ] column_name TO new_column_name
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ... ] )
ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

Deskripsi

`ALTER VIEW`mengubah berbagai properti tambahan tampilan. (Jika Anda ingin mengubah kueri penentu tampilan, gunakan`CREATE OR REPLACE VIEW`.) Anda harus memiliki pandangan untuk digunakan`ALTER VIEW`. Untuk mengubah skema tampilan, Anda juga harus memiliki `CREATE` hak istimewa pada skema baru. Untuk mengubah pemilik, Anda harus `SET ROLE` dapat memiliki peran baru, dan peran itu harus memiliki `CREATE` hak istimewa pada skema tampilan.

Parameter

`name`

Nama (opsional schema-qualified) dari tampilan yang ada.

`column_name`

Nama kolom yang ada, atau nama baru untuk kolom yang ada.

`IF EXISTS`

Jangan melempar kesalahan jika tampilan tidak ada. Pemberitahuan dikeluarkan dalam kasus ini.

`SET/DROP DEFAULT`

Formulir ini mengatur atau menghapus nilai default untuk kolom. Nilai default untuk kolom tampilan diganti ke salah satu `INSERT` atau `UPDATE` perintah di mana target adalah tampilan.

`new_owner`

Nama pengguna dari pemilik tampilan yang baru.

`new_name`

Nama baru untuk tampilan.

`new_schema`

Skema baru untuk tampilan.

`SET ( view_option_name [= view_option_value] [, ... ] )`

Menetapkan opsi tampilan. Berikut ini adalah opsi yang didukung:

- `check_option (enum)`- Mengubah opsi centang tampilan. Nilainya harus `local` atau`cascaded`.
- `security_barrier (boolean)`- Mengubah properti penghalang keamanan tampilan.
- `security_invoker (boolean)`- Mengubah properti security-invoker tampilan.

`RESET ( view_option_name [, ... ] )`

Mengatur ulang opsi tampilan ke nilai defaultnya.

Contoh

Mengganti nama tampilan `foo` menjadi`bar`:

```
ALTER VIEW foo RENAME TO bar;
```

Melampirkan nilai kolom default ke tampilan yang dapat diperbarui:

```
CREATE TABLE base_table (id int, ts timestamptz);
CREATE VIEW a_view AS SELECT * FROM base_table;
ALTER VIEW a_view ALTER COLUMN ts SET DEFAULT now();
INSERT INTO base_table(id) VALUES(1);  -- ts will receive a NULL
INSERT INTO a_view(id) VALUES(2);  -- ts will receive the current time
```

Kompatibilitas

`ALTER VIEW`adalah ekstensi PostgreSQL dari standar SQL yang didukung Aurora DSQL.

DROP VIEW

`DROP VIEW`Pernyataan tersebut menghapus tampilan yang ada. Aurora DSQL mendukung sintaks PostgreSQL lengkap untuk perintah ini.

Sintaksis yang didukung

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Deskripsi

`DROP VIEW`menjatuhkan tampilan yang ada. Untuk menjalankan perintah ini, Anda harus menjadi pemilik tampilan.

Parameter

`IF EXISTS`

Jangan melempar kesalahan jika tampilan tidak ada. Pemberitahuan dikeluarkan dalam kasus ini.

`name`

Nama (opsional schema-qualified) dari tampilan yang akan dihapus.

`CASCADE`

Secara otomatis menjatuhkan objek yang bergantung pada tampilan (seperti tampilan lain), dan pada gilirannya semua objek yang bergantung pada objek tersebut.

`RESTRICT`

Menolak untuk menjatuhkan tampilan jika ada objek yang bergantung padanya. Ini adalah opsi default.

Contoh

```
DROP VIEW kinds;
```

Kompatibilitas

Perintah ini sesuai dengan standar SQL, kecuali bahwa standar hanya memungkinkan satu tampilan untuk dijatuhkan per perintah, dan terlepas dari `IF EXISTS` opsi, yang merupakan ekstensi PostgreSQL yang didukung Aurora DSQL.

Migrasi dari PostgreSQL ke Aurora DSQL

Aurora DSQL dirancang agar kompatibel dengan PostgreSQL, mendukung fitur relasional inti seperti transaksi ACID, indeks sekunder, gabungan, dan operasi DHTML standar. Sebagian besar aplikasi PostgreSQL yang ada dapat bermigrasi ke Aurora DSQL dengan sedikit perubahan.

Bagian ini memberikan panduan praktis untuk memigrasikan aplikasi Anda ke Aurora DSQL, termasuk kompatibilitas kerangka kerja, pola migrasi, dan pertimbangan arsitektur.

Kerangka kerja dan kompatibilitas ORM

Aurora DSQL menggunakan protokol kawat PostgreSQL standar, memastikan kompatibilitas dengan driver dan kerangka kerja PostgreSQL. ORM paling populer bekerja dengan Aurora DSQL dengan sedikit atau tanpa perubahan. Lihat the section called “Adaptor Aurora DSQL” untuk implementasi referensi dan integrasi ORM yang tersedia.

Pola migrasi umum

Saat bermigrasi dari PostgreSQL ke Aurora DSQL, beberapa fitur bekerja secara berbeda atau memiliki sintaks alternatif. Bagian ini memberikan panduan tentang skenario migrasi umum.

Alternatif operasi DDL

Aurora DSQL menyediakan alternatif modern untuk operasi PostgreSQL DDL tradisional:

Pembuatan indeks

- Gunakan `CREATE INDEX ASYNC` alih-alih `CREATE INDEX` untuk pembuatan indeks non-pemblokiran.
- Manfaat: pembuatan Zero-downtime indeks pada tabel besar.

Penghapusan data

- Gunakan `DELETE FROM table_name` sebagai penggantiTRUNCATE.
- Alternatif: Untuk rekreasi meja lengkap, gunakan `DROP TABLE` diikuti olehCREATE TABLE.

Konfigurasi sistem

- Aurora DSQL sepenuhnya dikelola, sehingga konfigurasi ditangani secara otomatis berdasarkan pola beban kerja. Gunakan AWS Management Console atau API untuk mengelola setelan klaster.
- Manfaat: Tidak perlu penyetelan basis data atau manajemen parameter.

Pola desain skema

Sesuaikan pola PostgreSQL umum ini untuk kompatibilitas Aurora DSQL:

Pola integritas referensial

Aurora DSQL mendukung hubungan tabel dan operasi. `JOIN` Untuk integritas referensial, terapkan validasi di lapisan aplikasi Anda. Desain ini selaras dengan pola database terdistribusi modern di mana validasi lapisan aplikasi memberikan lebih banyak fleksibilitas dan menghindari kemacetan kinerja dari operasi cascading.

Pola: Menerapkan pemeriksaan integritas referensial di lapisan aplikasi Anda menggunakan konvensi penamaan yang konsisten, logika validasi, dan batas transaksi. Banyak aplikasi skala tinggi lebih memilih pendekatan ini untuk kontrol yang lebih baik atas penanganan kesalahan dan kinerja.

Penanganan data sementara

Gunakan CTE, subkueri, atau tabel biasa dengan logika pembersihan, bukan tabel sementara.

Alternatif: Buat tabel dengan nama khusus sesi dan bersihkan di aplikasi Anda.

Memahami perbedaan arsitektur

Arsitektur Aurora DSQL yang didistribusikan dan tanpa server sengaja berbeda dari PostgreSQL tradisional di beberapa area. Perbedaan ini memungkinkan manfaat utama Aurora DSQL dari kesederhanaan dan skala.

Model basis data yang disederhanakan

Database tunggal per cluster

Aurora DSQL menyediakan satu database built-in bernama per cluster. `postgres`

Tip migrasi: Jika aplikasi Anda menggunakan beberapa database, buat cluster Aurora DSQL terpisah untuk pemisahan logis, atau gunakan skema dalam satu cluster.

Tidak ada tabel sementara

Untuk penanganan data sementara, Anda HARUS menggunakan ekspresi tabel umum (CTE) dan subkueri, yang menyediakan alternatif fleksibel untuk kueri kompleks.

Alternatif: Gunakan CTE dengan `WITH` klausa untuk set hasil sementara, atau tabel reguler dengan penamaan unik untuk data khusus sesi.

Manajemen penyimpanan otomatis

Aurora DSQL menghilangkan ruang meja dan manajemen penyimpanan manual. Penyimpanan secara otomatis menskalakan dan mengoptimalkan berdasarkan pola data Anda.

Manfaat: Tidak perlu memantau ruang disk, merencanakan alokasi penyimpanan, atau mengelola konfigurasi tablespace.

Pola aplikasi modern

Aurora DSQL mendorong pola pengembangan aplikasi modern yang meningkatkan pemeliharaan dan kinerja:

Application-level logika bukan pemicu database

Untuk fungsionalitas seperti pemicu, terapkan logika berbasis peristiwa di lapisan aplikasi Anda.

Strategi migrasi: Pindahkan logika pemicu ke kode aplikasi, gunakan arsitektur berbasis peristiwa dengan AWS layanan seperti EventBridge, atau terapkan jejak audit menggunakan pencatatan aplikasi.

Fungsi SQL untuk pemrosesan data

Aurora DSQL mendukung SQL-based fungsi tetapi bukan bahasa prosedural seperti. PL/pgSQL

Alternatif: Gunakan fungsi SQL untuk transformasi data, atau pindahkan logika kompleks ke lapisan aplikasi atau fungsi Lambda AWS Anda.

Kontrol konkurensi yang optimis alih-alih penguncian pesimis

Aurora DSQL menggunakan optimistic concurrency control (OCC), pendekatan bebas kunci yang berbeda dari mekanisme penguncian basis data tradisional. Alih-alih memperoleh kunci yang memblokir transaksi lain, Aurora DSQL memungkinkan transaksi untuk dilanjutkan tanpa memblokir dan mendeteksi konflik pada waktu komit. Ini menghilangkan kebuntuan dan mencegah transaksi lambat memblokir operasi lain.

Perbedaan utama: Ketika konflik terjadi, Aurora DSQL mengembalikan kesalahan serialisasi daripada membuat transaksi menunggu kunci. Ini mengharuskan aplikasi untuk

mengimplementasikan logika coba lagi, mirip dengan menangani batas waktu kunci dalam database tradisional, tetapi konflik diselesaikan segera daripada menyebabkan pemblokiran menunggu.

Pola desain: Menerapkan logika transaksi idempoten dengan mekanisme coba lagi. Rancang skema untuk meminimalkan pertengkaran dengan menggunakan kunci primer acak dan menyebarkan pembaruan di seluruh rentang kunci Anda. Lihat perinciannya di Kontrol konkurensi di Aurora DSQL.

Penyederhanaan operasional

Aurora DSQL menghilangkan banyak tugas pemeliharaan basis data tradisional, mengurangi overhead operasional:

Tidak diperlukan perawatan manual

Aurora DSQL secara otomatis mengelola pengoptimalan penyimpanan, pengumpulan statistik, dan penyetelan kinerja. Perintah pemeliharaan tradisional seperti `VACUUM` ditangani oleh sistem.

Manfaat: Menghilangkan kebutuhan akan jendela pemeliharaan basis data, penjadwalan vakum, dan penyetelan parameter sistem.

Partisi dan penskalaan otomatis

Aurora DSQL secara otomatis mempartisi dan mendistribusikan data Anda berdasarkan pola akses. Gunakan UUID atau ID yang dihasilkan aplikasi untuk distribusi yang optimal.

Tip migrasi: Hapus logika partisi manual dan biarkan Aurora DSQL menangani distribusi data. Gunakan UUID atau ID yang dihasilkan aplikasi untuk distribusi yang optimal. Jika aplikasi Anda memerlukan pengidentifikasi berurutan, lihat. Urutan dan kolom identitas

Migrasi agen dengan alat AI

Agen pengkodean AI dapat mempercepat migrasi Anda ke Aurora DSQL dengan menganalisis skema, mengubah kode, dan menjalankan migrasi DDL dengan pemeriksaan keamanan bawaan.

Menggunakan Kiro untuk migrasi

Agen pengkodean seperti Kiro dapat membantu Anda menganalisis dan memigrasikan kode PostgreSQL Anda ke Aurora DSQL:

- Analisis skema: Unggah file skema yang ada dan minta Kiro mengidentifikasi potensi masalah kompatibilitas dan menyarankan alternatif
- Transformasi kode: Berikan kode aplikasi Anda dan minta Kiro untuk membantu memfaktorkan ulang logika pemicu, mengganti urutan dengan UUID, atau memodifikasi pola transaksi
- Perencanaan migrasi: Minta Kiro untuk membuat rencana migrasi langkah demi langkah berdasarkan arsitektur aplikasi spesifik Anda
- Migrasi DDL: Jalankan modifikasi skema menggunakan pola rekreasi meja dengan pemeriksaan keamanan bawaan dan verifikasi pengguna

Contoh petunjuk:

```
"Analyze this PostgreSQL schema for DSQL compatibility and suggest alternatives for any
 unsupported features"

"Help me refactor this trigger function into application-level logic for DSQL
 migration"

"Create a migration checklist for moving my Django application from PostgreSQL to DSQL"

"Drop the legacy_status column from the orders table"

"Change the price column from VARCHAR to DECIMAL in the products table"
```

Migrasi DDL dengan rekreasi meja

Saat menggunakan agen AI dengan server MCP Aurora DSQL, operasi ALTER TABLE tertentu menggunakan pola rekreasi tabel yang memigrasikan data Anda dengan aman. Agen menangani kompleksitas sambil memberi Anda informasi di setiap langkah.

Operasi berikut menggunakan pola rekreasi meja:

Operasi	Pendekatan
`DROP COLUMN`	Kecualikan kolom dari tabel baru
`ALTER COLUMN TYPE`	Mentransmisikan tipe data selama migrasi
`ALTER COLUMN SET/DROP NOT NULL`	Ubah kendala dalam definisi tabel baru

Operasi	Pendekatan
`ALTER COLUMN SET/DROP DEFAULT`	Tentukan default dalam definisi tabel baru
`ADD/DROP CONSTRAINT`	Sertakan atau hapus kendala di tabel baru
`MODIFY PRIMARY KEY`	Tentukan PK baru dengan validasi keunikan
Split/Merge kolom	Gunakan SPLIT_PART, SUBSTRING, atau CONCAT

Operasi ALTER TABLE berikut didukung secara langsung tanpa rekreasi meja:

- `ALTER TABLE ... RENAME COLUMN`— Ganti nama kolom
- `ALTER TABLE ... RENAME TO`— Ganti nama tabel
- `ALTER TABLE ... ADD COLUMN`- Tambahkan kolom baru

Fitur keamanan: Saat menjalankan migrasi DDL, agen AI menyajikan rencana migrasi, memverifikasi kompatibilitas data, mengonfirmasi jumlah baris, dan meminta persetujuan eksplisit sebelum operasi destruktif seperti DROP TABLE.

Migrasi batch: Untuk tabel yang melebihi 3.000 baris, agen secara otomatis mengumpulkan migrasi dengan penambahan 500-1.000 baris agar tetap berada dalam batas transaksi.

Server MCP Aurora DSQL

Server Aurora DSQL Model Context Protocol (MCP) memungkinkan asisten AI untuk terhubung langsung ke cluster Aurora DSQL Anda dan mencari dokumentasi Aurora DSQL. Ini memungkinkan AI untuk:

- Analisis skema yang ada dan sarankan perubahan migrasi
- Jalankan migrasi DDL dengan pola rekreasi meja
- Uji kueri dan verifikasi kompatibilitas selama migrasi
- Berikan panduan yang akurat dan terkini berdasarkan dokumentasi Aurora DSQL terbaru

Untuk menggunakan server Aurora DSQL MCP dengan asisten AI, lihat instruksi penyiapan untuk server MCP Aurora DSQL.

Pertimbangan Aurora DSQL untuk kompatibilitas PostgreSQL

Aurora DSQL memiliki perbedaan dukungan fitur dari PostgreSQL yang dikelola sendiri yang memungkinkan arsitektur terdistribusi, operasi tanpa server, dan penskalaan otomatis. Sebagian besar aplikasi bekerja dalam perbedaan ini tanpa modifikasi.

Untuk pertimbangan umum, lihatPertimbangan untuk bekerja dengan Amazon Aurora DSQL. Untuk kuota dan batasan, lihatKuota cluster dan batas database di Amazon Aurora DSQL.

- Aurora DSQL menggunakan database built-in tunggal bernama per cluster. `postgres` Untuk pemisahan logis, buat cluster Aurora DSQL terpisah atau gunakan skema dalam satu cluster.
- `postgres`Basis data menggunakan pengkodean UTF-8 karakter, yang menyediakan dukungan karakter internasional yang luas.
- Basis data hanya `C` menggunakan pemeriksaan.
- Aurora DSQL digunakan `UTC` sebagai zona waktu sistem. Postgres menyimpan semua tanggal dan waktu yang sadar zona waktu secara internal di UTC. Anda dapat mengatur parameter `TimeZone` konfigurasi untuk mengonversi bagaimana itu ditampilkan ke klien dan berfungsi sebagai default untuk input klien yang akan digunakan server untuk mengonversi ke UTC secara internal.
- Tingkat isolasi transaksi ditetapkan di PostgreSQL`Repeatable Read`.
- Transaksi memiliki kendala sebagai berikut:
  - Operasi DDL dan DML memerlukan transaksi terpisah
  - Transaksi hanya dapat mencakup 1 pernyataan DDL
  - Transaksi dapat memodifikasi hingga 3.000 baris, terlepas dari jumlah indeks sekunder
  - Batas 3.000 baris berlaku untuk semua pernyataan DHTML (,,) `INSERT` `UPDATE` `DELETE`
- Waktu koneksi database habis setelah 1 jam.
- Aurora DSQL mengelola izin melalui hibah tingkat skema. Pengguna admin membuat skema menggunakan `CREATE SCHEMA` dan memberikan akses menggunakan`GRANT USAGE ON SCHEMA`. Pengguna admin mengelola objek dalam skema publik, sementara pengguna non-admin membuat objek dalam skema yang dibuat pengguna untuk batas kepemilikan yang jelas. Untuk informasi selengkapnya, lihat Mengotorisasi peran database untuk menggunakan SQL dalam database Anda.

Butuh bantuan dengan migrasi?

Jika Anda menemukan fitur yang penting untuk migrasi Anda tetapi saat ini tidak didukung di Aurora DSQL, lihat Memberikan umpan balik tentang Amazon Aurora DSQL untuk informasi tentang cara berbagi umpan balik dengan AWS.

Kontrol konkurensi di Aurora DSQL

Concurrency memungkinkan beberapa sesi untuk mengakses dan memodifikasi data secara bersamaan tanpa mengorbankan integritas dan konsistensi data. Aurora DSQL menyediakan kompatibilitas PostgreSQL sambil menerapkan mekanisme kontrol konkurensi modern yang bebas kunci. Ini mempertahankan kepatuhan ACID penuh melalui isolasi snapshot, memastikan konsistensi dan keandalan data.

Keuntungan utama Aurora DSQL adalah arsitekturnya yang bebas kunci, yang menghilangkan kemacetan kinerja basis data yang umum. Aurora DSQL mencegah transaksi lambat memblokir operasi lain dan menghilangkan risiko kebuntuan. Pendekatan ini membuat Aurora DSQL sangat berharga untuk aplikasi throughput tinggi di mana kinerja dan skalabilitas sangat penting.

Respons kontrol konkurensi

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC), yang bekerja secara berbeda dari sistem berbasis kunci tradisional. Alih-alih menggunakan kunci, OCC mengevaluasi konflik pada waktu komit. Ketika Aurora DSQL mendeteksi konflik, ia mengembalikan kegagalan serialisasi PostgreSQL dengan kode SQLSTATE. `40001` Pesan respons menyertakan kode OCC yang mengidentifikasi jenis konflik:

OC000 - Konflik data

Dua transaksi mencoba memodifikasi baris yang sama. Transaksi dengan waktu komit paling awal berhasil, dan transaksi yang bertentangan menerima respons OC000:

```
ERROR: change conflicts with another transaction (OC000) (SQLSTATE 40001)
```

OC001 — Skema konflik

Katalog skema cache sesi sudah kedaluwarsa. Ketika Aurora DSQL mendeteksi bahwa versi katalog telah berubah sejak sesi memuat cache-nya, dan transaksi tidak dapat dengan aman melakukan rebase ke versi saat ini, transaksi menerima respons OC001:

```
ERROR: schema has been updated by another transaction (OC001) (SQLSTATE 40001)
```

Setiap operasi yang memodifikasi katalog skema dapat menyebabkan respons OC001, termasuk pernyataan DDL seperti `CREATE TABLE` dan`ALTER TABLE`, serta dan pernyataan. `GRANT` `REVOKE` Untuk informasi selengkapnya, lihat DDL dan transaksi terdistribusi di Aurora DSQL.

Rancang aplikasi Anda untuk menerapkan logika coba lagi untuk menangani respons ini. Pola desain yang ideal adalah idempoten, memungkinkan percobaan ulang transaksi sebagai jalan pertama bila memungkinkan. Logika yang direkomendasikan mirip dengan logika batalkan dan coba lagi dalam batas waktu penguncian PostgreSQL standar atau situasi kebuntuan. Namun, OCC mengharuskan aplikasi Anda untuk menggunakan logika ini lebih sering.

Pedoman untuk mengoptimalkan kinerja transaksi

Untuk mengoptimalkan kinerja, minimalkan pertengkaran tinggi pada tombol tunggal atau rentang kunci kecil. Untuk mencapai tujuan ini, rancang skema Anda untuk menyebarkan pembaruan pada rentang kunci klaster Anda dengan menggunakan pedoman berikut:

- Pilih kunci primer acak untuk tabel Anda.
- Hindari pola yang meningkatkan pertengkaran pada satu tombol. Pendekatan ini memastikan kinerja optimal bahkan ketika volume transaksi tumbuh.

DDL dan transaksi terdistribusi di Aurora DSQL

Bahasa definisi data (DDL) berperilaku berbeda di Aurora DSQL dari PostgreSQL. Aurora DSQL memiliki lapisan database Multi-AZ terdistribusi dan tidak ada yang dibagi yang dibangun di atas armada komputasi dan penyimpanan multi-tenant. Karena tidak ada satu pun node atau pemimpin basis data utama yang ada, katalog database didistribusikan. Dengan demikian, Aurora DSQL mengelola perubahan skema DDL sebagai transaksi terdistribusi.

Secara khusus, DDL berperilaku berbeda di Aurora DSQL sebagai berikut:

Respons kontrol konkurensi

Karena katalog database didistribusikan, Aurora DSQL mengelola perubahan skema DDL sebagai transaksi terdistribusi yang memperbarui versi katalog. Sesi yang memiliki salinan cache katalog pada versi sebelumnya dapat menerima respons kontrol konkurensi dengan

kode SQLSTATE `40001` dan kode OCC `OC001` ketika mereka selanjutnya berinteraksi dengan penyimpanan.

Misalnya, pertimbangkan urutan tindakan berikut:

1. Di sesi 1, pengguna menambahkan kolom ke tabel`mytable`. Ini memperbarui versi katalog.
2. Di sesi 2, pengguna mencoba menyisipkan baris ke dalam`mytable`. Sesi ini masih memiliki versi katalog sebelumnya yang di-cache.

   Aurora DSQL kembali. `SQL Error [40001]: ERROR: schema has been updated by another transaction (OC001)`

> **Note**
>
> Respons OC001 juga dapat terjadi ketika perubahan skema telah selesai sebelum transaksi yang terpengaruh dimulai. Prosesor kueri Aurora DSQL menemukan perubahan katalog secara reaktif selama eksekusi kueri, sehingga sesi yang telah menganggur mungkin masih beroperasi dengan versi katalog basi. Saat mencoba lagi, sesi menyegarkan cache katalognya dan transaksi biasanya berhasil.

DDL dan DML dalam transaksi yang sama

Transaksi di Aurora DSQL hanya dapat berisi satu pernyataan DDL dan tidak dapat memiliki pernyataan DDL dan DHTML. Pembatasan ini berarti Anda tidak dapat membuat tabel dan menyisipkan data ke dalam tabel yang sama dalam transaksi yang sama. Misalnya, Aurora DSQL mendukung transaksi berurutan berikut.

```
BEGIN;
  CREATE TABLE mytable (ID_col integer);
COMMIT;

BEGIN;
  INSERT into FOO VALUES (1);
COMMIT;
```

Aurora DSQL tidak mendukung transaksi berikut, yang mencakup keduanya dan pernyataan. `CREATE INSERT`

```
BEGIN;
  CREATE TABLE FOO (ID_col integer);
  INSERT into FOO VALUES (1);
COMMIT;
```

DDL asinkron

Dalam PostgreSQL standar, operasi DDL `CREATE INDEX` seperti mengunci tabel yang terpengaruh, membuatnya tidak tersedia untuk dibaca dan ditulis dari sesi lain. Di Aurora DSQL, pernyataan DDL ini berjalan secara asinkron menggunakan pengelola latar belakang. Akses ke tabel yang terpengaruh tidak diblokir. Dengan demikian, DDL pada tabel besar dapat berjalan tanpa downtime atau dampak kinerja. Untuk informasi selengkapnya tentang manajer pekerjaan asinkron di Aurora DSQL, lihat. Indeks asinkron di Aurora DSQL

Kunci utama di Aurora DSQL

Dalam Aurora DSQL, kunci utama adalah fitur yang secara fisik mengatur data tabel. Ini mirip dengan `CLUSTER` operasi di PostgreSQL atau indeks berkerumun di database lain. Saat Anda menentukan kunci primer, Aurora DSQL membuat indeks yang mencakup semua kolom dalam tabel. Struktur kunci utama di Aurora DSQL memastikan akses dan manajemen data yang efisien.

Struktur dan penyimpanan data

Saat Anda menentukan kunci primer, Aurora DSQL menyimpan data tabel dalam urutan kunci primer. Struktur terorganisir indeks ini memungkinkan pencarian kunci primer untuk mengambil semua nilai kolom secara langsung, bukan mengikuti pointer ke data seperti dalam indeks tradisional. B-tree Berbeda dengan `CLUSTER` operasi di PostgreSQL, yang mengatur ulang data hanya sekali, Aurora DSQL mempertahankan urutan ini secara otomatis dan terus menerus. Pendekatan ini meningkatkan kinerja kueri yang bergantung pada akses kunci primer.

Aurora DSQL juga menggunakan kunci utama untuk menghasilkan kunci unik di seluruh cluster untuk setiap baris dalam tabel dan indeks. Kunci unik ini juga mendukung manajemen data terdistribusi. Ini memungkinkan partisi data secara otomatis di beberapa node, mendukung penyimpanan yang dapat diskalakan dan konkurensi tinggi. Akibatnya, struktur kunci utama membantu Aurora DSQL menskalakan secara otomatis dan mengelola beban kerja bersamaan secara efisien.

Pedoman untuk memilih kunci utama

Saat memilih dan menggunakan kunci primer di Aurora DSQL, pertimbangkan pedoman berikut:

- Tentukan kunci utama saat Anda membuat tabel. Anda tidak dapat mengubah kunci ini atau menambahkan kunci utama baru nanti. Kunci utama menjadi bagian dari kunci luster-lebar yang digunakan untuk partisi data dan penskalaan otomatis throughput tulis. Jika Anda tidak menentukan kunci utama, Aurora DSQL menetapkan ID tersembunyi sintetis.
- Untuk tabel dengan volume tulis tinggi, hindari penggunaan bilangan bulat yang meningkat secara monoton sebagai kunci utama. Hal ini dapat menyebabkan masalah kinerja dengan mengarahkan semua sisipan baru ke satu partisi. Sebagai gantinya, gunakan kunci primer dengan distribusi acak untuk memastikan distribusi penulisan yang merata di seluruh partisi penyimpanan.
- Untuk tabel yang jarang berubah atau hanya-baca, Anda dapat menggunakan tombol naik. Contoh kunci naik adalah stempel waktu atau nomor urut. Kunci padat memiliki banyak nilai yang berjarak dekat atau duplikat. Anda dapat menggunakan tombol naik meskipun padat karena kinerja tulis kurang kritis.
- Jika pemindaian tabel lengkap tidak memenuhi persyaratan kinerja Anda, pilih metode akses yang lebih efisien. Dalam kebanyakan kasus, ini berarti menggunakan kunci utama yang cocok dengan kunci gabungan dan pencarian Anda yang paling umum dalam kueri.
- Ukuran gabungan maksimum kolom dalam kunci primer adalah 1 kibibyte. Untuk informasi selengkapnya, lihat Batas basis data di Aurora DSQL dan tipe data yang didukung di Aurora DSQL.
- Anda dapat menyertakan hingga 8 kolom dalam kunci primer atau indeks sekunder. Untuk informasi selengkapnya, lihat Batas basis data di Aurora DSQL dan tipe data yang didukung di Aurora DSQL.

Urutan dan kolom identitas

Urutan dan kolom identitas menghasilkan nilai integer dan berguna ketika pengidentifikasi kompak atau dapat dibaca manusia diperlukan. Nilai-nilai ini melibatkan alokasi dan perilaku caching yang dijelaskan dalam dokumentasi. `CREATE SEQUENCE`

Topik

- Fungsi manipulasi urutan
- Kolom identitas
- Bekerja dengan urutan dan kolom identitas

Fungsi manipulasi urutan

Bagian ini menjelaskan fungsi untuk beroperasi pada objek urutan, juga disebut generator urutan atau hanya urutan. Objek urutan adalah tabel baris tunggal khusus yang dibuat dengan`CREATE SEQUENCE`. Objek urutan biasanya digunakan untuk menghasilkan pengidentifikasi unik untuk baris tabel. Fungsi urutan menyediakan metode sederhana yang aman untuk multipengguna untuk mendapatkan nilai urutan berturut-turut dari objek urutan.

> **Important**
>
> Saat menggunakan urutan, nilai cache harus dipertimbangkan dengan cermat. Untuk informasi selengkapnya, lihat info penting di `CREATE SEQUENCE` halaman.
> Untuk panduan tentang cara terbaik menggunakan urutan berdasarkan pola beban kerja, lihat. Bekerja dengan urutan dan kolom identitas

Fungsi	Deskripsi
`nextval ( regclass ) # bigint`	Memajukan objek urutan ke nilai berikutnya dan mengembalikan nilai itu. Ini dilakukan secara atom: bahkan jika beberapa sesi berjalan secara `nextval` bersamaan, masing-masing akan dengan aman menerima nilai urutan yang berbeda. Jika objek urutan telah dibuat dengan parameter default, `nextval` panggilan berturut-turut akan mengembalikan nilai yang meningkat dimulai dengan 1. Perilaku lain dapat diperoleh dengan menggunakan parameter yang sesuai dalam `CREATE SEQUENCE` perintah. Fungsi ini membutuhkan USAGE atau UPDATE hak istimewa pada urutan.
`setval ( regclass, bigint [, boolean ] ) # bigint`	Menetapkan nilai objek urutan saat ini, dan opsional `is_called` benderanya. Bentuk dua parameter menetapkan `last_value` bidang urutan ke nilai yang ditentukan dan menetapkan `is_called` bidangnya `true`, yang berarti bahwa yang berikutnya `nextval` akan memajukan urutan sebelum mengembalikan nilai. Nilai yang akan dilaporkan oleh juga `currval`

Fungsi	Deskripsi

	diatur ke nilai yang ditentukan. Dalam bentuk tiga parameter, `is_called` dapat diatur ke salah satu `true` atau`false`. `true`memiliki efek yang sama dengan bentuk dua parameter. Jika diatur ke`false`, berikutnya `nextval` akan mengembalikan persis nilai yang ditentukan, dan kemajuan urutan dimulai dengan yang berikut ini. `nextval` Selain itu, nilai yang dilaporkan oleh `currval` tidak diubah di sini. Contoh:
	`SELECT setval('myseq', 42);  -- Next nextval will return 43`
	`SELECT setval('myseq', 42, true);  -- Same as above`
	`SELECT setval('myseq', 42, false);  -- Next nextval will return 42`
	Hasil yang dikembalikan oleh `setval` hanyalah nilai dari argumen keduanya. Fungsi ini membutuhkan `UPDATE` hak istimewa pada urutan.

`currval ( regclass ) # bigint`	Mengembalikan nilai yang paling baru diperoleh `nextval` untuk urutan ini dalam sesi saat ini. (Kesalahan dilaporkan jika tidak `nextval` pernah dipanggil untuk urutan ini dalam sesi ini.) Karena ini mengembalikan nilai session-local, ini memberikan jawaban yang dapat diprediksi apakah sesi lain telah berjalan atau tidak `nextval` karena sesi saat ini melakukannya. Fungsi ini membutuhkan `USAGE` atau `SELECT` hak istimewa pada urutan.

Fungsi	Deskripsi
`lastval () # bigint`	Mengembalikan nilai yang terakhir dikembalikan oleh `nextval` dalam transaksi saat ini. Fungsi ini identik dengan`currval`, kecuali bahwa alih-alih mengambil nama urutan sebagai argumen, ini mengacu pada urutan `nextval` mana pun yang paling baru diterapkan dalam transaksi saat ini. Ini adalah kesalahan untuk menelepon `lastval` jika `nextval` belum dipanggil dalam transaksi saat ini. Fungsi ini membutuhkan USAGE atau SELECT hak istimewa pada urutan yang terakhir digunakan.

> **Warning**
>
> Nilai yang diperoleh `nextval` tidak direklamasi untuk digunakan kembali jika transaksi pemanggilan kemudian dibatalkan. Ini berarti bahwa transaksi dibatalkan atau kerusakan database dapat mengakibatkan kesenjangan dalam urutan nilai yang ditetapkan. Itu bisa terjadi tanpa membatalkan transaksi juga. Misalnya, `ON CONFLICT` klausa `INSERT` with an akan menghitung to-be-inserted tuple, termasuk melakukan `nextval` panggilan yang diperlukan, sebelum mendeteksi konflik apa pun yang akan menyebabkannya mengikuti aturan sebagai gantinya. `ON CONFLICT` Dengan demikian, objek urutan Aurora DSQL tidak dapat digunakan untuk mendapatkan urutan “tanpa celah”.
> Demikian juga, perubahan status urutan `setval` yang dibuat oleh segera terlihat oleh transaksi lain, dan tidak dibatalkan jika transaksi panggilan kembali.

Urutan yang akan dioperasikan oleh fungsi urutan ditentukan oleh `regclass` argumen, yang hanyalah OID dari urutan dalam katalog `pg_class` sistem. Namun, Anda tidak perlu mencari OID dengan tangan, karena konverter input tipe `regclass` data akan melakukan pekerjaan untuk Anda. Lihat dokumentasi PostgreSQL tentang Jenis Pengenal Objek untuk detailnya.

Kolom identitas

> **Important**
>
> Saat menggunakan kolom identitas, nilai cache harus dipertimbangkan dengan cermat. Untuk informasi selengkapnya, lihat info penting di `CREATE SEQUENCE` halaman.
> Untuk panduan tentang cara terbaik menggunakan kolom identitas berdasarkan pola beban kerja, lihatBekerja dengan urutan dan kolom identitas.

Kolom identitas adalah kolom khusus yang dihasilkan secara otomatis dari urutan implisit. Hal ini dapat digunakan untuk menghasilkan nilai-nilai kunci. Untuk membuat kolom identitas, gunakan `GENERATED ... AS IDENTITY` klausa di`CREATE TABLE`, misalnya:

```
CREATE TABLE people (
    id bigint GENERATED ALWAYS AS IDENTITY (CACHE 70000),
    ...
);
```

atau sebagai alternatif:

```
CREATE TABLE people (
    id bigint GENERATED BY DEFAULT AS IDENTITY (CACHE 70000),
    ...
);
```

Lihat `CREATE TABLE` untuk detail selengkapnya.

Jika `INSERT` perintah dijalankan pada tabel dengan kolom identitas dan tidak ada nilai yang secara eksplisit ditentukan untuk kolom identitas, maka nilai yang dihasilkan oleh urutan implisit dimasukkan. Misalnya, dengan definisi sebelumnya dan mengasumsikan kolom tambahan yang sesuai, menulis:

```
INSERT INTO people (name, address) VALUES ('A', 'foo');
INSERT INTO people (name, address) VALUES ('B', 'bar');
```

akan menghasilkan nilai untuk `id` kolom mulai dari 1 dan menghasilkan data tabel berikut:

```
 id | name | address
----+------+---------
```

```
 1 | A    | foo
 2 | B    | bar
```

Atau, kata kunci DEFAULT dapat ditentukan sebagai pengganti nilai untuk secara eksplisit meminta nilai yang dihasilkan urutan:

```
INSERT INTO people (id, name, address) VALUES (DEFAULT, 'C', 'baz');
```

Demikian pula, kata kunci DEFAULT dapat digunakan dalam UPDATE perintah.

Jadi, dalam banyak hal, kolom identitas berperilaku seperti kolom dengan nilai default.

Klausa ALWAYS dan BY DEFAULT definisi kolom menentukan bagaimana nilai yang ditentukan pengguna secara eksplisit ditangani dan perintah. INSERT UPDATE Dalam sebuah INSERT perintah, jika ALWAYS dipilih, nilai yang ditentukan pengguna hanya diterima jika INSERT pernyataan menentukan. OVERRIDING SYSTEM VALUE Jika BY DEFAULT dipilih, maka nilai yang ditentukan pengguna diutamakan. Dengan demikian, menggunakan BY DEFAULT hasil dalam perilaku yang lebih mirip dengan nilai default, di mana nilai default dapat diganti dengan nilai eksplisit, sedangkan ALWAYS memberikan beberapa perlindungan lebih terhadap memasukkan nilai eksplisit secara tidak sengaja.

Tipe data kolom identitas harus menjadi salah satu tipe data yang didukung oleh urutan. (Lihat CREATE SEQUENCE). Properti dari urutan terkait dapat ditentukan saat membuat kolom identitas (lihatCREATE TABLE) atau diubah setelahnya (lihatALTER TABLE).

Kolom identitas secara otomatis ditandai sebagaiNOT NULL. Kolom identitas, bagaimanapun, tidak menjamin keunikan. (Urutan biasanya mengembalikan nilai unik, tetapi urutan dapat diatur ulang, atau nilai dapat dimasukkan secara manual ke dalam kolom identitas, seperti yang dibahas sebelumnya.) Keunikan perlu ditegakkan menggunakan a PRIMARY KEY atau kendala. UNIQUE

Bekerja dengan urutan dan kolom identitas

Bagian ini membantu Anda memahami cara terbaik untuk menggunakan urutan dan kolom identitas berdasarkan pola beban kerja.

> **Important**
>
> Lihat info penting di CREATE SEQUENCE halaman untuk detail selengkapnya tentang alokasi dan perilaku caching.

Memilih jenis pengenal

Amazon Aurora DSQL mendukung pengidentifikasi berbasis UUID dan nilai integer yang dihasilkan menggunakan urutan atau kolom identitas. Opsi-opsi ini berbeda dalam bagaimana nilai dialokasikan dan bagaimana skalanya di bawah beban.

Nilai UUID dapat dihasilkan tanpa koordinasi dan sangat cocok untuk beban kerja di mana pengidentifikasi sering dibuat atau di banyak sesi. Karena Amazon Aurora DSQL dirancang untuk operasi terdistribusi, menghindari koordinasi seringkali bermanfaat. Untuk alasan ini, UUIDs direkomendasikan sebagai tipe pengenal default, terutama untuk kunci utama dalam beban kerja di mana skalabilitas penting dan urutan pengenal yang ketat tidak diperlukan.

Urutan dan kolom identitas menghasilkan nilai bilangan bulat kompak yang nyaman untuk pengidentifikasi, pelaporan, dan antarmuka eksternal yang dapat dibaca manusia. Ketika pengidentifikasi numerik lebih disukai karena alasan kegunaan atau integrasi, pertimbangkan untuk menggunakan kolom urutan atau identitas dalam kombinasi dengan pengidentifikasi berbasis UUID. Ketika urutan bilangan bulat atau nilai identitas diperlukan, memilih ukuran cache yang sesuai menjadi bagian penting dari desain beban kerja. Lihat bagian berikut untuk panduan memilih ukuran cache.

Memilih ukuran cache

Memilih nilai cache yang sesuai adalah bagian penting dari penggunaan urutan dan kolom identitas secara efektif. Pengaturan cache menentukan bagaimana alokasi pengidentifikasi berperilaku di bawah beban, mempengaruhi throughput sistem dan seberapa dekat nilai mencerminkan urutan alokasi.

Ukuran cache yang lebih besar **`CACHE >= 65536`** sangat cocok ketika:

- Pengidentifikasi dihasilkan pada frekuensi tinggi
- Banyak sesi menyisipkan secara bersamaan
- Beban kerja dapat mentolerir kesenjangan dan efek pemesanan yang terlihat

Misalnya, beban kerja konsumsi peristiwa volume tinggi (seperti IoT atau telemetri), serta pengidentifikasi operasional seperti menjalankan pekerjaan, referensi kasus dukungan IDs, atau nomor pesanan internal biasanya mendapat manfaat dari ukuran cache yang lebih besar, di mana pengidentifikasi sering dibuat dan pemesanan yang ketat tidak diperlukan.

Ukuran cache 1 lebih selaras ketika:

- Tingkat alokasi relatif rendah
- Pengidentifikasi diharapkan mengikuti urutan alokasi lebih dekat dari waktu ke waktu
- Meminimalkan kesenjangan lebih penting daripada throughput maksimum

Beban kerja seperti menetapkan akun atau nomor referensi, di mana pengidentifikasi dihasilkan lebih jarang dan pemesanan lebih dekat diinginkan, lebih baik diselaraskan dengan ukuran cache 1.

Indeks asinkron di Aurora DSQL

`CREATE INDEX ASYNC`Perintah membuat indeks pada satu atau lebih kolom dari tabel tertentu. Perintah ini adalah operasi DDL asinkron yang tidak memblokir transaksi lain. Ketika Anda menjalankan`CREATE INDEX ASYNC`, Aurora DSQL segera mengembalikan file. `job_id`

Anda dapat memantau status pekerjaan asinkron ini menggunakan tampilan sistem. `sys.jobs` Saat pekerjaan pembuatan indeks sedang berlangsung, Anda dapat menggunakan prosedur dan perintah ini:

`sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`*

Memblokir sesi saat ini sampai pekerjaan yang ditentukan selesai atau gagal. Mengembalikan nilai Boolean yang menunjukkan keberhasilan atau kegagalan.

`DROP INDEX`

Membatalkan pekerjaan pembuatan indeks yang sedang berlangsung.

Ketika pembuatan indeks asinkron selesai, Aurora DSQL memperbarui katalog sistem untuk menandai indeks sebagai aktif.

> **Note**
>
> Perhatikan bahwa transaksi bersamaan yang mengakses objek di namespace yang sama selama pembaruan ini mungkin mengalami kesalahan konkurensi.

Ketika Aurora DSQL menyelesaikan tugas indeks asinkron, Aurora akan memperbarui katalog sistem untuk menunjukkan bahwa indeks aktif. Jika transaksi lain mereferensikan objek di namespace yang sama saat ini, Anda mungkin melihat kesalahan konkurensi.

Sintaksis

`CREATE INDEX ASYNC`menggunakan sintaks berikut.

```
CREATE [ UNIQUE ] INDEX ASYNC [ IF NOT EXISTS ] name ON table_name
     ( { column_name } [ NULLS { FIRST | LAST } ] )
     [ INCLUDE ( column_name [, ...] ) ]
     [ NULLS [ NOT ] DISTINCT ]
```

Parameter

`UNIQUE`

Menunjukkan ke Aurora DSQL untuk memeriksa nilai duplikat dalam tabel saat membuat indeks dan setiap kali Anda menambahkan data. Jika Anda menentukan parameter ini, menyisipkan dan memperbarui operasi yang akan menghasilkan entri duplikat menghasilkan kesalahan.

`IF NOT EXISTS`

Menunjukkan bahwa Aurora DSQL seharusnya tidak melempar pengecualian jika indeks dengan nama yang sama sudah ada. Dalam situasi ini, Aurora DSQL tidak membuat indeks baru. Perhatikan bahwa indeks yang Anda coba buat bisa memiliki struktur yang sangat berbeda dari indeks yang ada. Jika Anda menentukan parameter ini, nama indeks diperlukan.

`name`

Nama indeks. Anda tidak dapat menyertakan nama skema Anda dalam parameter ini.

Aurora DSQL membuat indeks dalam skema yang sama dengan tabel induknya. Nama indeks harus berbeda dari nama objek lain, seperti tabel atau indeks, dalam skema.

Jika Anda tidak menentukan nama, Aurora DSQL menghasilkan nama secara otomatis berdasarkan nama tabel induk dan kolom yang diindeks. Misalnya, jika Anda menjalankan`CREATE INDEX ASYNC on table1 (col1, col2)`, Aurora DSQL secara otomatis memberi nama indeks. `table1_col1_col2_idx`

`NULLS FIRST | LAST`

Urutan urutan kolom null dan non-null. `FIRST`menunjukkan bahwa Aurora DSQL harus mengurutkan kolom null sebelum kolom non-null. `LAST`menunjukkan bahwa Aurora DSQL harus mengurutkan kolom null setelah kolom non-null.

`INCLUDE`

Daftar kolom untuk disertakan dalam indeks sebagai kolom non-kunci. Anda tidak dapat menggunakan kolom non-kunci dalam kualifikasi pencarian pemindaian indeks. Aurora DSQL mengabaikan kolom dalam hal keunikan untuk indeks.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Menentukan apakah Aurora DSQL harus mempertimbangkan nilai null sebagai berbeda dalam indeks unik. Defaultnya adalah`DISTINCT`, yang berarti bahwa indeks unik dapat berisi beberapa nilai null dalam kolom. `NOT DISTINCT`menunjukkan bahwa indeks tidak dapat berisi beberapa nilai null dalam kolom.

Catatan penggunaan

Pertimbangkan panduan berikut ini:

- `CREATE INDEX ASYNC`Perintah tidak memperkenalkan kunci. Ini juga tidak mempengaruhi tabel dasar yang Aurora DSQL gunakan untuk membuat indeks.
- Selama operasi migrasi skema, `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`* prosedur ini berguna. Ini memastikan bahwa operasi DDL dan DML berikutnya menargetkan indeks yang baru dibuat.
- Setiap kali Aurora DSQL menjalankan tugas asinkron baru, ia memeriksa `sys.jobs` tampilan dan menghapus tugas yang memiliki status atau selama lebih dari 30 menit. `completed` `failed` Dengan demikian, `sys.jobs` terutama menunjukkan tugas yang sedang berlangsung dan tidak berisi informasi tentang tugas-tugas lama.
- Jika Aurora DSQL gagal membangun indeks asinkron, indeks tetap ada. `INVALID` Untuk indeks unik, operasi DHTML tunduk pada kendala keunikan sampai Anda menjatuhkan indeks. Kami menyarankan Anda menjatuhkan indeks yang tidak valid dan membuatnya kembali.

Membuat indeks: contoh

Contoh berikut menunjukkan cara membuat skema, tabel, dan kemudian indeks.

1. Buat tabel bernama`test.departments`.

```
CREATE SCHEMA test;

CREATE TABLE test.departments (name varchar(255) primary key NOT null,
     manager varchar(255),
     size varchar(4));
```

2. Masukkan baris ke dalam tabel.

```
INSERT INTO test.departments VALUES ('Human Resources', 'John Doe', '10')
```

3. Buat indeks asinkron.

```
CREATE INDEX ASYNC test_index on test.departments(name, manager, size);
```

   `CREATE INDEX`Perintah mengembalikan ID pekerjaan, seperti yang ditunjukkan di bawah ini.

```
job_id
--------------------------
jh2gbtx4mzhgfkbimtgwn5j45y
```

   `job_id`Ini menunjukkan bahwa Aurora DSQL telah mengirimkan pekerjaan baru untuk membuat indeks. Anda dapat menggunakan prosedur `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* untuk memblokir pekerjaan lain pada sesi sampai pekerjaan selesai atau habis waktu.

Menanyakan status pembuatan indeks: contoh

Kueri tampilan `sys.jobs` sistem untuk memeriksa status pembuatan indeks Anda, seperti yang ditunjukkan pada contoh berikut.

```
SELECT * FROM sys.jobs where job_id = 'wqhu6ewifze5xitg3umt24h5ua';
```

Aurora DSQL mengembalikan respon yang mirip dengan berikut ini.

```
           job_id           |  status   | details |  job_type   | class_id | object_id
 |    object_name    |       start_time       |      update_time
----------------------------+-----------+---------+-------------+----------+-----------
+------------------+------------------------+------------------------
 wqhu6ewifze5xitg3umt24h5ua | completed |         | INDEX_BUILD |     1259 |     26433
 | public.nt2_c1_idx | 2025-09-25 22:07:31+00 | 2025-09-25 22:07:46+00
```

Kolom status dapat menjadi salah satu nilai berikut.

Status	Deskripsi
`submitted`	Tugas dikirimkan, tetapi Aurora DSQL belum mulai memprosesnya.
`processing`	Aurora DSQL sedang memproses tugas.
`failed`	Tugas gagal. Lihat kolom detail untuk informasi lebih lanjut. Jika Aurora DSQL gagal membangun indeks, Aurora DSQL tidak secara otomatis menghapus definisi indeks. Anda harus menghapus indeks secara manual dengan `DROP INDEX` perintah.
`completed`	Aurora DSQL telah menyelesaikan tugas dengan sukses.

Anda juga dapat menanyakan status indeks melalui tabel katalog `pg_index` dan`pg_class`. Secara khusus, atribut `indisvalid` dan `indisimmediate` dapat memberi tahu Anda status indeks Anda. Sementara Aurora DSQL membuat indeks Anda, ia memiliki status awal. `INVALID` `indisvalid`Bendera untuk indeks mengembalikan `FALSE` atau`f`, yang menunjukkan bahwa indeks tidak valid. Jika bendera kembali `TRUE` atau`t`, indeks sudah siap.

```
SELECT relname AS index_name, indisvalid as is_valid, pg_get_indexdef(indexrelid) AS
 index_definition
from pg_index, pg_class
WHERE pg_class.oid = indexrelid AND indrelid = 'test.departments'::regclass;
```

```
    index_name    | is_valid |
 index_definition
------------------+----------
+-------------------------------------------------------------------------------------------------
 department_pkey  |     t    | CREATE UNIQUE INDEX department_pkey ON test.departments
 USING btree_index (title) INCLUDE (name, manager, size)
 test_index1      |     t    | CREATE INDEX test_index1 ON test.departments USING
 btree_index (name, manager, size)
```

Kegagalan pembuatan indeks unik

Jika pekerjaan pembuatan indeks unik asinkron Anda menunjukkan status gagal dengan detailFound `duplicate key while validating index for UCVs`, ini menunjukkan bahwa indeks unik tidak dapat dibuat karena pelanggaran batasan keunikan.

Untuk mengatasi kegagalan pembuatan indeks unik

1. Hapus baris apa pun di tabel utama Anda yang memiliki entri duplikat untuk kunci yang ditentukan dalam indeks sekunder unik Anda.
2. Jatuhkan indeks yang gagal.
3. Keluarkan perintah buat indeks baru.

Mendeteksi pelanggaran keunikan di tabel utama

Kueri SQL berikut membantu Anda mengidentifikasi nilai duplikat dalam kolom tertentu dari tabel Anda. Ini sangat berguna ketika Anda perlu menerapkan keunikan pada kolom yang saat ini tidak ditetapkan sebagai kunci utama atau tidak memiliki kendala unik, seperti alamat email dalam tabel pengguna.

Contoh di bawah ini menunjukkan cara membuat tabel pengguna sampel, mengisinya dengan data uji yang berisi duplikat yang diketahui, dan kemudian menjalankan kueri deteksi.

Tentukan skema tabel

```
-- Drop the table if it exists
DROP TABLE IF EXISTS users;

-- Create the users table with a simple integer primary key
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY,
```

```
    email VARCHAR(255),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Masukkan data sampel yang mencakup kumpulan alamat email duplikat

```
-- Insert sample data with explicit IDs
INSERT INTO users (user_id, email, first_name, last_name) VALUES
    (1, 'john.doe@example.com', 'John', 'Doe'),
    (2, 'jane.smith@example.com', 'Jane', 'Smith'),
    (3, 'john.doe@example.com', 'Johnny', 'Doe'),
    (4, 'alice.wong@example.com', 'Alice', 'Wong'),
    (5, 'bob.jones@example.com', 'Bob', 'Jones'),
    (6, 'alice.wong@example.com', 'Alicia', 'Wong'),
    (7, 'bob.jones@example.com', 'Robert', 'Jones');
```

Jalankan kueri deteksi duplikat

```
-- Query to find duplicates
WITH duplicates AS (
    SELECT email, COUNT(*) as duplicate_count
    FROM users
    GROUP BY email
    HAVING COUNT(*) > 1
)
SELECT u.*, d.duplicate_count
FROM users u
INNER JOIN duplicates d ON u.email = d.email
ORDER BY u.email, u.user_id;
```

Lihat semua catatan dengan alamat email duplikat

```
 user_id |         email          | first_name | last_name |         created_at
 | duplicate_count
---------+------------------------+------------+-----------
+----------------------------+-----------------
       4 | akua.mansa@example.com | Akua       | Mansa     | 2025-05-21 20:55:53.714432
 |                2
       6 | akua.mansa@example.com | Akua       | Mansa     | 2025-05-21 20:55:53.714432
 |                2
```

```
       1 | john.doe@example.com   | John       | Doe       | 2025-05-21 20:55:53.714432
 |               2
       3 | john.doe@example.com   | Johnny     | Doe       | 2025-05-21 20:55:53.714432
 |               2
(4 rows)
```

Jika kita mencoba pernyataan pembuatan indeks sekarang, itu akan gagal:

```
postgres=> CREATE UNIQUE INDEX ASYNC idx_users_email ON users(email);
      job_id
----------------------------
 ve32upmjz5dgdknpbleeca5tri
(1 row)

postgres=> select * from sys.jobs;
           job_id           |  status   |                       details
       |  job_type   | class_id | object_id |      object_name       |       start_time
       |      update_time
----------------------------+-----------
+-----------------------------------------------------+-------------+----------
+-----------+------------------------+------------------------+------------------------
 qpn6aqlkijgmzilyidcpwrpova | completed |
          | DROP        |     1259 |     26384 |                        | 2025-05-20
 00:47:10+00 | 2025-05-20 00:47:32+00
 ve32upmjz5dgdknpbleeca5tri | failed    | Found duplicate key while validating index
 for UCVs | INDEX_BUILD |     1259 |     26396 | public.idx_users_email | 2025-05-20
 00:49:49+00 | 2025-05-20 00:49:56+00
(2 rows)
```

Tabel dan perintah sistem di Aurora DSQL

Lihat bagian berikut untuk mempelajari tentang tabel dan katalog sistem yang didukung di Aurora DSQL serta kueri yang berguna untuk mengambil informasi tentang sistem, seperti versi.

Tabel sistem

Aurora DSQL kompatibel dengan PostgreSQL, begitu banyak tabel katalog sistem dan tampilan dari PostgreSQL juga ada di Aurora DSQL.

Tabel dan tampilan katalog PostgreSQL penting

Tabel berikut menjelaskan tabel dan tampilan paling umum yang mungkin Anda gunakan di Aurora DSQL.

Nama	Deskripsi
`pg_namespace`	Informasi tentang semua skema
`pg_tables`	Informasi tentang semua tabel
`pg_attribute`	Informasi tentang semua atribut
`pg_views`	Informasi tentang (pra-) tampilan yang ditentukan
`pg_class`	Menjelaskan semua tabel, kolom, indeks, dan objek serupa
`pg_stats`	Pandangan tentang statistik perencana
`pg_user`	Informasi tentang pengguna
`pg_roles`	Informasi tentang pengguna dan grup
`pg_indexes`	Daftar semua indeks
`pg_constraint`	Daftar kendala pada tabel

Tabel katalog yang didukung dan tidak didukung

Tabel berikut menunjukkan tabel mana yang didukung dan tidak didukung di Aurora DSQL.

Nama	Berlaku untuk Aurora DSQL
`pg_aggregate`	Tidak
`pg_am`	Ya
`pg_amop`	Tidak
`pg_amproc`	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_attrdef	Ya
pg_attribute	Ya
pg_authid	Tidak (gunakanpg_roles)
pg_auth_members	Ya
pg_cast	Ya
pg_class	Ya
pg_collation	Ya
pg_constraint	Ya
pg_conversion	Tidak
pg_database	Tidak
pg_db_role_setting	Ya
pg_default_acl	Ya
pg_depend	Ya
pg_description	Ya
pg_enum	Tidak
pg_event_trigger	Tidak
pg_extension	Tidak
pg_foreign_data_wrapper	Tidak
pg_foreign_server	Tidak
pg_foreign_table	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_index	Ya
pg_inherits	Ya
pg_init_privs	Tidak
pg_language	Tidak
pg_largeobject	Tidak
pg_largeobject_metadata	Ya
pg_namespace	Ya
pg_opclass	Tidak
pg_operator	Ya
pg_opfamily	Tidak
pg_parameter_acl	Ya
pg_partitioned_table	Tidak
pg_policy	Tidak
pg_proc	Tidak
pg_publication	Tidak
pg_publication_namespace	Tidak
pg_publication_rel	Tidak
pg_range	Ya
pg_replication_origin	Tidak
pg_rewrite	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_seclabel	Tidak
pg_sequence	Tidak
pg_shdepend	Ya
pg_shdescription	Ya
pg_shseclabel	Tidak
pg_statistic	Ya
pg_statistic_ext	Tidak
pg_statistic_ext_data	Tidak
pg_subscription	Tidak
pg_subscription_rel	Tidak
pg_tablespace	Tidak
pg_transform	Tidak
pg_trigger	Tidak
pg_ts_config	Ya
pg_ts_config_map	Ya
pg_ts_dict	Ya
pg_ts_parser	Ya
pg_ts_template	Ya
pg_type	Ya
pg_user_mapping	Tidak

Tampilan sistem yang didukung dan tidak didukung

Tabel berikut menunjukkan tampilan mana yang didukung dan tidak didukung di Aurora DSQL.

Nama	Berlaku untuk Aurora DSQL
pg_available_extensions	Tidak
pg_available_extension_versions	Tidak
pg_backend_memory_contexts	Ya
pg_config	Tidak
pg_cursors	Tidak
pg_file_settings	Tidak
pg_group	Ya
pg_hba_file_rules	Tidak
pg_ident_file_mappings	Tidak
pg_indexes	Ya
pg_locks	Tidak
pg_matviews	Tidak
pg_policies	Tidak
pg_prepared_statements	Tidak
pg_prepared_xacts	Tidak
pg_publication_tables	Tidak
pg_replication_origin_status	Tidak
pg_replication_slots	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_roles	Ya
pg_rules	Tidak
pg_seclabels	Tidak
pg_sequences	Tidak
pg_settings	Ya
pg_shadow	Ya
pg_shmem_allocations	Ya
pg_stats	Ya
pg_stats_ext	Tidak
pg_stats_ext_exprs	Tidak
pg_tables	Ya
pg_timezone_abbrevs	Ya
pg_timezone_names	Ya
pg_user	Ya
pg_user_mappings	Tidak
pg_views	Ya
pg_stat_activity	Tidak
pg_stat_replication	Tidak
pg_stat_replication_slots	Tidak
pg_stat_wal_receiver	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_stat_recovery_prefetch	Tidak
pg_stat_subscription	Tidak
pg_stat_subscription_stats	Tidak
pg_stat_ssl	Ya
pg_stat_gssapi	Tidak
pg_stat_archiver	Tidak
pg_stat_io	Tidak
pg_stat_bgwriter	Tidak
pg_stat_wal	Tidak
pg_stat_database	Tidak
pg_stat_database_conflicts	Tidak
pg_stat_all_tables	Tidak
pg_stat_all_indexes	Tidak
pg_statio_all_tables	Tidak
pg_statio_all_indexes	Tidak
pg_statio_all_sequences	Tidak
pg_stat_slru	Tidak
pg_statio_user_tables	Tidak
pg_statio_user_sequences	Tidak
pg_stat_user_functions	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_stat_user_indexes	Tidak
pg_stat_progress_analyze	Tidak
pg_stat_progress_basebackup	Tidak
pg_stat_progress_cluster	Tidak
pg_stat_progress_create_index	Tidak
pg_stat_progress_vacuum	Tidak
pg_stat_sys_indexes	Tidak
pg_stat_sys_tables	Tidak
pg_stat_xact_all_tables	Tidak
pg_stat_xact_sys_tables	Tidak
pg_stat_xact_user_functions	Tidak
pg_stat_xact_user_tables	Tidak
pg_statio_sys_indexes	Tidak
pg_statio_sys_sequences	Tidak
pg_statio_sys_tables	Tidak
pg_statio_user_indexes	Tidak

Tampilan sys.jobs

`sys.jobs`memberikan informasi status tentang pekerjaan asinkron. Misalnya, setelah Anda membuat indeks asinkron, Aurora DSQL mengembalikan indeks asinkron. `job_uuid` Anda dapat menggunakan ini `job_uuid` `sys.jobs` untuk mencari status pekerjaan.

```
SELECT * FROM sys.jobs;
```

Aurora DSQL mengembalikan respon yang mirip dengan berikut ini.

```
           job_id           |  status   | details |  job_type   | class_id | object_id
 |    object_name    |       start_time       |      update_time
----------------------------+-----------+---------+-------------+----------+-----------
+------------------+------------------------+------------------------
 wqhu6ewifze5xitg3umt24h5ua | completed |         | INDEX_BUILD |     1259 |     26433
 | public.nt2_c1_idx | 2025-09-25 22:07:31+00 | 2025-09-25 22:07:46+00
 kkngzf33dndl3daacxehpx5eba | completed |         | ANALYZE     |     1259 |     26419
 | public.nt         | 2025-09-25 21:57:05+00 | 2025-09-25 21:57:27+00
 fyopxjb6ovdn7po6lrkj63cyea | completed |         | DROP        |     1259 |     26422
 |                   | 2025-09-25 22:05:57+00 | 2025-09-25 22:06:03+00
```

Tabel berikut menjelaskan kolom dalam `sys.jobs` tampilan.

sys.jobs lihat kolom

Kolom	Jenis	Deskripsi
`job_id`	`text`	UUID basis-32 yang mewakili pekerjaan.
`status`	`text`	Status pekerjaan saat ini. Nilai yang mungkin adalah `submitted` , `processing` , `completed` , dan `failed`. Untuk informasi selengkapnya, lihat nilai status sys.jobs.
`details`	`text`	Setiap detail yang relevan tentang pekerjaan itu. Jika pekerjaan gagal, alasan terperinci disediakan.
`job_type`	`text`	Jenis pekerjaan asinkron. Nilai yang mungkin adalah: `INDEX_BUILD` — build indeks asinkron. `ANALYZE`— pekerjaan analisis otomatis yang dikirimkan sistem. `DROP`Menghapus data fisik setelah `DROP INDEX` operasi `DROP TABLE` atau
`class_id`	`oid`	OID dari tabel katalog yang berisi objek.
`object_id`	`oid`	OID dari objek.

Kolom	Jenis	Deskripsi
`object_name`	`text`	Nama objek yang sepenuhnya memenuhi syarat. DR0Ppekerjaan tidak dapat mereferensikan objek yang sudah dijatuhkan. Jika objek yang direferensikan telah dijatuhkan, `object_name` mungkin NULL.
`start_time`	`timestamp with time zone`	Stempel waktu di mana pekerjaan itu diserahkan.
`update_time`	`timestamp with time zone`	Stempel waktu di mana baris pekerjaan terakhir diperbarui.

nilai status sys.jobs

Status	Deskripsi
`submitted`	Tugas dikirimkan, tetapi Aurora DSQL belum mulai memprosesnya.
`processing`	Aurora DSQL sedang memproses tugas.
`failed`	Tugas gagal. Lihat `details` kolom untuk informasi lebih lanjut.
`completed`	Aurora DSQL telah menyelesaikan tugas dengan sukses.

Tampilan sys.iam_pg_role_mappings

Tampilan `sys.iam_pg_role_mappings` memberikan informasi tentang izin yang diberikan kepada pengguna IAM. Misalnya, jika `DQSLDBConnect` peran IAM yang memberikan Aurora DSQL akses ke non-admin dan pengguna `testuser` bernama diberikan `DQSLDBConnect` peran dan izin yang sesuai, Anda dapat menanyakan tampilan untuk melihat pengguna mana yang diberikan izin `sys.iam_pg_role_mappings` mana.

```
SELECT * FROM sys.iam_pg_role_mappings;
```

Kueri metadata sistem yang berguna

Gunakan kueri ini untuk mendapatkan statistik tabel dan metadata sistem tanpa melakukan operasi mahal seperti pemindaian tabel penuh.

Dapatkan perkiraan jumlah baris untuk tabel

Untuk mendapatkan perkiraan hitungan baris dalam tabel tanpa melakukan pemindaian tabel lengkap, gunakan kueri berikut:

```
SELECT reltuples FROM pg_class WHERE relname = 'table_name';
```

Perintah tersebut mengembalikan output serupa dengan berikut ini:

```
  reltuples
--------------
 9.993836e+08
```

Pendekatan ini lebih efisien daripada `SELECT COUNT(*)` tabel besar di Aurora DSQL.

Dapatkan versi utama Aurora DSQL saat ini

Untuk mendapatkan versi utama cluster Aurora DSQL saat ini, gunakan kueri berikut:

```
SELECT * FROM sys.dsql_major_version();
```

Perintah tersebut mengembalikan output serupa dengan berikut ini:

```
 dsql_major_version
--------------------
                  1
```

Ini mengembalikan versi utama koneksi SQL di Aurora DSQL.

Dapatkan versi PostgreSQL saat ini

Untuk mendapatkan versi PostgreSQL cluster Aurora DSQL saat ini, gunakan kueri berikut:

```
SHOW server_version;
```

Perintah tersebut mengembalikan output serupa dengan berikut ini:

```
 server_version
----------------
 16.13
```

Ini mengembalikan versi PostgreSQL koneksi SQL di Aurora DSQL.

Perintah **ANALYSIS**

`ANALYZE` Perintah mengumpulkan statistik tentang isi tabel dalam database dan menyimpan hasilnya dalam tampilan `pg_stats` sistem. Selanjutnya, perencana kueri menggunakan statistik ini untuk membantu menentukan rencana eksekusi yang paling efisien untuk kueri.

Di Aurora DSQL, Anda tidak dapat menjalankan `ANALYZE` perintah dalam transaksi eksplisit. `ANALYZE`tidak tunduk pada batas batas waktu transaksi database.

Untuk mengurangi kebutuhan intervensi manual dan menjaga statistik tetap up to date, Aurora DSQL secara otomatis berjalan `ANALYZE` sebagai proses latar belakang. Pekerjaan latar belakang ini dipicu secara otomatis berdasarkan tingkat perubahan yang diamati dalam tabel. Ini terkait dengan jumlah baris (tupel) yang telah dimasukkan, diperbarui, atau dihapus sejak analisis terakhir.

`ANALYZE`berjalan secara asinkron di latar belakang dan aktivitasnya dapat dipantau dalam tampilan sistem sys.jobs dengan kueri berikut:

```
SELECT * FROM sys.jobs WHERE job_type = 'ANALYZE';
```

Pertimbangan utama

> **Note**
>
> `ANALYZE`pekerjaan ditagih seperti pekerjaan asinkron lainnya di Aurora DSQL. Saat Anda memodifikasi tabel, ini secara tidak langsung dapat memicu pekerjaan pengumpulan statistik latar belakang otomatis, yang dapat mengakibatkan biaya pengukuran karena aktivitas tingkat sistem terkait.

`ANALYZE`Pekerjaan latar belakang, dipicu secara otomatis, mengumpulkan jenis statistik yang sama dengan manual `ANALYZE` dan menerapkannya secara default ke tabel pengguna. Tabel sistem dan katalog dikecualikan dari proses otomatis ini.

Bekerja dengan rencana Aurora DSQL EXPLY

Aurora DSQL menggunakan struktur rencana EXPLORE yang mirip dengan PostgreSQL, tetapi dengan penambahan kunci yang mencerminkan arsitektur terdistribusi dan model eksekusi.

Dalam dokumentasi ini, kami akan memberikan gambaran umum tentang rencana Aurora DSQL EXPLORE, menyoroti persamaan dan perbedaan dibandingkan dengan PostgreSQL. Kami akan membahas berbagai jenis operasi pemindaian yang tersedia di Aurora DSQL dan membantu Anda memahami biaya menjalankan kueri Anda.

PostgreSQL VS Aurora DSQL JELASKAN rencana

Aurora DSQL dibangun di atas database PostgreSQL dan berbagi sebagian besar struktur rencana dengan PostgreSQL, tetapi memiliki perbedaan arsitektur utama yang mempengaruhi eksekusi dan pengoptimalan kueri:

Fitur	PostgreSQL	Aurora DSQL
Penyimpanan Data	Penyimpanan Tumpukan	Tidak ada tumpukan, semua baris diindeks oleh pengidentifikasi unik
Kunci Utama	Indeks kunci primer terpisah dari data tabel	Indeks kunci primer adalah tabel dengan semua kolom tambahan sebagai kolom INCLUDE
Indeks Sekunder	Indeks sekunder standar	Bekerja sama dengan PostgreSQL, dengan kemampuan untuk menyertakan kolom non-kunci
Kemampuan Penyaringan	Kondisi Indeks, Filter Tumpukan	Kondisi Indeks, Filter Penyimpanan, Filter Prosesor Kueri
Jenis Pindai	Pemindaian Berurutan, Pemindaian Indeks, Pemindaian Hanya Indeks	Pemindaian Penuh, Pemindaian Hanya Indeks, Pemindaian Indeks
Eksekusi Kueri	Lokal ke Database	Didistribusikan (komputasi dan penyimpanan terpisah)

Aurora DSQL menyimpan data tabel secara langsung dalam urutan kunci primer daripada di tumpukan terpisah. Setiap baris diidentifikasi oleh kunci unik, biasanya kunci utama, yang memungkinkan database untuk mengoptimalkan pencarian lebih efisien. Perbedaan arsitektur menjelaskan mengapa Aurora DSQL sering menggunakan Index Only Scan dalam kasus di mana PostgreSQL mungkin memilih pemindaian sekuensial.

Perbedaan utama lainnya adalah bahwa Aurora DSQL memisahkan komputasi dari penyimpanan, memungkinkan filter diterapkan lebih awal di jalur eksekusi untuk mengurangi pergerakan data dan meningkatkan kinerja.

Untuk selengkapnya menggunakan rencana EXPLORE dengan PostgreSQL, lihat dokumentasi PostgreSQL EXPLORE.

Elemen kunci dalam rencana Aurora DSQL JELASKAN

Rencana Aurora DSQL EXPLOW memberikan informasi rinci tentang bagaimana kueri dijalankan, termasuk di mana pemfilteran terjadi dan kolom mana yang diambil dari penyimpanan. Memahami keluaran ini membantu Anda mengoptimalkan kinerja kueri.

Indeks Cond

Kondisi yang digunakan untuk menavigasi indeks. Pemfilteran paling efisien yang mengurangi data yang dipindai. Di Aurora DSQL, kondisi indeks dapat diterapkan pada beberapa lapisan rencana eksekusi.

Proyeksi

Kolom diambil dari penyimpanan. Proyeksi yang lebih sedikit berarti kinerja yang lebih baik.

Filter Penyimpanan

Kondisi diterapkan pada tingkat penyimpanan. Lebih efisien daripada filter prosesor kueri.

Filter Prosesor Kueri

Kondisi diterapkan pada tingkat prosesor kueri. Memerlukan transfer semua data sebelum penyaringan, yang menghasilkan pergerakan data yang lebih tinggi dan overhead pemrosesan.

Filter dalam Aurora DSQL

Aurora DSQL memisahkan komputasi dari penyimpanan, yang berarti bahwa titik di mana filter diterapkan selama eksekusi kueri memiliki dampak signifikan pada kinerja. Filter yang diterapkan

sebelum volume besar data ditransfer mengurangi latensi dan meningkatkan efisiensi. Semakin awal filter diterapkan, semakin sedikit data yang perlu diproses, dipindahkan, dan dipindai, menghasilkan kueri yang lebih cepat.

Aurora DSQL dapat menerapkan filter pada beberapa tahap di jalur kueri. Memahami tahapan ini adalah kunci untuk menafsirkan rencana kueri dan mengoptimalkan kinerja.

Tingkat	Jenis Filter	Deskripsi
1	Kondisi Indeks	Diterapkan saat memindai indeks. Membatasi berapa banyak data yang dibaca dari penyimpan an dan mengurangi data yang dikirim ke lapisan komputasi.
2	Filter Penyimpanan	Diterapkan setelah data dibaca dari penyimpanan tetapi sebelum dikirim ke komputasi. Contoh di sini adalah filter pada kolom include indeks. Mengurangi transfer data tetapi bukan jumlah yang dibaca.
3	Filter Prosesor Kueri	Diterapkan setelah data mencapai lapisan komputasi. Semua data harus ditransfer terlebih dahulu, yang meningkatkan latensi dan biaya. Saat ini, Aurora DSQL tidak dapat melakukan semua operasi penyaringan dan proyeksi pada penyimpan an, sehingga beberapa kueri mungkin terpaksa kembali ke jenis penyaringan ini.

Membaca Aurora DSQL JELASKAN rencana

Memahami cara membaca rencana EXPLORE adalah kunci untuk mengoptimalkan kinerja kueri. Di bagian ini, kita akan menelusuri contoh nyata dari rencana kueri Aurora DSQL, menunjukkan bagaimana berbagai jenis pemindaian berperilaku, menjelaskan di mana filter diterapkan, dan menyoroti peluang untuk pengoptimalan.

Contoh tabel yang digunakan dalam contoh-contoh ini

Contoh di bawah ini merujuk dua tabel: `transaction` dan`account`.

`transaction`Tabel tidak memiliki kunci utama, yang menyebabkan Aurora DSQL melakukan pemindaian tabel penuh saat menanyakannya.

`account`Tabel memiliki indeks pada`customer_id`. Indeks ini mencakup `balance` dan `status` sebagai kolom penutup, yang memungkinkan kueri tertentu dipenuhi langsung dari indeks tanpa membaca dari tabel dasar. Namun, indeks tidak termasuk`created_at`, jadi kueri yang merujuk kolom ini memerlukan akses tabel tambahan.

```
CREATE TABLE transaction (
    account_id uuid,
    transaction_date timestamp,
    description text
);

CREATE TABLE account (
    customer_id uuid,
    balance numeric,
    status varchar,
    created_at timestamp
);

CREATE INDEX ASYNC idx1 ON account (customer_id) INCLUDE (balance, status);
```

Contoh Pemindaian Lengkap

Aurora DSQL memiliki Pemindaian Berurutan, yang secara fungsional identik dengan PostgreSQL, serta Pemindaian Penuh. Satu-satunya perbedaan antara keduanya adalah bahwa Full Scan dapat memanfaatkan penyaringan ekstra pada penyimpanan. Karena ini, hampir selalu dipilih di atas Pemindaian Berurutan. Karena kesamaannya, kami hanya akan membahas contoh Pemindaian Penuh yang lebih menarik.

Pemindaian Penuh sebagian besar akan digunakan pada tabel tanpa kunci utama. Karena kunci primer Aurora DSQL secara default mencakup indeks penuh, Aurora DSQL kemungkinan besar akan menggunakan Index Only Scan pada kunci utama dalam banyak situasi di mana PostgreSQL akan menggunakan Sequential Scan. Seperti kebanyakan database lainnya, tabel tanpa indeks di atasnya akan berskala buruk.

```
EXPLAIN SELECT account_id FROM transaction WHERE transaction_date > '2025-01-01' AND
 description LIKE '%external%';
```

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Full Scan (btree-table) on transaction  (cost=125100.05..177933.38 rows=33333
 width=16)
   Filter: (description ~~ '%external%'::text)
   -> Storage Scan on transaction (cost=12510.05..17793.38 rows=66666 width=16)
        Projections: account_id, description
        Filters: (transaction_date > '2025-01-01 00:00:00'::timestamp without time
 zone)
        -> B-Tree Scan on transaction (cost=12510.05..17793.38 rows=100000 width=30)
```

Rencana ini menunjukkan dua filter yang diterapkan pada tahap yang berbeda. `transaction_date > '2025-01-01'`Kondisi ini diterapkan pada lapisan penyimpanan, mengurangi berapa banyak data yang dikembalikan. `description LIKE '%external%'`Kondisi ini diterapkan kemudian di prosesor kueri, setelah data ditransfer, membuatnya kurang efisien. Mendorong filter yang lebih selektif ke dalam lapisan penyimpanan atau indeks umumnya meningkatkan kinerja.

Contoh Index Only Scan

Index Only Scan adalah jenis pemindaian yang paling optimal di Aurora DSQL karena menghasilkan perjalanan pulang pergi paling sedikit ke lapisan penyimpanan dan dapat melakukan pemfilteran paling banyak. Tetapi hanya karena Anda melihat Index Only Scan tidak berarti Anda memiliki rencana terbaik. Karena semua tingkat penyaringan yang berbeda yang dapat terjadi, penting untuk tetap memperhatikan berbagai tempat penyaringan dapat terjadi.

```
EXPLAIN SELECT balance FROM account
WHERE customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'
AND balance > 100
AND status = 'pending';
```

```
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Only Scan using idx1 on account  (cost=725.05..1025.08 rows=8 width=18)
   Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   Filter: (balance > '100'::numeric)
   -> Storage Scan on idx1 (cost=12510.05..17793.38 rows=9 width=16)
        Projections: balance
        Filters: ((status)::text = 'pending'::text)
        -> B-Tree Scan on idx1 (cost=12510.05..17793.38 rows=10 width=30)
            Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
```

Dalam rencana ini, kondisi indeks,`customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'`), dievaluasi terlebih dahulu selama pemindaian indeks, yang merupakan tahap paling efisien karena membatasi berapa banyak data yang dibaca dari penyimpanan. Filter penyimpanan`status = 'pending'`,, diterapkan setelah data dibaca tetapi sebelum dikirim ke lapisan komputasi, mengurangi jumlah data yang ditransfer. Akhirnya, filter prosesor kueri`balance > 100`,, berjalan terakhir, setelah data dipindahkan, membuatnya paling tidak efisien. Dari jumlah tersebut, kondisi indeks memberikan kinerja terbesar karena secara langsung mengontrol berapa banyak data yang dipindai.

Contoh Pemindaian Indeks

Index Scan mirip dengan Index Only Scan, kecuali mereka memiliki langkah ekstra untuk memanggil ke tabel dasar. Karena Aurora DSQL dapat menentukan filter penyimpanan, ia dapat melakukannya baik pada panggilan indeks maupun panggilan pencarian.

Untuk memperjelas hal ini, Aurora DSQL menyajikan rencana sebagai dua node. Dengan cara ini, Anda dapat dengan jelas melihat seberapa banyak menambahkan kolom include akan membantu dalam hal baris yang dikembalikan dari penyimpanan.

```
EXPLAIN SELECT balance FROM account
WHERE customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'
AND balance > 100
AND status = 'pending'
AND created_at > '2025-01-01';
```

```
                                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Index Scan using idx1 on account  (cost=728.18..1132.20 rows=3 width=18)
   Filter: (balance > '100'::numeric)
   Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   -> Storage Scan on idx1 (cost=12510.05..17793.38 rows=8 width=16)
        Projections: balance
        Filters: ((status)::text = 'pending'::text)
        -> B-Tree Scan on account (cost=12510.05..17793.38 rows=10 width=30)
            Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   -> Storage Lookup on account (cost=12510.05..17793.38 rows=4 width=16)
        Filters: (created_at > '2025-01-01 00:00:00'::timestamp without time zone)
        -> B-Tree Lookup on transaction (cost=12510.05..17793.38 rows=8 width=30)
```

Rencana ini menunjukkan bagaimana penyaringan terjadi di beberapa tahap:

- Kondisi indeks pada data `customer_id` filter lebih awal.
- Filter penyimpanan pada `status` lebih lanjut mempersempit hasil sebelum dikirim ke komputasi.
- Filter prosesor kueri pada `balance` diterapkan kemudian, setelah transfer.
- Filter pencarian `created_at` dievaluasi saat mengambil kolom tambahan dari tabel dasar.

Menambahkan kolom yang sering digunakan sebagai `INCLUDE` bidang seringkali dapat menghilangkan pencarian ini dan meningkatkan kinerja.

Praktik Terbaik

- Sejajarkan filter dengan kolom yang diindeks untuk mendorong pemfilteran sebelumnya.
- Gunakan kolom INCLUDE untuk memungkinkan Index-Only Pemindaian dan menghindari pencarian.
- Validasi estimasi baris saat menyelidiki masalah kinerja. Aurora DSQL mengelola statistik secara otomatis dengan berjalan `ANALYZE` di latar belakang berdasarkan tingkat perubahan data. Jika perkiraan tampak tidak akurat, Anda dapat menjalankan `ANALYZE` secara manual untuk menyegarkan statistik segera.
- Hindari kueri yang tidak diindeks pada tabel besar untuk mencegah Pemindaian Penuh yang mahal.

Memahami DPU dalam EXPLY ANALYSIS

Aurora DSQL menyediakan informasi Distributed Processing Unit (DPU) tingkat pernyataan dalam output `EXPLAIN ANALYZE VERBOSE` rencana, memberi Anda visibilitas yang lebih dalam ke biaya kueri selama pengembangan. Bagian ini menjelaskan apa itu DPU dan bagaimana menafsirkannya dalam output. `EXPLAIN ANALYZE VERBOSE`

Apa itu DPU?

Distributed Processing Unit (DPU) adalah ukuran normalisasi dari pekerjaan yang dilakukan oleh Aurora DSQL. Ini terdiri dari:

- ComputedPU - Waktu yang dihabiskan untuk mengeksekusi kueri SQL
- ReadDPU — Sumber daya yang digunakan untuk membaca data dari penyimpanan
- WritedPU - Sumber daya yang digunakan untuk menulis data ke penyimpanan

- MultiRegionWriteDPU— Sumber daya yang digunakan untuk mereplikasi penulisan ke cluster peered dalam konfigurasi Multi-wilayah.

Penggunaan DPU di EXPLY ANALYZE VERBOSE

Aurora DSQL diperluas `EXPLAIN ANALYZE VERBOSE` untuk menyertakan perkiraan penggunaan DPU tingkat pernyataan hingga akhir output. Ini memberikan visibilitas langsung ke biaya kueri, membantu Anda mengidentifikasi driver biaya beban kerja, menyesuaikan kinerja kueri, dan perkiraan penggunaan sumber daya yang lebih baik.

Contoh berikut menunjukkan bagaimana menafsirkan estimasi DPU tingkat pernyataan yang termasuk dalam EXPLORE ANALYZE VERBOSE output.

Contoh 1: SELECT Query

```
EXPLAIN ANALYZE VERBOSE SELECT * FROM test_table;
```

```
QUERY PLAN
-------------------------------------------------
Index Only Scan using test_table_pkey on public.test_table  (cost=125100.05..171100.05
 rows=1000000 width=36) (actual time=2.973..4.482 rows=120 loops=1)
  Output: id, context
  -> Storage Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000 width=36)
 (actual rows=120 loops=1)
      Projections: id, context
      -> B-Tree Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000
 width=36) (actual rows=120 loops=1)
Query Identifier: qymgw1m77maoe
Planning Time: 11.415 ms
Execution Time: 4.528 ms
Statement DPU Estimate:
  Compute: 0.01607 DPU
  Read: 0.04312 DPU
  Write: 0.00000 DPU
  Total: 0.05919 DPU
```

Dalam contoh ini, pernyataan SELECT melakukan pemindaian indeks saja, sehingga sebagian besar biaya berasal dari Baca DPU (0,04312), mewakili data yang diambil dari penyimpanan dan Compute DPU (0,01607), yang mencerminkan sumber daya komputasi yang digunakan untuk memproses

dan mengembalikan hasilnya. Tidak ada Write DPU karena kueri tidak mengubah data. Total DPU (0,05919) adalah jumlah dari Compute + Read+Write.

Contoh 2: INSERT Query

```
EXPLAIN ANALYZE VERBOSE INSERT INTO test_table VALUES (1, 'name1'), (2, 'name2'), (3,
 'name3');
```

```
QUERY PLAN
-----------------------------------------------------
Insert on public.test_table  (cost=0.00..0.04 rows=0 width=0) (actual time=0.055..0.056
 rows=0 loops=1)
  ->  Values Scan on "*VALUES*"  (cost=0.00..0.04 rows=3 width=122) (actual
 time=0.003..0.008 rows=3 loops=1)
       Output: "*VALUES*".column1, "*VALUES*".column2
Query Identifier: jtkjkexhjotbo
Planning Time: 0.068 ms
Execution Time: 0.543 ms
Statement DPU Estimate:
  Compute: 0.01550 DPU
  Read: 0.00307 DPU (Transaction minimum: 0.00375)
  Write: 0.01875 DPU (Transaction minimum: 0.05000)
  Total: 0.03732 DPU
```

Pernyataan ini terutama melakukan penulisan, sehingga sebagian besar biaya dikaitkan dengan Tulis DPU. Compute DPU (0,01550) mewakili pekerjaan yang dilakukan untuk memproses dan menyisipkan nilai. DPU Baca (0,00307) mencerminkan pembacaan sistem minor (untuk pencarian katalog atau pemeriksaan indeks).

Perhatikan minimum Transaksi yang ditampilkan di samping Read and Write DPU. Ini menunjukkan biaya dasar per transaksi yang berlaku hanya ketika operasi termasuk membaca atau menulis. Bukan berarti setiap transaksi otomatis dikenakan biaya 0,00375 Read DPU atau 0,05 Write DPU. Sebaliknya, minimum ini diterapkan pada tingkat transaksi selama agregasi biaya dan hanya jika pembacaan atau penulisan terjadi dalam transaksi itu. Karena perbedaan ruang lingkup ini, perkiraan tingkat pernyataan `EXPLAIN ANALYZE VERBOSE` mungkin tidak sama persis dengan metrik tingkat transaksi yang dilaporkan dalam atau data penagihan. CloudWatch

Menggunakan Informasi DPU untuk Optimalisasi

Per-statement Perkiraan DPU memberi Anda cara yang ampuh untuk mengoptimalkan kueri di luar waktu eksekusi. Kasus penggunaan umum meliputi:

- Kesadaran Biaya: Memahami seberapa mahal kueri relatif terhadap orang lain.
- Optimasi Skema: Bandingkan dampak indeks atau perubahan skema pada kinerja dan efisiensi sumber daya.
- Perencanaan Anggaran: Perkirakan biaya beban kerja berdasarkan penggunaan DPU yang diamati.
- Perbandingan Kueri: Evaluasi pendekatan kueri alternatif dengan konsumsi DPU relatifnya.

Menafsirkan Informasi DPU

Ingatlah praktik terbaik berikut saat menggunakan data DPU dari`EXPLAIN ANALYZE VERBOSE`:

- Gunakan secara terarah: Perlakukan DPU yang dilaporkan sebagai cara untuk memahami biaya relatif kueri daripada kecocokan persis dengan CloudWatch metrik atau data penagihan. Perbedaan diharapkan karena `EXPLAIN ANALYZE VERBOSE` melaporkan biaya tingkat pernyataan, sementara CloudWatch agregat aktivitas tingkat transaksi. CloudWatch juga mencakup operasi latar belakang (seperti ANALISIS atau pemadatan) dan overhead transaksi (`BEGIN`/`COMMIT`) yang `EXPLAIN ANALYZE VERBOSE` sengaja dikecualikan.
- Variabilitas DPU di seluruh proses normal dalam sistem terdistribusi dan tidak menunjukkan kesalahan. Faktor-faktor seperti caching, perubahan rencana eksekusi, konkurensi, atau pergeseran dalam distribusi data semuanya dapat menyebabkan kueri yang sama mengkonsumsi sumber daya yang berbeda dari satu proses ke yang berikutnya.
- Operasi kecil batch: Jika beban kerja Anda mengeluarkan banyak pernyataan kecil, pertimbangkan untuk mengelompokkannya ke dalam operasi yang lebih besar (tidak melebihi 10MB). Ini mengurangi overhead pembulatan dan menghasilkan perkiraan biaya yang lebih berarti.
- Gunakan untuk penyetelan, bukan penagihan: Data DPU `EXPLAIN ANALYZE VERBOSE` dirancang untuk kesadaran biaya, penyetelan kueri, dan pengoptimalan. Ini bukan metrik tingkat penagihan. Selalu mengandalkan CloudWatch metrik atau laporan penagihan bulanan untuk data biaya dan penggunaan otoritatif.

Mengelola klaster Aurora DSQL

Aurora DSQL menyediakan beberapa opsi konfigurasi untuk membantu Anda membangun infrastruktur database yang tepat untuk kebutuhan Anda. Untuk menyiapkan infrastruktur cluster Aurora DSQL Anda, tinjau bagian berikut.

Topik

- Mengkonfigurasi kluster wilayah tunggal
- Mengkonfigurasi klaster Multi-wilayah
- Mengkonfigurasi cluster Aurora DSQL menggunakan AWS CloudFormation
- Siklus hidup cluster Aurora DSQL

Fitur dan fungsionalitas yang dibahas dalam panduan ini memastikan bahwa lingkungan Aurora DSQL Anda lebih tangguh, responsif, dan mampu mendukung aplikasi Anda saat mereka tumbuh dan berkembang.

Mengkonfigurasi kluster wilayah tunggal

Mengkonfigurasi dan mengelola cluster untuk Wilayah AWS menggunakan salah satu AWS CLI atau bahasa pemrograman pilihan Anda termasuk Python, C ++,, Java, Rust JavaScript, Ruby, .NET, dan Golang. AWS CLI Ini menyediakan akses cepat melalui perintah shell, sementara AWS Software Development Kits (SDK) memungkinkan kontrol terprogram melalui dukungan bahasa asli.

Topik

- Penggunaan AWS SDK
- Penggunaan AWS CLI

Penggunaan AWS SDK

AWS SDK menyediakan akses terprogram ke Aurora DSQL dalam bahasa pemrograman pilihan Anda. Bagian berikut menunjukkan bagaimana melakukan operasi cluster umum menggunakan bahasa pemrograman yang berbeda.

Membuat klaster

Contoh berikut menunjukkan cara membuat cluster Single-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import boto3


def create_cluster(region):
    try:
        client = boto3.client("dsql", region_name=region)
        tags = {"Name": "Python single region cluster"}
        cluster = client.create_cluster(tags=tags, deletionProtectionEnabled=True)
        print(f"Initiated creation of cluster: {cluster["identifier"]}")

        print(f"Waiting for {cluster["arn"]} to become ACTIVE")
        client.get_waiter("cluster_active").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        return cluster
    except:
        print("Unable to create cluster")
        raise


def main():
    region = "us-east-1"
    response = create_cluster(region)
    print(f"Created cluster: {response["arn"]}")


if __name__ == "__main__":
    main()
```

C++

Contoh berikut memungkinkan Anda membuat cluster dalam satu Wilayah AWS.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/CreateClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Creates a single-region cluster in Amazon Aurora DSQL
 */
CreateClusterResult CreateCluster(const Aws::String& region) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create the cluster
    CreateClusterRequest createClusterRequest;
    createClusterRequest.SetDeletionProtectionEnabled(true);
    createClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp single region cluster";
    createClusterRequest.SetTags(tags);

    auto createOutcome = client.CreateCluster(createClusterRequest);
    if (!createOutcome.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region << ": "
                  << createOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to create cluster in " + region);
    }

    auto cluster = createOutcome.GetResult();
```

```
    std::cout << "Created " << cluster.GetArn() << std::endl;

    return cluster;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region for the single-region setup
            Aws::String region = "us-east-1";

            auto cluster = CreateCluster(region);

            std::cout << "Created single region cluster:" << std::endl;
            std::cout << "Cluster ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Cluster Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import { DSQLClient, CreateClusterCommand, waitUntilClusterActive } from "@aws-sdk/
client-dsql";

async function createCluster(region) {

    const client = new DSQLClient({ region });

    try {
        const createClusterCommand = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript single region cluster"
```

```
            },
        });
        const response = await client.send(createClusterCommand);

        console.log(`Waiting for cluster ${response.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response.identifier
            }
        );
        console.log(`Cluster Id ${response.identifier} is now active`);
        return;
    } catch (error) {
        console.error(`Unable to create cluster in ${region}: `, error.message);
        throw error;
    }
}

async function main() {
    const region = "us-east-1";

    await createCluster(region);
}

main();
```

Java

Gunakan contoh berikut untuk membuat cluster dalam satu Wilayah AWS.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
```

```
import java.time.Duration;
import java.util.Map;

public class CreateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            CreateClusterRequest request = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .tags(Map.of("Name", "java single region cluster"))
                    .build();
            CreateClusterResponse cluster = client.createCluster(request);
            System.out.println("Created " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for a cluster's
            // transition to ACTIVE.
            System.out.println("Waiting for cluster to become ACTIVE");
            WaiterResponse<GetClusterResponse> waiterResponse =
 client.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster.identifier()),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            waiterResponse.matched().response().ifPresent(System.out::println);
        }
    }
}
```

Rust

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
```

```
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{Client, Config};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    let region_provider = Region::new(region);

    let config = load_defaults(BehaviorVersion::latest())
        .region(region_provider)
        .load()
        .await;

    let config = Config::new(&config);

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_cluster(region: &'static str) -> GetClusterOutput {
    let client = dsql_client(region).await;

    let tags = HashMap::from([
        (String::from("Name"), String::from("rust single region cluster")),
    ]);

    println!("Creating cluster in {region}");
    let cluster = client
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
        .send()
        .await
        .unwrap();

    println!("Created {}", cluster.arn);

    println!("Waiting for {} to become ACTIVE", cluster.arn);
    let cluster_output = client
        .wait_until_cluster_active()
        .identifier(&cluster.identifier)
        .send()
```

```
        .await
        .unwrap();

    cluster_output
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let region = "us-east-1";

    let cluster = create_cluster(region).await;

    println!("Created single region cluster:");
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
require "aws-sdk-dsql"
require "pp"

def create_cluster(region)
  client = Aws::DSQL::Client.new(region: region)

  puts "Creating cluster in #{region}"
  cluster = client.create_cluster(
    deletion_protection_enabled: true,
    tags: {
      Name: "ruby single region cluster"
    }
  )
  puts "Created #{cluster.arn}"

  puts "Waiting for #{cluster.arn} to become ACTIVE"
  cluster = client.wait_until(:cluster_active, identifier: cluster.identifier) do |
w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
```

```
  cluster
rescue Aws::Errors::ServiceError => e
  abort "Failed to create cluster: #{e.message}"
end

def main
  region = "us-east-1"

  cluster = create_cluster(region)

  puts "Created single region cluster:"
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class CreateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = new DefaultAWSCredentialsChain().GetCredentials();
            var clientConfig = new AmazonDSQLConfig
```

```
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create a cluster with deletion protection enabled and a name tag.
        /// </summary>
        public static async Task<CreateClusterResponse> Create(RegionEndpoint
 region)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp single region cluster" }
                };

                var createClusterRequest = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags
                };

                var cluster = await client.CreateClusterAsync(createClusterRequest);
                Console.WriteLine($"Created {cluster.Arn}");

                return cluster;
            }
        }

        public static async Task Main()
        {
            var region = RegionEndpoint.USEast1;

            var cluster = await Create(region);

            Console.WriteLine("Created single region cluster:");
            Console.WriteLine($"Cluster ARN: {cluster.Arn}");
        }
    }
}
```

Golang

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func CreateCluster(ctx context.Context, region string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 client := dsql.NewFromConfig(cfg)

 deleteProtect := true

 input := &dsql.CreateClusterInput{
  DeletionProtectionEnabled: &deleteProtect,
  Tags: map[string]string{
   "Name": "go single-region cluster",
  },
 }

 clusterProperties, err := client.CreateCluster(context.Background(), input)

 if err != nil {
  return fmt.Errorf("failed to create cluster. %v", err)
 }

 // Create the waiter with our custom options
 waiter := dsql.NewClusterActiveWaiter(client, func(o
 *dsql.ClusterActiveWaiterOptions) {
```

```
  o.MaxDelay = 30 * time.Second
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })

 // Create the input for the clusterProperties to monitor
 clusterInput := &dsql.GetClusterInput{
  Identifier: clusterProperties.Identifier,
 }

 fmt.Printf("Waiting for cluster %s to become ACTIVE\n", *clusterProperties.Arn)
 err = waiter.Wait(ctx, clusterInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to become active: %w", err)
 }

 fmt.Printf("Created single region cluster: %s\n", *clusterProperties.Arn)
 return nil
}

func main() {
 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := CreateCluster(ctx, "us-east-1")
 if err != nil {
  fmt.Printf("failed to create cluster: %v", err)
  panic(err)
 }

}
```

Dapatkan cluster

Contoh berikut menunjukkan cara mendapatkan informasi tentang cluster Single-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk mendapatkan informasi tentang kluster Single-region, gunakan contoh berikut.

```
import boto3
from datetime import datetime
import json


def get_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.get_cluster(identifier=identifier)
    except:
        print(f"Unable to get cluster {identifier} in region {region}")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    response = get_cluster(region, cluster_id)

    print(json.dumps(response, indent=2, default=lambda obj: obj.isoformat() if
 isinstance(obj, datetime) else None))


if __name__ == "__main__":
    main()
```

C++

Gunakan contoh berikut untuk mendapatkan informasi tentang kluster Single-region.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Retrieves information about a cluster in Amazon Aurora DSQL
```

```
 */
GetClusterResult GetCluster(const Aws::String& region, const Aws::String&
 identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Get the cluster
    GetClusterRequest getClusterRequest;
    getClusterRequest.SetIdentifier(identifier);

    auto getOutcome = client.GetCluster(getClusterRequest);
    if (!getOutcome.IsSuccess()) {
        std::cerr << "Failed to retrieve cluster " << identifier << " in " << region
 << ": "
                  << getOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to retrieve cluster " + identifier + " in
 region " + region);
    }

    return getOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            auto cluster = GetCluster(region, clusterId);

            // Print cluster details
            std::cout << "Cluster Details:" << std::endl;
            std::cout << "ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
```

```
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk mendapatkan informasi tentang kluster Single-region, gunakan contoh berikut.

```
import { DSQLClient, GetClusterCommand } from "@aws-sdk/client-dsql";

async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  const getClusterCommand = new GetClusterCommand({
    identifier: clusterId,
  });

  try {
    return await client.send(getClusterCommand);
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or deleted");
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  const response = await getCluster(region, clusterId);
  console.log("Cluster: ", response);
}

main();
```

Java

Contoh berikut memungkinkan Anda mendapatkan informasi tentang kluster Single-region.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

public class GetCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            GetClusterResponse cluster = client.getCluster(r ->
 r.identifier(clusterId));
            System.out.println(cluster);
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

Contoh berikut memungkinkan Anda mendapatkan informasi tentang kluster Single-region.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
```

```
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Get a ClusterResource from DSQL cluster identifier
pub async fn get_cluster(region: &'static str, identifier: &'static str) ->
 GetClusterOutput {
    let client = dsql_client(region).await;
    client
        .get_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = get_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Contoh berikut memungkinkan Anda mendapatkan informasi tentang kluster Single-region.

```
require "aws-sdk-dsql"
require "pp"

def get_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  client.get_cluster(identifier: identifier)
rescue Aws::Errors::ServiceError => e
  abort "Unable to retrieve cluster #{identifier} in region #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  cluster = get_cluster(region, cluster_id)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Contoh berikut memungkinkan Anda mendapatkan informasi tentang kluster Single-region.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class GetCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
```

```
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Get information about a DSQL cluster.
        /// </summary>
        public static async Task<GetClusterResponse> Get(RegionEndpoint region,
 string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var getClusterRequest = new GetClusterRequest
                {
                    Identifier = identifier
                };

                return await client.GetClusterAsync(getClusterRequest);
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            var response = await Get(region, clusterId);
            Console.WriteLine($"Cluster ARN: {response.Arn}");
        }
    }
}
```

Golang

Contoh berikut memungkinkan Anda mendapatkan informasi tentang kluster Single-region.

```
package main
```

```
import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func GetCluster(ctx context.Context, region, identifier string) (clusterStatus
 *dsql.GetClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := &dsql.GetClusterInput{
  Identifier: aws.String(identifier),
 }
 clusterStatus, err = client.GetCluster(context.Background(), input)

 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }

 log.Printf("Cluster ARN: %s", *clusterStatus.Arn)

 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 _, err := GetCluster(ctx, region, identifier)
```

```
 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }
}
```

Memperbarui klaster

Contoh berikut menunjukkan cara memperbarui cluster Single-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk memperbarui kluster Single-region, gunakan contoh berikut.

```
import boto3


def update_cluster(region, cluster_id, deletion_protection_enabled):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.update_cluster(identifier=cluster_id,
 deletionProtectionEnabled=deletion_protection_enabled)
    except:
        print("Unable to update cluster")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    deletion_protection_enabled = False
    response = update_cluster(region, cluster_id, deletion_protection_enabled)
    print(f"Updated {response["arn"]} with deletion_protection_enabled:
 {deletion_protection_enabled}")


if __name__ == "__main__":
    main()
```

C++

Gunakan contoh berikut untuk memperbarui kluster Single-region.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Updates a cluster in Amazon Aurora DSQL
 */
UpdateClusterResult UpdateCluster(const Aws::String& region, const
 Aws::Map<Aws::String, Aws::String>& updateParams) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create update request
    UpdateClusterRequest updateRequest;
    updateRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Set identifier (required)
    if (updateParams.find("identifier") != updateParams.end()) {
        updateRequest.SetIdentifier(updateParams.at("identifier"));
    } else {
        throw std::runtime_error("Cluster identifier is required for update
 operation");
    }

    // Set deletion protection if specified
    if (updateParams.find("deletion_protection_enabled") != updateParams.end()) {
        bool deletionProtection = (updateParams.at("deletion_protection_enabled") ==
 "true");
        updateRequest.SetDeletionProtectionEnabled(deletionProtection);
    }

    // Execute the update
```

```
    auto updateOutcome = client.UpdateCluster(updateRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster: " <<
 updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to update cluster");
    }

    return updateOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and update parameters
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            // Create parameter map
            Aws::Map<Aws::String, Aws::String> updateParams;
            updateParams["identifier"] = clusterId;
            updateParams["deletion_protection_enabled"] = "false";

            auto updatedCluster = UpdateCluster(region, updateParams);

            std::cout << "Updated " << updatedCluster.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk memperbarui kluster Single-region, gunakan contoh berikut.

```
import { DSQLClient, UpdateClusterCommand } from "@aws-sdk/client-dsql";

export async function updateCluster(region, clusterId, deletionProtectionEnabled) {
```

```
  const client = new DSQLClient({ region });

  const updateClusterCommand = new UpdateClusterCommand({
    identifier: clusterId,
    deletionProtectionEnabled: deletionProtectionEnabled
  });

  try {
    return await client.send(updateClusterCommand);
  } catch (error) {
    console.error("Unable to update cluster", error.message);
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";
  const deletionProtectionEnabled = false;

  const response = await updateCluster(region, clusterId,
 deletionProtectionEnabled);
  console.log(`Updated ${response.arn}`);
}

main();
```

Java

Gunakan contoh berikut untuk memperbarui kluster Single-region.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;
import software.amazon.awssdk.services.dsql.model.UpdateClusterResponse;

public class UpdateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
```

```
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            UpdateClusterRequest request = UpdateClusterRequest.builder()
                    .identifier(clusterId)
                    .deletionProtectionEnabled(false)
                    .build();
            UpdateClusterResponse cluster = client.updateCluster(request);
            System.out.println("Updated " + cluster.arn());
        }
    }
}
```

Rust

Gunakan contoh berikut untuk memperbarui kluster Single-region.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::update_cluster::UpdateClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();
```

```
    Client::from_conf(config)
}

/// Update a DSQL cluster and set delete protection to false. Also add new tags.
pub async fn update_cluster(region: &'static str, identifier: &'static str) ->
 UpdateClusterOutput {
    let client = dsql_client(region).await;
    // Update delete protection
    let update_response = client
        .update_cluster()
        .identifier(identifier)
        .deletion_protection_enabled(false)
        .send()
        .await
        .unwrap();

    update_response
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = update_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Gunakan contoh berikut untuk memperbarui kluster Single-region.

```
require "aws-sdk-dsql"

def update_cluster(region, update_params)
  client = Aws::DSQL::Client.new(region: region)
  client.update_cluster(update_params)
rescue Aws::Errors::ServiceError => e
  abort "Unable to update cluster: #{e.message}"
end

def main
```

```
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  updated_cluster = update_cluster(region, {
    identifier: cluster_id,
    deletion_protection_enabled: false
  })
  puts "Updated #{updated_cluster.arn}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Gunakan contoh berikut untuk memperbarui kluster Single-region.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class UpdateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
```

```
        /// Update a DSQL cluster and set delete protection to false.
        /// </summary>
        public static async Task<UpdateClusterResponse> Update(RegionEndpoint
 region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = identifier,
                    DeletionProtectionEnabled = false
                };

                UpdateClusterResponse response = await
 client.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Updated {response.Arn}");

                return response;
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            await Update(region, clusterId);
        }
    }
}
```

Golang

Gunakan contoh berikut untuk memperbarui kluster Single-region.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
```

```
)

func UpdateCluster(ctx context.Context, region, id string, deleteProtection bool)
 (clusterStatus *dsql.UpdateClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := dsql.UpdateClusterInput{
  Identifier:                &id,
  DeletionProtectionEnabled: &deleteProtection,
 }

 clusterStatus, err = client.UpdateCluster(context.Background(), &input)

 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }

 log.Printf("Cluster updated successfully: %v", clusterStatus.Status)
 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"
 deleteProtection := false

 _, err := UpdateCluster(ctx, region, identifier, deleteProtection)
 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }
}
```

Hapus cluster

Contoh berikut menunjukkan cara menghapus cluster Single-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import boto3


def delete_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        cluster = client.delete_cluster(identifier=identifier)
        print(f"Initiated delete of {cluster["arn"]}")

        print("Waiting for cluster to finish deletion")
        client.get_waiter("cluster_not_exists").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )
    except:
        print("Unable to delete cluster " + identifier)
        raise


def main():
    region = "us-east-1"
    cluster_id = "<cluster id>"  # Use a placeholder in docs
    delete_cluster(region, cluster_id)
    print(f"Deleted {cluster_id}")


if __name__ == "__main__":
    main()
```

C++

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/DeleteClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Deletes a single-region cluster in Amazon Aurora DSQL
 */
void DeleteCluster(const Aws::String& region, const Aws::String& identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Delete the cluster
    DeleteClusterRequest deleteRequest;
    deleteRequest.SetIdentifier(identifier);
    deleteRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome = client.DeleteCluster(deleteRequest);
    if (!deleteOutcome.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << identifier << " in " << region
 << ": "
                  << deleteOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to delete cluster " + identifier + " in " +
 region);
    }

    auto cluster = deleteOutcome.GetResult();
    std::cout << "Initiated delete of " << cluster.GetArn() << std::endl;
}
```

```
int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            DeleteCluster(region, clusterId);

            std::cout << "Deleted " << clusterId << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import { DSQLClient, DeleteClusterCommand, waitUntilClusterNotExists } from "@aws-
sdk/client-dsql";

async function deleteCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  try {
    const deleteClusterCommand = new DeleteClusterCommand({
      identifier: clusterId,
    });
    const response = await client.send(deleteClusterCommand);

    console.log(`Waiting for cluster ${response.identifier} to finish deletion`);

    await waitUntilClusterNotExists(
      {
        client: client,
        maxWaitTime: 300 // Wait for 5 minutes
```

```
      },
      {
        identifier: response.identifier
      }
    );
    console.log(`Cluster Id ${response.identifier} is now deleted`);
    return;
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or already deleted");
    } else {
      console.error("Unable to delete cluster: ", error.message);
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  await deleteCluster(region, clusterId);
}

main();
```

Java

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

import java.time.Duration;

public class DeleteCluster {

    public static void main(String[] args) {
```

```
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";


        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            DeleteClusterResponse cluster = client.deleteCluster(r ->
 r.identifier(clusterId));
            System.out.println("Initiated delete of " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for deletion.
            System.out.println("Waiting for cluster to finish deletion");
            client.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            System.out.println("Deleted " + cluster.arn());
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
```

```
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Delete a DSQL cluster
pub async fn delete_cluster(region: &'static str, identifier: &'static str) {
    let client = dsql_client(region).await;
    let delete_response = client
        .delete_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap();
    println!("Initiated delete of {}", delete_response.arn);

    println!("Waiting for cluster to finish deletion");
    client
        .wait_until_cluster_not_exists()
        .identifier(identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";
    let cluster_id = "<cluster to be deleted>";

    delete_cluster(region, cluster_id).await;
    println!("Deleted {cluster_id}");
```

```
    Ok(())
}
```

Ruby

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
require "aws-sdk-dsql"

def delete_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  cluster = client.delete_cluster(identifier: identifier)
  puts "Initiated delete of #{cluster.arn}"

  # The DSQL SDK offers built-in waiters to poll for deletion.
  puts "Waiting for cluster to finish deletion"
  client.wait_until(:cluster_not_exists, identifier: cluster.identifier) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Unable to delete cluster #{identifier} in #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  delete_cluster(region, cluster_id)
  puts "Deleted #{cluster_id}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
```

```
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class DeleteSingleRegionCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Delete a DSQL cluster.
        /// </summary>
        public static async Task Delete(RegionEndpoint region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var deleteRequest = new DeleteClusterRequest
                {
                    Identifier = identifier
                };

                var deleteResponse = await client.DeleteClusterAsync(deleteRequest);
                Console.WriteLine($"Initiated deletion of {deleteResponse.Arn}");
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<cluster to be deleted>";
```

```
            await Delete(region, clusterId);
        }
    }
}
```

Golang

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func DeleteSingleRegion(ctx context.Context, identifier, region string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 // Create delete cluster input
 deleteInput := &dsql.DeleteClusterInput{
  Identifier: &identifier,
 }

 // Delete the cluster
 result, err := client.DeleteCluster(ctx, deleteInput)
 if err != nil {
  return fmt.Errorf("failed to delete cluster: %w", err)
 }

 fmt.Printf("Initiated deletion of cluster: %s\n", *result.Arn)
```

```
 // Create waiter to check cluster deletion
 waiter := dsql.NewClusterNotExistsWaiter(client, func(options
 *dsql.ClusterNotExistsWaiterOptions) {
  options.MinDelay = 10 * time.Second
  options.MaxDelay = 30 * time.Second
  options.LogWaitAttempts = true
 })

 // Create the input for checking cluster status
 getInput := &dsql.GetClusterInput{
  Identifier: &identifier,
 }

 // Wait for the cluster to be deleted
 fmt.Printf("Waiting for cluster %s to be deleted...\n", identifier)
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to be deleted: %w", err)
 }

 fmt.Printf("Cluster %s has been successfully deleted\n", identifier)
 return nil
}

func DeleteCluster(ctx context.Context) {
}

// Example usage in main function
func main() {
 // Your existing setup code for client configuration...

 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 // Need to make sure that cluster does not have delete protection enabled
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 err := DeleteSingleRegion(ctx, identifier, region)
 if err != nil {
  log.Fatalf("Failed to delete cluster: %v", err)
 }
```

```
}
```

Untuk contoh dan contoh kode lainnya, kunjungi repositori Aurora DSQL Sampel. GitHub

Penggunaan AWS CLI

AWS CLI menyediakan antarmuka baris perintah untuk mengelola cluster Aurora DSQL Anda. Contoh berikut menunjukkan operasi manajemen cluster umum.

Membuat klaster

Buat cluster menggunakan create-cluster perintah.

> **Note**
>
> Pembuatan cluster adalah operasi asinkron. Panggil GetCluster API hingga statusnya berubah menjadi`ACTIVE`. Anda dapat terhubung ke cluster Anda setelah menjadi aktif.

Example Perintah

```
aws dsql create-cluster --region us-east-1
```

> **Note**
>
> Untuk menonaktifkan perlindungan penghapusan selama pembuatan, sertakan bendera. `--no-deletion-protection-enabled`

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "CREATING",
    "creationTime": "2024-05-25T16:56:49.784000-07:00",
    "deletionProtectionEnabled": true,
    "tag": {},
```

```
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

Menggambarkan sebuah cluster

Dapatkan informasi tentang cluster menggunakan get-cluster perintah.

Example Perintah

```
aws dsql get-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2024-11-27T00:32:14.434000-08:00",
    "deletionProtectionEnabled": false,
    "encryptionDetails": {
        "encryptionType": "CUSTOMER_MANAGED_KMS_KEY",
        "kmsKeyArn": "arn:aws:kms:us-east-1:111122223333:key/123a456b-c789-01de-2f34-
g5hi6j7k8lm9",
        "encryptionStatus": "ENABLED"
    }
}
```

Memperbarui klaster

Perbarui cluster yang ada menggunakan update-cluster perintah.

> **Note**
>
> Pembaruan adalah operasi asinkron. Panggil GetCluster API hingga status berubah `ACTIVE` untuk melihat perubahan Anda.

Example Perintah

```
aws dsql update-cluster \
  --region us-east-1 \
  --no-deletion-protection-enabled \
  --identifier your_cluster_id
```

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

Menghapus klaster

Hapus cluster yang ada menggunakan delete-cluster perintah.

> **Note**
>
> Anda hanya dapat menghapus klaster yang memiliki perlindungan penghapusan dinonaktifkan. Secara default, perlindungan penghapusan diaktifkan saat Anda membuat cluster baru.

Example Perintah

```
aws dsql delete-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "DELETING",
```

```
    "creationTime": "2024-05-24T09:16:43.778000-07:00"
}
```

Daftar cluster

Buat daftar cluster Anda menggunakan list-clusters perintah.

Example Perintah

```
aws dsql list-clusters --region us-east-1
```

Example Respons

```
{
    "clusters": [
        {
            "identifier": "abc0def1baz2quux3quux4quuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuux"
        },
        {
            "identifier": "abc0def1baz2quux3quux5quuuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux5quuuux"
        }
    ]
}
```

Mengkonfigurasi klaster Multi-wilayah

Konfigurasikan dan kelola cluster di beberapa Wilayah AWS menggunakan bahasa pemrograman AWS CLI atau pilihan Anda termasuk Python, C ++,, Java, Rust JavaScript, Ruby, .NET, dan Golang. AWS CLI Ini menyediakan akses cepat melalui perintah shell, sementara AWS SDK mengaktifkan kontrol terprogram melalui dukungan bahasa asli.

Topik

- Penggunaan AWS SDK
- Penggunaan AWS CLI

Penggunaan AWS SDK

AWS SDK menyediakan akses terprogram ke Aurora DSQL dalam bahasa pemrograman pilihan Anda. Bagian berikut menunjukkan bagaimana melakukan operasi cluster umum menggunakan bahasa pemrograman yang berbeda.

Membuat klaster

Contoh berikut menunjukkan cara membuat cluster Multi-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
import boto3


def create_multi_region_clusters(region_1, region_2, witness_region):
    try:
        client_1 = boto3.client("dsql", region_name=region_1)
        client_2 = boto3.client("dsql", region_name=region_2)

        # We can only set the witness region for the first cluster
        cluster_1 = client_1.create_cluster(
            deletionProtectionEnabled=True,
            multiRegionProperties={"witnessRegion": witness_region},
            tags={"Name": "Python multi region cluster"}
        )
        print(f"Created {cluster_1["arn"]}")

        # For the second cluster we can set witness region and designate cluster_1
 as a peer
        cluster_2 = client_2.create_cluster(
            deletionProtectionEnabled=True,
            multiRegionProperties={"witnessRegion": witness_region, "clusters":
 [cluster_1["arn"]]},
            tags={"Name": "Python multi region cluster"}
        )

        print(f"Created {cluster_2["arn"]}")
        # Now that we know the cluster_2 arn we can set it as a peer of cluster_1
```

```
        client_1.update_cluster(
            identifier=cluster_1["identifier"],
            multiRegionProperties={"witnessRegion": witness_region, "clusters":
 [cluster_2["arn"]]}
        )
        print(f"Added {cluster_2["arn"]} as a peer of {cluster_1["arn"]}")

        # Now that multiRegionProperties is fully defined for both clusters
        # they'll begin the transition to ACTIVE
        print(f"Waiting for {cluster_1["arn"]} to become ACTIVE")
        client_1.get_waiter("cluster_active").wait(
            identifier=cluster_1["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        print(f"Waiting for {cluster_2["arn"]} to become ACTIVE")
        client_2.get_waiter("cluster_active").wait(
            identifier=cluster_2["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        return (cluster_1, cluster_2)

    except:
        print("Unable to create cluster")
        raise


def main():
    region_1 = "us-east-1"
    region_2 = "us-east-2"
    witness_region = "us-west-2"
    (cluster_1, cluster_2) = create_multi_region_clusters(region_1, region_2,
 witness_region)
    print("Created multi region clusters:")
    print("Cluster id: " + cluster_1['arn'])
    print("Cluster id: " + cluster_2['arn'])
```

```
if __name__ == "__main__":
    main()
```

C++

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/CreateClusterRequest.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <aws/dsql/model/MultiRegionProperties.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Creates multi-region clusters in Amazon Aurora DSQL
 */
std::pair<CreateClusterResult, CreateClusterResult> CreateMultiRegionClusters(
    const Aws::String& region1,
    const Aws::String& region2,
    const Aws::String& witnessRegion) {

    // Create clients for each region
    DSQL::DSQLClientConfiguration clientConfig1;
    clientConfig1.region = region1;
    DSQL::DSQLClient client1(clientConfig1);

    DSQL::DSQLClientConfiguration clientConfig2;
    clientConfig2.region = region2;
    DSQL::DSQLClient client2(clientConfig2);

    std::cout << "Creating cluster in " << region1 << std::endl;
```

```
    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region
    MultiRegionProperties multiRegionProps1;
    multiRegionProps1.SetWitnessRegion(witnessRegion);
    createClusterRequest1.SetMultiRegionProperties(multiRegionProps1);

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp multi region cluster 1";
    createClusterRequest1.SetTags(tags);
    createClusterRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto createOutcome1 = client1.CreateCluster(createClusterRequest1);
    if (!createOutcome1.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region1 << ": "
                  << createOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }

    auto cluster1 = createOutcome1.GetResult();
    std::cout << "Created " << cluster1.GetArn() << std::endl;

    // Create second cluster
    std::cout << "Creating cluster in " << region2 << std::endl;

    CreateClusterRequest createClusterRequest2;
    createClusterRequest2.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region and cluster1 as peer
    MultiRegionProperties multiRegionProps2;
    multiRegionProps2.SetWitnessRegion(witnessRegion);

    Aws::Vector<Aws::String> clusters;
    clusters.push_back(cluster1.GetArn());
    multiRegionProps2.SetClusters(clusters);

    tags["Name"] = "cpp multi region cluster 2";
    createClusterRequest2.SetMultiRegionProperties(multiRegionProps2);
    createClusterRequest2.SetTags(tags);
    createClusterRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());
```

```
    auto createOutcome2 = client2.CreateCluster(createClusterRequest2);
    if (!createOutcome2.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region2 << ": "
                  << createOutcome2.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }

    auto cluster2 = createOutcome2.GetResult();
    std::cout << "Created " << cluster2.GetArn() << std::endl;

    // Now that we know the cluster2 arn we can set it as a peer of cluster1
    UpdateClusterRequest updateClusterRequest;
    updateClusterRequest.SetIdentifier(cluster1.GetIdentifier());

    MultiRegionProperties updatedProps;
    updatedProps.SetWitnessRegion(witnessRegion);

    Aws::Vector<Aws::String> updatedClusters;
    updatedClusters.push_back(cluster2.GetArn());
    updatedProps.SetClusters(updatedClusters);

    updateClusterRequest.SetMultiRegionProperties(updatedProps);
    updateClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto updateOutcome = client1.UpdateCluster(updateClusterRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster in " << region1 << ": "
                  << updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to update multi-region clusters");
    }

    std::cout << "Added " << cluster2.GetArn() << " as a peer of " <<
 cluster1.GetArn() << std::endl;

    return std::make_pair(cluster1, cluster2);
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define regions for the multi-region setup
            Aws::String region1 = "us-east-1";
```

```
            Aws::String region2 = "us-east-2";
            Aws::String witnessRegion = "us-west-2";

            auto [cluster1, cluster2] = CreateMultiRegionClusters(region1, region2,
 witnessRegion);

            std::cout << "Created multi region clusters:" << std::endl;
            std::cout << "Cluster 1 ARN: " << cluster1.GetArn() << std::endl;
            std::cout << "Cluster 2 ARN: " << cluster2.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
import { DSQLClient, CreateClusterCommand, UpdateClusterCommand,
 waitUntilClusterActive } from "@aws-sdk/client-dsql";

async function createMultiRegionCluster(region1, region2, witnessRegion) {

    const client1 = new DSQLClient({ region: region1 });
    const client2 = new DSQLClient({ region: region2 });

    try {
        // We can only set the witness region for the first cluster
        console.log(`Creating cluster in ${region1}`);
        const createClusterCommand1 = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript multi region cluster 1"
            },
            multiRegionProperties: {
                witnessRegion: witnessRegion
            }
        });
```

```
        const response1 = await client1.send(createClusterCommand1);
        console.log(`Created ${response1.arn}`);

        // For the second cluster we can set witness region and designate the first
cluster as a peer
        console.log(`Creating cluster in ${region2}`);
        const createClusterCommand2 = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript multi region cluster 2"
            },
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response1.arn]
            }
        });
        const response2 = await client2.send(createClusterCommand2);
        console.log(`Created ${response2.arn}`);

        // Now that we know the second cluster arn we can set it as a peer of the
first cluster
        const updateClusterCommand = new UpdateClusterCommand({
            identifier: response1.identifier,
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response2.arn]
            }
        });
        await client1.send(updateClusterCommand);
        console.log(`Added ${response2.arn} as a peer of ${response1.arn}`);

        // Now that multiRegionProperties is fully defined for both clusters they'll
begin the transition to ACTIVE
        console.log(`Waiting for cluster ${response1.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster 1 is now active`);
```

```
        console.log(`Waiting for cluster ${response2.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
        );
        console.log(`Cluster 2 is now active`);
        console.log("The multi region clusters are now active");
        return;
    } catch (error) {
        console.error("Failed to create cluster: ", error.message);
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const region2 = "us-east-2";
    const witnessRegion = "us-west-2";

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.DsqlClientBuilder;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
```

```
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;

import java.time.Duration;
import java.util.Map;

public class CreateMultiRegionCluster {

    public static void main(String[] args) {
        Region region1 = Region.US_EAST_1;
        Region region2 = Region.US_EAST_2;
        Region witnessRegion = Region.US_WEST_2;

        DsqlClientBuilder clientBuilder = DsqlClient.builder()
                .credentialsProvider(DefaultCredentialsProvider.create());

        try (
            DsqlClient client1 = clientBuilder.region(region1).build();
            DsqlClient client2 = clientBuilder.region(region2).build()
        ) {
            // We can only set the witness region for the first cluster
            System.out.println("Creating cluster in " + region1);
            CreateClusterRequest request1 = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .multiRegionProperties(mrp ->
 mrp.witnessRegion(witnessRegion.toString()))
                    .tags(Map.of("Name", "java multi region cluster"))
                    .build();
            CreateClusterResponse cluster1 = client1.createCluster(request1);
            System.out.println("Created " + cluster1.arn());

            // For the second cluster we can set the witness region and designate
            // cluster1 as a peer.
            System.out.println("Creating cluster in " + region2);
            CreateClusterRequest request2 = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .multiRegionProperties(mrp ->

 mrp.witnessRegion(witnessRegion.toString()).clusters(cluster1.arn())
                    )
                    .tags(Map.of("Name", "java multi region cluster"))
                    .build();
            CreateClusterResponse cluster2 = client2.createCluster(request2);
```

```
            System.out.println("Created " + cluster2.arn());

            // Now that we know the cluster2 ARN we can set it as a peer of cluster1
            UpdateClusterRequest updateReq = UpdateClusterRequest.builder()
                    .identifier(cluster1.identifier())
                    .multiRegionProperties(mrp ->

mrp.witnessRegion(witnessRegion.toString()).clusters(cluster2.arn())
                    )
                    .build();
            client1.updateCluster(updateReq);
            System.out.printf("Added %s as a peer of %s%n", cluster2.arn(),
cluster1.arn());

            // Now that MultiRegionProperties is fully defined for both clusters
they'll begin
            // the transition to ACTIVE.
            System.out.printf("Waiting for cluster %s to become ACTIVE%n",
cluster1.arn());
            GetClusterResponse activeCluster1 =
client1.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster1.identifier()),
                    config -> config.backoffStrategyV2(

BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            ).matched().response().orElseThrow();

            System.out.printf("Waiting for cluster %s to become ACTIVE%n",
cluster2.arn());
            GetClusterResponse activeCluster2 =
client2.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster2.identifier()),
                    config -> config.backoffStrategyV2(

BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            ).matched().response().orElseThrow();

            System.out.println("Created multi region clusters:");
            System.out.println(activeCluster1);
            System.out.println(activeCluster2);
        }
    }
```

```
}
```

Rust

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::types::MultiRegionProperties;
use aws_sdk_dsql::{Client, Config};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_multi_region_clusters(
    region_1: &'static str,
    region_2: &'static str,
    witness_region: &'static str,
) -> (GetClusterOutput, GetClusterOutput) {
    let client_1 = dsql_client(region_1).await;
    let client_2 = dsql_client(region_2).await;
```

```
    let tags = HashMap::from([(
        String::from("Name"),
        String::from("rust multi region cluster"),
    )]);

    // We can only set the witness region for the first cluster
    println!("Creating cluster in {region_1}");
    let cluster_1 = client_1
        .create_cluster()
        .set_tags(Some(tags.clone()))
        .deletion_protection_enabled(true)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .build(),
        )
        .send()
        .await
        .unwrap();
    let cluster_1_arn = &cluster_1.arn;
    println!("Created {cluster_1_arn}");

    // For the second cluster we can set witness region and designate cluster_1 as a
peer
    println!("Creating cluster in {region_2}");
    let cluster_2 = client_2
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .clusters(&cluster_1.arn)
                .build(),
        )
        .send()
        .await
        .unwrap();
    let cluster_2_arn = &cluster_2.arn;
    println!("Created {cluster_2_arn}");

    // Now that we know the cluster_2 arn we can set it as a peer of cluster_1
    client_1
        .update_cluster()
```

```
        .identifier(&cluster_1.identifier)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .clusters(&cluster_2.arn)
                .build(),
        )
        .send()
        .await
        .unwrap();
    println!("Added {cluster_2_arn} as a peer of {cluster_1_arn}");

    // Now that the multi-region properties are fully defined for both clusters
    // they'll begin the transition to ACTIVE
    println!("Waiting for {cluster_1_arn} to become ACTIVE");
    let cluster_1_output = client_1
        .wait_until_cluster_active()
        .identifier(&cluster_1.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap();

    println!("Waiting for {cluster_2_arn} to become ACTIVE");
    let cluster_2_output = client_2
        .wait_until_cluster_active()
        .identifier(&cluster_2.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap();

    (cluster_1_output, cluster_2_output)
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region_1 = "us-east-1";
    let region_2 = "us-east-2";
    let witness_region = "us-west-2";

    let (cluster_1, cluster_2) =
```

```
        create_multi_region_clusters(region_1, region_2, witness_region).await;

    println!("Created multi region clusters:");
    println!("{:#?}", cluster_1);
    println!("{:#?}", cluster_2);

    Ok(())
}
```

Ruby

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
require "aws-sdk-dsql"
require "pp"

def create_multi_region_clusters(region_1, region_2, witness_region)
  client_1 = Aws::DSQL::Client.new(region: region_1)
  client_2 = Aws::DSQL::Client.new(region: region_2)

  # We can only set the witness region for the first cluster
  puts "Creating cluster in #{region_1}"
  cluster_1 = client_1.create_cluster(
    deletion_protection_enabled: true,
    multi_region_properties: {
      witness_region: witness_region
    },
    tags: {
      Name: "ruby multi region cluster"
    }
  )
  puts "Created #{cluster_1.arn}"

  # For the second cluster we can set witness region and designate cluster_1 as a
 peer
  puts "Creating cluster in #{region_2}"
  cluster_2 = client_2.create_cluster(
    deletion_protection_enabled: true,
    multi_region_properties: {
      witness_region: witness_region,
      clusters: [ cluster_1.arn ]
```

```
    },
    tags: {
      Name: "ruby multi region cluster"
    }
  )
  puts "Created #{cluster_2.arn}"

  # Now that we know the cluster_2 arn we can set it as a peer of cluster_1
  client_1.update_cluster(
    identifier: cluster_1.identifier,
    multi_region_properties: {
      witness_region: witness_region,
      clusters: [ cluster_2.arn ]
    }
  )
  puts "Added #{cluster_2.arn} as a peer of #{cluster_1.arn}"

  # Now that multi_region_properties is fully defined for both clusters
  # they'll begin the transition to ACTIVE
  puts "Waiting for #{cluster_1.arn} to become ACTIVE"
  cluster_1 = client_1.wait_until(:cluster_active, identifier: cluster_1.identifier)
 do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end

  puts "Waiting for #{cluster_2.arn} to become ACTIVE"
  cluster_2 = client_2.wait_until(:cluster_active, identifier: cluster_2.identifier)
 do |w|
    w.max_attempts = 30
    w.delay = 10
  end

  [ cluster_1, cluster_2 ]
rescue Aws::Errors::ServiceError => e
  abort "Failed to create multi-region clusters: #{e.message}"
end

def main
  region_1 = "us-east-1"
  region_2 = "us-east-2"
  witness_region = "us-west-2"
```

```
  cluster_1, cluster_2 = create_multi_region_clusters(region_1, region_2,
 witness_region)

  puts "Created multi region clusters:"
  pp cluster_1
  pp cluster_2
end

main if $PROGRAM_NAME == __FILE__
```

Golang

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
 dtypes "github.com/aws/aws-sdk-go-v2/service/dsql/types"
)

func CreateMultiRegionClusters(ctx context.Context, witness, region1, region2
 string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region1))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Create a DSQL region 1 client
 client := dsql.NewFromConfig(cfg)

 cfg2, err := config.LoadDefaultConfig(ctx, config.WithRegion(region2))
 if err != nil {
```

```
 log.Fatalf("Failed to load AWS configuration: %v", err)
}

// Create a DSQL region 2 client
client2 := dsql.NewFromConfig(cfg2, func(o *dsql.Options) {
 o.Region = region2
})

// Create cluster
deleteProtect := true

// We can only set the witness region for the first cluster
input := &dsql.CreateClusterInput{
 DeletionProtectionEnabled: &deleteProtect,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String(witness),
 },
 Tags: map[string]string{
  "Name": "go multi-region cluster",
 },
}

clusterProperties, err := client.CreateCluster(context.Background(), input)

if err != nil {
 return fmt.Errorf("failed to create first cluster: %v", err)
}

// create second cluster
cluster2Arns := []string{*clusterProperties.Arn}

// For the second cluster we can set witness region and designate the first cluster
as a peer
input2 := &dsql.CreateClusterInput{
 DeletionProtectionEnabled: &deleteProtect,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String("us-west-2"),
  Clusters:      cluster2Arns,
 },
 Tags: map[string]string{
  "Name": "go multi-region cluster",
 },
}
```

```
clusterProperties2, err := client2.CreateCluster(context.Background(), input2)

if err != nil {
 return fmt.Errorf("failed to create second cluster: %v", err)
}

// link initial cluster to second cluster
cluster1Arns := []string{*clusterProperties2.Arn}

// Now that we know the second cluster arn we can set it as a peer of the first
cluster
input3 := dsql.UpdateClusterInput{
 Identifier: clusterProperties.Identifier,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String("us-west-2"),
  Clusters:      cluster1Arns,
 }}

_, err = client.UpdateCluster(context.Background(), &input3)

if err != nil {
 return fmt.Errorf("failed to update cluster to associate with first cluster. %v",
err)
}

// Create the waiter with our custom options for first cluster
waiter := dsql.NewClusterActiveWaiter(client, func(o
*dsql.ClusterActiveWaiterOptions) {
 o.MaxDelay = 30 * time.Second // Creating a multi-region cluster can take a few
minutes
 o.MinDelay = 10 * time.Second
 o.LogWaitAttempts = true
})

// Now that multiRegionProperties is fully defined for both clusters
// they'll begin the transition to ACTIVE

// Create the input for the clusterProperties to monitor for first cluster
getInput := &dsql.GetClusterInput{
 Identifier: clusterProperties.Identifier,
}

// Wait for the first cluster to become active
```

```
 fmt.Printf("Waiting for first cluster %s to become active...\n",
 *clusterProperties.Identifier)
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for first cluster to become active: %w", err)
 }

 // Create the waiter with our custom options
 waiter2 := dsql.NewClusterActiveWaiter(client2, func(o
 *dsql.ClusterActiveWaiterOptions) {
  o.MaxDelay = 30 * time.Second // Creating a multi-region cluster can take a few
 minutes
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })

 // Create the input for the clusterProperties to monitor for second
 getInput2 := &dsql.GetClusterInput{
  Identifier: clusterProperties2.Identifier,
 }

 // Wait for the second cluster to become active
 fmt.Printf("Waiting for second cluster %s to become active...\n",
 *clusterProperties2.Identifier)
 err = waiter2.Wait(ctx, getInput2, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for second cluster to become active: %w", err)
 }

 fmt.Printf("Cluster %s is now active\n", *clusterProperties.Identifier)
 fmt.Printf("Cluster %s is now active\n", *clusterProperties2.Identifier)
 return nil
}

// Example usage in main function
func main() {
 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := CreateMultiRegionClusters(ctx, "us-west-2", "us-east-1", "us-east-2")
 if err != nil {
  fmt.Printf("failed to create multi-region clusters: %v", err)
  panic(err)
```

```
 }

}
```

.NET

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class CreateMultiRegionClusters
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region,
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create multi-region clusters with a witness region.
        /// </summary>
```

```
        public static async Task<(CreateClusterResponse, CreateClusterResponse)>
Create(
            RegionEndpoint region1,
            RegionEndpoint region2,
            RegionEndpoint witnessRegion)
        {
            using (var client1 = await CreateDSQLClient(region1))
            using (var client2 = await CreateDSQLClient(region2))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp multi region cluster" }
                };

                // We can only set the witness region for the first cluster
                var createClusterRequest1 = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName
                    }
                };

                var cluster1 = await
client1.CreateClusterAsync(createClusterRequest1);
                var cluster1Arn = cluster1.Arn;
                Console.WriteLine($"Initiated creation of {cluster1Arn}");

                // For the second cluster we can set witness region and designate
cluster1 as a peer
                var createClusterRequest2 = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName,
                        Clusters = new List<string> { cluster1.Arn }
                    }
                };
```

```
                var cluster2 = await
 client2.CreateClusterAsync(createClusterRequest2);
                var cluster2Arn = cluster2.Arn;
                Console.WriteLine($"Initiated creation of {cluster2Arn}");

                // Now that we know the cluster2 arn we can set it as a peer of
 cluster1
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = cluster1.Identifier,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName,
                        Clusters = new List<string> { cluster2.Arn }
                    }
                };

                await client1.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Added {cluster2Arn} as a peer of
 {cluster1Arn}");

                return (cluster1, cluster2);
            }
        }

        private static async Task Main()
        {
            var region1 = RegionEndpoint.USEast1;
            var region2 = RegionEndpoint.USEast2;
            var witnessRegion = RegionEndpoint.USWest2;

            var (cluster1, cluster2) = await Create(region1, region2,
 witnessRegion);

            Console.WriteLine("Created multi region clusters:");
            Console.WriteLine($"Cluster 1: {cluster1.Arn}");
            Console.WriteLine($"Cluster 2: {cluster2.Arn}");
        }
    }
}
```

Dapatkan cluster

Contoh berikut menunjukkan cara mendapatkan informasi tentang cluster Multi-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk mendapatkan informasi tentang cluster Multi-region, gunakan contoh berikut.

```
import boto3
from datetime import datetime
import json


def get_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.get_cluster(identifier=identifier)
    except:
        print(f"Unable to get cluster {identifier} in region {region}")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    response = get_cluster(region, cluster_id)

    print(json.dumps(response, indent=2, default=lambda obj: obj.isoformat() if
 isinstance(obj, datetime) else None))


if __name__ == "__main__":
    main()
```

C++

Gunakan contoh berikut untuk mendapatkan informasi tentang cluster Multi-region.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
```

```
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Retrieves information about a cluster in Amazon Aurora DSQL
 */
GetClusterResult GetCluster(const Aws::String& region, const Aws::String&
 identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Get the cluster
    GetClusterRequest getClusterRequest;
    getClusterRequest.SetIdentifier(identifier);

    auto getOutcome = client.GetCluster(getClusterRequest);
    if (!getOutcome.IsSuccess()) {
        std::cerr << "Failed to retrieve cluster " << identifier << " in " << region
 << ": "
                  << getOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to retrieve cluster " + identifier + " in
 region " + region);
    }

    return getOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            auto cluster = GetCluster(region, clusterId);
```

```
            // Print cluster details
            std::cout << "Cluster Details:" << std::endl;
            std::cout << "ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk mendapatkan informasi tentang cluster Multi-region, gunakan contoh berikut.

```
import { DSQLClient, GetClusterCommand } from "@aws-sdk/client-dsql";

async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  const getClusterCommand = new GetClusterCommand({
    identifier: clusterId,
  });

  try {
    return await client.send(getClusterCommand);
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or deleted");
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  const response = await getCluster(region, clusterId);
```

```
  console.log("Cluster: ", response);
}

main();
```

Java

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster Multi-region.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

public class GetCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            GetClusterResponse cluster = client.getCluster(r ->
 r.identifier(clusterId));
            System.out.println(cluster);
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster Multi-region.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Get a ClusterResource from DSQL cluster identifier
pub async fn get_cluster(region: &'static str, identifier: &'static str) ->
 GetClusterOutput {
    let client = dsql_client(region).await;
    client
        .get_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = get_cluster(region, "<your cluster id>").await;
```

```
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster Multi-region.

```
require "aws-sdk-dsql"
require "pp"

def get_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  client.get_cluster(identifier: identifier)
rescue Aws::Errors::ServiceError => e
  abort "Unable to retrieve cluster #{identifier} in region #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  cluster = get_cluster(region, cluster_id)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster Multi-region.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
```

```
   public class GetCluster
   {
       /// <summary>
       /// Create a client. We will use this later for performing operations on the
cluster.
       /// </summary>
       private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
region)
       {
           var awsCredentials = await
DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
           var clientConfig = new AmazonDSQLConfig
           {
               RegionEndpoint = region
           };
           return new AmazonDSQLClient(awsCredentials, clientConfig);
       }

       /// <summary>
       /// Get information about a DSQL cluster.
       /// </summary>
       public static async Task<GetClusterResponse> Get(RegionEndpoint region,
string identifier)
       {
           using (var client = await CreateDSQLClient(region))
           {
               var getClusterRequest = new GetClusterRequest
               {
                   Identifier = identifier
               };

               return await client.GetClusterAsync(getClusterRequest);
           }
       }

       private static async Task Main()
       {
           var region = RegionEndpoint.USEast1;
           var clusterId = "<your cluster id>";

           var response = await Get(region, clusterId);
           Console.WriteLine($"Cluster ARN: {response.Arn}");
       }
   }
```

```
}
```

Golang

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster Multi-region.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func GetCluster(ctx context.Context, region, identifier string) (clusterStatus
 *dsql.GetClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := &dsql.GetClusterInput{
  Identifier: aws.String(identifier),
 }
 clusterStatus, err = client.GetCluster(context.Background(), input)

 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }

 log.Printf("Cluster ARN: %s", *clusterStatus.Arn)

 return clusterStatus, nil
}
```

```
func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 _, err := GetCluster(ctx, region, identifier)
 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }
}
```

Memperbarui klaster

Contoh berikut menunjukkan cara memperbarui cluster Multi-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk memperbarui cluster Multi-region, gunakan contoh berikut.

```
import boto3


def update_cluster(region, cluster_id, deletion_protection_enabled):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.update_cluster(identifier=cluster_id,
 deletionProtectionEnabled=deletion_protection_enabled)
    except:
        print("Unable to update cluster")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    deletion_protection_enabled = False
    response = update_cluster(region, cluster_id, deletion_protection_enabled)
```

```
    print(f"Updated {response["arn"]} with deletion_protection_enabled:
 {deletion_protection_enabled}")


if __name__ == "__main__":
    main()
```

C++

Gunakan contoh berikut untuk memperbarui cluster Multi-region.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Updates a cluster in Amazon Aurora DSQL
 */
UpdateClusterResult UpdateCluster(const Aws::String& region, const
 Aws::Map<Aws::String, Aws::String>& updateParams) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create update request
    UpdateClusterRequest updateRequest;
    updateRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Set identifier (required)
    if (updateParams.find("identifier") != updateParams.end()) {
        updateRequest.SetIdentifier(updateParams.at("identifier"));
    } else {
        throw std::runtime_error("Cluster identifier is required for update
 operation");
    }
```

```
    // Set deletion protection if specified
    if (updateParams.find("deletion_protection_enabled") != updateParams.end()) {
        bool deletionProtection = (updateParams.at("deletion_protection_enabled") ==
 "true");
        updateRequest.SetDeletionProtectionEnabled(deletionProtection);
    }

    // Execute the update
    auto updateOutcome = client.UpdateCluster(updateRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster: " <<
 updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to update cluster");
    }

    return updateOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and update parameters
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            // Create parameter map
            Aws::Map<Aws::String, Aws::String> updateParams;
            updateParams["identifier"] = clusterId;
            updateParams["deletion_protection_enabled"] = "false";

            auto updatedCluster = UpdateCluster(region, updateParams);

            std::cout << "Updated " << updatedCluster.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk memperbarui cluster Multi-region, gunakan contoh berikut.

```
import { DSQLClient, UpdateClusterCommand } from "@aws-sdk/client-dsql";

export async function updateCluster(region, clusterId, deletionProtectionEnabled) {

  const client = new DSQLClient({ region });

  const updateClusterCommand = new UpdateClusterCommand({
    identifier: clusterId,
    deletionProtectionEnabled: deletionProtectionEnabled
  });

  try {
    return await client.send(updateClusterCommand);
  } catch (error) {
    console.error("Unable to update cluster", error.message);
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";
  const deletionProtectionEnabled = false;

  const response = await updateCluster(region, clusterId,
 deletionProtectionEnabled);
  console.log(`Updated ${response.arn}`);
}

main();
```

Java

Gunakan contoh berikut untuk memperbarui cluster Multi-region.

```
package org.example;
```

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;
import software.amazon.awssdk.services.dsql.model.UpdateClusterResponse;

public class UpdateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            UpdateClusterRequest request = UpdateClusterRequest.builder()
                    .identifier(clusterId)
                    .deletionProtectionEnabled(false)
                    .build();
            UpdateClusterResponse cluster = client.updateCluster(request);
            System.out.println("Updated " + cluster.arn());
        }
    }
}
```

Rust

Gunakan contoh berikut untuk memperbarui cluster Multi-region.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::update_cluster::UpdateClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;
```

```
    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Update a DSQL cluster and set delete protection to false. Also add new tags.
pub async fn update_cluster(region: &'static str, identifier: &'static str) ->
 UpdateClusterOutput {
    let client = dsql_client(region).await;
    // Update delete protection
    let update_response = client
        .update_cluster()
        .identifier(identifier)
        .deletion_protection_enabled(false)
        .send()
        .await
        .unwrap();

    update_response
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = update_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Gunakan contoh berikut untuk memperbarui cluster Multi-region.

```
require "aws-sdk-dsql"

def update_cluster(region, update_params)
  client = Aws::DSQL::Client.new(region: region)
  client.update_cluster(update_params)
rescue Aws::Errors::ServiceError => e
  abort "Unable to update cluster: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  updated_cluster = update_cluster(region, {
    identifier: cluster_id,
    deletion_protection_enabled: false
  })
  puts "Updated #{updated_cluster.arn}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Gunakan contoh berikut untuk memperbarui cluster Multi-region.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class UpdateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
```

```
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Update a DSQL cluster and set delete protection to false.
        /// </summary>
        public static async Task<UpdateClusterResponse> Update(RegionEndpoint
 region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = identifier,
                    DeletionProtectionEnabled = false
                };

                UpdateClusterResponse response = await
 client.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Updated {response.Arn}");

                return response;
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            await Update(region, clusterId);
        }
    }
}
```

Golang

Gunakan contoh berikut untuk memperbarui cluster Multi-region.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func UpdateCluster(ctx context.Context, region, id string, deleteProtection bool)
 (clusterStatus *dsql.UpdateClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := dsql.UpdateClusterInput{
  Identifier:                &id,
  DeletionProtectionEnabled: &deleteProtection,
 }

 clusterStatus, err = client.UpdateCluster(context.Background(), &input)

 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }

 log.Printf("Cluster updated successfully: %v", clusterStatus.Status)
 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()
```

```
 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"
 deleteProtection := false

 _, err := UpdateCluster(ctx, region, identifier, deleteProtection)
 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }
}
```

Hapus cluster

Contoh berikut menunjukkan cara menghapus cluster Multi-region menggunakan bahasa pemrograman yang berbeda.

Python

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
import boto3


def delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2):
    try:

        client_1 = boto3.client("dsql", region_name=region_1)
        client_2 = boto3.client("dsql", region_name=region_2)

        client_1.delete_cluster(identifier=cluster_id_1)
        print(f"Deleting cluster {cluster_id_1} in {region_1}")

        # cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted

        client_2.delete_cluster(identifier=cluster_id_2)
        print(f"Deleting cluster {cluster_id_2} in {region_2}")

        # Now that both clusters have been marked for deletion they will transition
        # to DELETING state and finalize deletion
        print(f"Waiting for {cluster_id_1} to finish deletion")
```

```
        client_1.get_waiter("cluster_not_exists").wait(
            identifier=cluster_id_1,
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        print(f"Waiting for {cluster_id_2} to finish deletion")
        client_2.get_waiter("cluster_not_exists").wait(
            identifier=cluster_id_2,
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

    except:
        print("Unable to delete cluster")
        raise


def main():
    region_1 = "us-east-1"
    cluster_id_1 = "<cluster 1 id>"
    region_2 = "us-east-2"
    cluster_id_2 = "<cluster 2 id>"

    delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
    print(f"Deleted {cluster_id_1} in {region_1} and {cluster_id_2} in {region_2}")


if __name__ == "__main__":
    main()
```

C++

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
```

```
#include <aws/dsql/model/DeleteClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Deletes multi-region clusters in Amazon Aurora DSQL
 */
void DeleteMultiRegionClusters(
    const Aws::String& region1,
    const Aws::String& clusterId1,
    const Aws::String& region2,
    const Aws::String& clusterId2) {

    // Create clients for each region
    DSQL::DSQLClientConfiguration clientConfig1;
    clientConfig1.region = region1;
    DSQL::DSQLClient client1(clientConfig1);

    DSQL::DSQLClientConfiguration clientConfig2;
    clientConfig2.region = region2;
    DSQL::DSQLClient client2(clientConfig2);

    // Delete the first cluster
    std::cout << "Deleting cluster " << clusterId1 << " in " << region1 <<
 std::endl;

    DeleteClusterRequest deleteRequest1;
    deleteRequest1.SetIdentifier(clusterId1);
    deleteRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome1 = client1.DeleteCluster(deleteRequest1);
    if (!deleteOutcome1.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << clusterId1 << " in " << region1
 << ": "
                  << deleteOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to delete multi-region clusters");
    }
```

```
    // cluster1 will stay in PENDING_DELETE state until cluster2 is deleted
    std::cout << "Deleting cluster " << clusterId2 << " in " << region2 <<
 std::endl;

    DeleteClusterRequest deleteRequest2;
    deleteRequest2.SetIdentifier(clusterId2);
    deleteRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome2 = client2.DeleteCluster(deleteRequest2);
    if (!deleteOutcome2.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << clusterId2 << " in " << region2
 << ": "
                  << deleteOutcome2.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to delete multi-region clusters");
    }
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            Aws::String region1 = "us-east-1";
            Aws::String clusterId1 = "<your cluster id 1>";
            Aws::String region2 = "us-east-2";
            Aws::String clusterId2 = "<your cluster id 2>";

            DeleteMultiRegionClusters(region1, clusterId1, region2, clusterId2);

            std::cout << "Deleted " << clusterId1 << " in " << region1
                      << " and " << clusterId2 << " in " << region2 << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
import { DSQLClient, DeleteClusterCommand, waitUntilClusterNotExists } from "@aws-
sdk/client-dsql";

async function deleteMultiRegionClusters(region1, cluster1_id, region2, cluster2_id)
 {

    const client1 = new DSQLClient({ region: region1 });
    const client2 = new DSQLClient({ region: region2 });

    try {
        const deleteClusterCommand1 = new DeleteClusterCommand({
            identifier: cluster1_id,
        });
        const response1 = await client1.send(deleteClusterCommand1);

        const deleteClusterCommand2 = new DeleteClusterCommand({
            identifier: cluster2_id,
        });
        const response2 = await client2.send(deleteClusterCommand2);

        console.log(`Waiting for cluster1 ${response1.identifier} to finish
 deletion`);
        await waitUntilClusterNotExists(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster1 Id ${response1.identifier} is now deleted`);

        console.log(`Waiting for cluster2 ${response2.identifier} to finish
 deletion`);
        await waitUntilClusterNotExists(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
```

```
        );
        console.log(`Cluster2 Id ${response2.identifier} is now deleted`);
        return;
    } catch (error) {
        if (error.name === "ResourceNotFoundException") {
            console.log("Some or all Cluster ARNs not found or already deleted");
        } else {
            console.error("Unable to delete multi-region clusters: ",
 error.message);
        }
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const cluster1_id = "<CLUSTER_ID_1>";
    const region2 = "us-east-2";
    const cluster2_id = "<CLUSTER_ID_2>";

    const response = await deleteMultiRegionClusters(region1, cluster1_id, region2,
 cluster2_id);
    console.log(`Deleted ${cluster1_id} in ${region1} and ${cluster2_id} in
 ${region2}`);
}

main();
```

Java

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.DsqlClientBuilder;
import software.amazon.awssdk.services.dsql.model.DeleteClusterRequest;

import java.time.Duration;
```

```
public class DeleteMultiRegionClusters {

    public static void main(String[] args) {
        Region region1 = Region.US_EAST_1;
        String clusterId1 = "<your cluster id 1>";
        Region region2 = Region.US_EAST_2;
        String clusterId2 = "<your cluster id 2>";

        DsqlClientBuilder clientBuilder = DsqlClient.builder()
                .credentialsProvider(DefaultCredentialsProvider.create());

        try (
            DsqlClient client1 = clientBuilder.region(region1).build();
            DsqlClient client2 = clientBuilder.region(region2).build()
        ) {
            System.out.printf("Deleting cluster %s in %s%n", clusterId1, region1);
            DeleteClusterRequest request1 = DeleteClusterRequest.builder()
                    .identifier(clusterId1)
                    .build();
            client1.deleteCluster(request1);

            // cluster1 will stay in PENDING_DELETE until cluster2 is deleted
            System.out.printf("Deleting cluster %s in %s%n", clusterId2, region2);
            DeleteClusterRequest request2 = DeleteClusterRequest.builder()
                    .identifier(clusterId2)
                    .build();
            client2.deleteCluster(request2);

            // Now that both clusters have been marked for deletion they will
 transition
            // to DELETING state and finalize deletion.
            System.out.printf("Waiting for cluster %s to finish deletion%n",
 clusterId1);
            client1.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId1),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );

            System.out.printf("Waiting for cluster %s to finish deletion%n",
 clusterId2);
```

```
            client2.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId2),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );

            System.out.printf("Deleted %s in %s and %s in %s%n", clusterId1,
 region1, clusterId2, region2);
        }
    }
}
```

Rust

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{Client, Config};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
```

```
pub async fn delete_multi_region_clusters(
    region_1: &'static str,
    cluster_id_1: &'static str,
    region_2: &'static str,
    cluster_id_2: &'static str,
) {
    let client_1 = dsql_client(region_1).await;
    let client_2 = dsql_client(region_2).await;

    println!("Deleting cluster {cluster_id_1} in {region_1}");
    client_1
        .delete_cluster()
        .identifier(cluster_id_1)
        .send()
        .await
        .unwrap();

    // cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted
    println!("Deleting cluster {cluster_id_2} in {region_2}");
    client_2
        .delete_cluster()
        .identifier(cluster_id_2)
        .send()
        .await
        .unwrap();

    // Now that both clusters have been marked for deletion they will transition
    // to DELETING state and finalize deletion
    println!("Waiting for {cluster_id_1} to finish deletion");
    client_1
        .wait_until_cluster_not_exists()
        .identifier(cluster_id_1)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();

    println!("Waiting for {cluster_id_2} to finish deletion");
    client_2
        .wait_until_cluster_not_exists()
        .identifier(cluster_id_2)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}
```

```
#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region_1 = "us-east-1";
    let cluster_id_1 = "<cluster 1 to be deleted>";
    let region_2 = "us-east-2";
    let cluster_id_2 = "<cluster 2 to be deleted>";

    delete_multi_region_clusters(region_1, cluster_id_1, region_2,
 cluster_id_2).await;
    println!("Deleted {cluster_id_1} in {region_1} and {cluster_id_2} in
 {region_2}");

    Ok(())
}
```

Ruby

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
require "aws-sdk-dsql"

def delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
  client_1 = Aws::DSQL::Client.new(region: region_1)
  client_2 = Aws::DSQL::Client.new(region: region_2)

  puts "Deleting cluster #{cluster_id_1} in #{region_1}"
  client_1.delete_cluster(identifier: cluster_id_1)

  # cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted
  puts "Deleting #{cluster_id_2} in #{region_2}"
  client_2.delete_cluster(identifier: cluster_id_2)

  # Now that both clusters have been marked for deletion they will transition
  # to DELETING state and finalize deletion
  puts "Waiting for #{cluster_id_1} to finish deletion"
  client_1.wait_until(:cluster_not_exists, identifier: cluster_id_1) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
```

```
  puts "Waiting for #{cluster_id_2} to finish deletion"
  client_2.wait_until(:cluster_not_exists, identifier: cluster_id_2) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Failed to delete multi-region clusters: #{e.message}"
end

def main
  region_1 = "us-east-1"
  cluster_id_1 = "<your cluster id 1>"
  region_2 = "us-east-2"
  cluster_id_2 = "<your cluster id 2>"

  delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
  puts "Deleted #{cluster_id_1} in #{region_1} and #{cluster_id_2} in #{region_2}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class DeleteMultiRegionClusters
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
```

```
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
region)
        {
            var awsCredentials = await
DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region,
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Delete multi-region clusters.
        /// </summary>
        public static async Task Delete(
            RegionEndpoint region1,
            string clusterId1,
            RegionEndpoint region2,
            string clusterId2)
        {
            using (var client1 = await CreateDSQLClient(region1))
            using (var client2 = await CreateDSQLClient(region2))
            {
                var deleteRequest1 = new DeleteClusterRequest
                {
                    Identifier = clusterId1
                };

                var deleteResponse1 = await
client1.DeleteClusterAsync(deleteRequest1);
                Console.WriteLine($"Initiated deletion of {deleteResponse1.Arn}");

                // cluster 1 will stay in PENDING_DELETE state until cluster 2 is
deleted
                var deleteRequest2 = new DeleteClusterRequest
                {
                    Identifier = clusterId2
                };

                var deleteResponse2 = await
client2.DeleteClusterAsync(deleteRequest2);
                Console.WriteLine($"Initiated deletion of {deleteResponse2.Arn}");
```

```
            }
        }

        private static async Task Main()
        {
            var region1 = RegionEndpoint.USEast1;
            var cluster1 = "<cluster 1 to be deleted>";
            var region2 = RegionEndpoint.USEast2;
            var cluster2 = "<cluster 2 to be deleted>";

            await Delete(region1, cluster1, region2, cluster2);
        }
    }
}
```

Golang

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func DeleteMultiRegionClusters(ctx context.Context, region1, clusterId1, region2,
 clusterId2 string) error {
 // Load the AWS configuration for region 1
 cfg1, err := config.LoadDefaultConfig(ctx, config.WithRegion(region1))
 if err != nil {
  return fmt.Errorf("unable to load SDK config for region %s: %w", region1, err)
 }

 // Load the AWS configuration for region 2
 cfg2, err := config.LoadDefaultConfig(ctx, config.WithRegion(region2))
```

```
if err != nil {
 return fmt.Errorf("unable to load SDK config for region %s: %w", region2, err)
}

// Create DSQL clients for both regions
client1 := dsql.NewFromConfig(cfg1)
client2 := dsql.NewFromConfig(cfg2)

// Delete cluster in region 1
fmt.Printf("Deleting cluster %s in %s\n", clusterId1, region1)
_, err = client1.DeleteCluster(ctx, &dsql.DeleteClusterInput{
 Identifier: aws.String(clusterId1),
})
if err != nil {
 return fmt.Errorf("failed to delete cluster in region %s: %w", region1, err)
}

// Delete cluster in region 2
fmt.Printf("Deleting cluster %s in %s\n", clusterId2, region2)
_, err = client2.DeleteCluster(ctx, &dsql.DeleteClusterInput{
 Identifier: aws.String(clusterId2),
})
if err != nil {
 return fmt.Errorf("failed to delete cluster in region %s: %w", region2, err)
}

// Create waiters for both regions
waiter1 := dsql.NewClusterNotExistsWaiter(client1, func(options
*dsql.ClusterNotExistsWaiterOptions) {
 options.MinDelay = 10 * time.Second
 options.MaxDelay = 30 * time.Second
 options.LogWaitAttempts = true
})

waiter2 := dsql.NewClusterNotExistsWaiter(client2, func(options
*dsql.ClusterNotExistsWaiterOptions) {
 options.MinDelay = 10 * time.Second
 options.MaxDelay = 30 * time.Second
 options.LogWaitAttempts = true
})

// Wait for cluster in region 1 to be deleted
fmt.Printf("Waiting for cluster %s to finish deletion\n", clusterId1)
err = waiter1.Wait(ctx, &dsql.GetClusterInput{
```

```
  Identifier: aws.String(clusterId1),
 }, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster deletion in region %s: %w", region1,
 err)
 }

 // Wait for cluster in region 2 to be deleted
 fmt.Printf("Waiting for cluster %s to finish deletion\n", clusterId2)
 err = waiter2.Wait(ctx, &dsql.GetClusterInput{
  Identifier: aws.String(clusterId2),
 }, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster deletion in region %s: %w", region2,
 err)
 }

 fmt.Printf("Successfully deleted clusters %s in %s and %s in %s\n",
  clusterId1, region1, clusterId2, region2)
 return nil
}

// Example usage in main function
func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := DeleteMultiRegionClusters(
  ctx,
  "us-east-1",      // region1
  "<CLUSTER_ID_1>", // clusterId1
  "us-east-2",      // region2
  "<CLUSTER_ID_2>", // clusterId2
 )
 if err != nil {
  log.Fatalf("Failed to delete multi-region clusters: %v", err)
 }
}
```

Untuk contoh dan contoh kode lainnya, kunjungi repositori Aurora DSQL Sampel. GitHub

Penggunaan AWS CLI

AWS CLI menyediakan antarmuka baris perintah untuk mengelola cluster DSQL Aurora Multi-wilayah Anda. Contoh berikut menunjukkan cara membuat, mengkonfigurasi, dan menghapus klaster Multi-region.

Connect ke klaster Multi-region

Multi-Region cluster peered menyediakan dua titik akhir regional, satu di setiap cluster peered. Wilayah AWS Kedua titik akhir menyajikan database logis tunggal yang mendukung operasi baca dan tulis bersamaan dengan konsistensi data yang kuat. Selain cluster peered, cluster Multi-region juga memiliki Wilayah saksi yang menyimpan jendela terbatas log transaksi terenkripsi, yang digunakan untuk meningkatkan daya tahan dan ketersediaan Multi-region. Multi-Region saksi Daerah tidak memiliki titik akhir.

Buat klaster Multi-wilayah

Untuk membuat klaster Multi-wilayah, pertama-tama Anda membuat klaster dengan Wilayah saksi. Kemudian Anda mengintip cluster ini dengan cluster kedua yang berbagi Wilayah saksi yang sama dengan cluster pertama Anda. Contoh berikut menunjukkan cara membuat cluster di AS Timur (Virginia N.) dan AS Timur (Ohio) dengan US West (Oregon) sebagai Wilayah saksi.

Langkah 1: Buat cluster satu di AS Timur (Virginia N.)

Untuk membuat cluster di AS Timur (Virginia N.) Wilayah AWS dengan properti Multi-region, gunakan perintah di bawah ini.

```
aws dsql create-cluster \
--region us-east-1 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Respons:

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
```

```
        "encryptionStatus": "ENABLED"
    }
    "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

> **Note**
>
> Ketika operasi API berhasil, cluster memasuki `PENDING_SETUP` status. Pembuatan cluster tetap ada `PENDING_SETUP` sampai Anda memperbarui cluster dengan ARN dari peer cluster.

Langkah 2: Buat cluster dua di US East (Ohio)

Untuk membuat cluster di US East (Ohio) Wilayah AWS dengan properti Multi-region, gunakan perintah di bawah ini.

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Respons:

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
    "status": "PENDING_SETUP",
    "creationTime": "2025-05-06T06:51:16.145000-07:00",
    "deletionProtectionEnabled": true,
    "multiRegionProperties": {
        "witnessRegion": "us-west-2",
        "clusters": [
            "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"
        ]
    }
}
```

Ketika operasi API berhasil, klaster bertransisi ke `PENDING_SETUP` status. Pembuatan cluster tetap dalam `PENDING_SETUP` status sampai Anda memperbaruinya dengan ARN cluster lain untuk mengintip.

Langkah 3: Peer cluster di AS Timur (Virginia N.) dengan US East (Ohio)

Untuk mengintip cluster AS East (Virginia N.) Anda dengan cluster US East (Ohio) Anda, gunakan perintah. `update-cluster` Tentukan nama cluster US East (Virginia N.) Anda dan string JSON dengan ARN cluster US East (Ohio).

```
aws dsql update-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4' \
--multi-region-properties '{"witnessRegion": "us-west-2","clusters": ["arn:aws:dsql:us-
east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"]}'
```

Example Respons

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

Langkah 4: Peer cluster di AS Timur (Ohio) dengan AS Timur (Virginia N.)

Untuk mengintip cluster US East (Ohio) Anda dengan cluster US East (Virginia N.) Anda, gunakan perintah. `update-cluster` Tentukan nama cluster US East (Ohio) Anda dan string JSON dengan ARN cluster US East (N. Virginia).

Example

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5' \
--multi-region-properties '{"witnessRegion": "us-west-2", "clusters":
 ["arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Example Respons

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
```

```
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:51:16.145000-07:00"
}
```

> **Note**
>
> Setelah berhasil mengintip, kedua cluster beralih dari “PENDING_SETUP” ke “CREATING” dan akhirnya ke status “ACTIVE” saat siap digunakan.

Lihat properti klaster multi-wilayah

Saat mendeskripsikan sebuah klaster, Anda dapat melihat properti Multi-region untuk cluster secara berbeda. Wilayah AWS

Example

```
aws dsql get-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4'
```

Example Respons

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "PENDING_SETUP",
    "encryptionDetails": {
    "encryptionType": "AWS_OWNED_KMS_KEY",
    "encryptionStatus": "ENABLED"
},
    "creationTime": "2024-11-27T00:32:14.434000-08:00",
    "deletionProtectionEnabled": false,
    "multiRegionProperties": {
       "witnessRegion": "us-west-2",
       "clusters": [
          "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
          "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"
       ]
    }
}
```

Peer cluster selama pembuatan

Anda dapat mengurangi jumlah langkah dengan memasukkan informasi peering selama pembuatan cluster. Setelah membuat cluster pertama Anda di US East (N. Virginia) (Langkah 1), Anda dapat membuat cluster kedua Anda di US East (Ohio) sambil secara bersamaan memulai proses peering dengan memasukkan ARN dari cluster pertama.

Example

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2","clusters": ["arn:aws:dsql:us-
east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Ini menggabungkan Langkah 2 dan 4, tetapi Anda masih harus menyelesaikan Langkah 3 (memperbarui cluster pertama dengan ARN dari cluster kedua) untuk membangun hubungan peering. Setelah semua langkah selesai, kedua cluster akan bertransisi melalui status yang sama seperti dalam proses standar: dari PENDING_SETUP ke CREATING, dan akhirnya ke ACTIVE saat siap digunakan.

Hapus klaster Multi-wilayah

Untuk menghapus cluster Multi-region, Anda harus menyelesaikan dua langkah.

1. Matikan perlindungan penghapusan untuk setiap cluster.
2. Hapus setiap cluster peered secara terpisah di masing-masing Wilayah AWS

Perbarui dan hapus cluster di AS Timur (Virginia N.)

1. Matikan perlindungan penghapusan menggunakan perintah. `update-cluster`

   ```
   aws dsql update-cluster \
     --region us-east-1 \
     --identifier 'foo0bar1baz2quux3quuxquux4' \
     --no-deletion-protection-enabled
   ```

2. Hapus cluster menggunakan `delete-cluster` perintah.

   ```
   aws dsql delete-cluster \
     --region us-east-1 \
   ```

```
  --identifier 'foo0bar1baz2quux3quuxquux4'
```

Perintah mengembalikan respons berikut.

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
foo0bar1baz2quux3quuxquux4",
    "status": "PENDING_DELETE",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

> **Note**
>
> Transisi cluster ke PENDING_DELETE status. Penghapusan tidak selesai sampai Anda menghapus cluster peered di US East (Ohio).

Perbarui dan hapus cluster di AS Timur (Ohio)

1. Matikan perlindungan penghapusan menggunakan perintah. `update-cluster`

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quux4quuux' \
--no-deletion-protection-enabled
```

2. Hapus cluster menggunakan `delete-cluster` perintah.

```
aws dsql delete-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5'
```

Perintah mengembalikan respon berikut:

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/
foo0bar1baz2quux3quuxquux5",
    "status": "PENDING_DELETE",
```

```
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

> **Note**
>
> Transisi cluster ke `PENDING_DELETE` status. Setelah beberapa detik, sistem secara otomatis mentransisikan kedua cluster peered ke `DELETING` status setelah validasi.

Mengkonfigurasi cluster Aurora DSQL menggunakan AWS CloudFormation

Anda dapat menggunakan CloudFormation sumber daya yang sama `AWS::DSQL::Cluster` untuk menyebarkan dan mengelola klaster DSQL Aurora wilayah tunggal dan Multi-wilayah.

Lihat referensi jenis sumber daya Amazon Aurora DSQL untuk mengetahui lebih lanjut tentang cara membuat, memodifikasi, dan mengelola klaster menggunakan sumber daya. `AWS::DSQL::Cluster`

Membuat Konfigurasi Cluster Awal

Pertama, buat AWS CloudFormation template untuk menentukan cluster Multi-region Anda:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
```

Buat tumpukan di kedua Wilayah menggunakan perintah AWS CLI berikut:

```
aws cloudformation create-stack --region us-east-2 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

```
aws cloudformation create-stack --region us-east-1 \
    --stack-name MRCluster \
```

```
    --template-body file://mr-cluster.yaml
```

Menemukan Pengidentifikasi Cluster

Ambil ID sumber daya fisik untuk klaster Anda:

```
aws cloudformation describe-stack-resources -region us-east-2 \
    --stack-name MRCluster \
    --query 'StackResources[].PhysicalResourceId'
[
  "auabudrks5jwh4mjt6o5xxhr4y"
]
```

```
aws cloudformation describe-stack-resources -region us-east-1 \
    --stack-name MRCluster \
    --query 'StackResources[].PhysicalResourceId'
[
  "imabudrfon4p2z3nv2jo4rlajm"
]
```

Memperbarui Konfigurasi Cluster

Perbarui AWS CloudFormation template Anda untuk menyertakan kedua ARN cluster:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
        Clusters:
        - arn:aws:dsql:us-east-2:123456789012:cluster/auabudrks5jwh4mjt6o5xxhr4y
        - arn:aws:dsql:us-east-1:123456789012:cluster/imabudrfon4p2z3nv2jo4rlajm
```

Terapkan konfigurasi yang diperbarui ke kedua Wilayah:

```
aws cloudformation update-stack --region us-east-2 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

```
aws cloudformation update-stack --region us-east-1 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

Siklus hidup cluster Aurora DSQL

Memahami siklus hidup klaster Aurora DSQL membantu Anda mengelola cluster secara efektif. Bagian ini mencakup definisi status cluster dan fitur skala ke nol yang mengoptimalkan biaya.

Mendefinisikan status cluster Aurora DSQL

Status cluster Aurora DSQL memberikan informasi penting tentang kesehatan dan konektivitas cluster. Anda dapat melihat status cluster dan instance cluster dengan menggunakan Konsol Manajemen AWS, AWS CLI, atau Aurora DSQL API.

Tabel berikut menjelaskan setiap status yang mungkin untuk cluster Aurora DSQL dan apa arti setiap status.

Status	Deskripsi
Creating	Aurora DSQL sedang mencoba untuk membuat atau mengkonfigurasi sumber daya untuk cluster. Setiap upaya koneksi akan gagal saat cluster dalam keadaan ini.
Aktif	Cluster ini operasional dan siap digunakan.
Menganggur	Sebuah cluster menjadi idle ketika menganggur cukup lama bagi Aurora DSQL untuk mengurangi sumber daya yang sedang berjalan untuk mengurang i kapasitas dan biaya. Saat Anda terhubung ke cluster idle, Aurora DSQL mentransisikan cluster kembali ke status Aktif.
Tidak aktif	Cluster Idle menjadi Tidak Aktif ketika tidak ada aktivitas di cluster untuk waktu yang lama. Dalam status ditangguhkan ini, sumber daya yang sedang berjalan diskalakan ke nol saat data Anda dipertahankan. Saat Anda mencoba terhubung ke cluster yang tidak aktif, Aurora DSQL secara otomatis mentransi sikan cluster kembali ke status Aktif. Waktu untuk mengembalikan tergantung pada ukuran cluster.

Status	Deskripsi
Memperbarui	Transisi klaster ke status Memperbarui saat Anda membuat perubahan pada konfigurasi klaster.
Deleting	Transisi klaster ke status Menghapus saat Anda mengirimkan permintaan untuk menghapusnya.
Dihapus	Cluster telah berhasil dihapus.
Gagal	Aurora DSQL tidak dapat membuat cluster karena mengalami kesalahan.
Pengaturan Tertunda	Hanya untuk klaster Multi-wilayah. Kluster Multi-wilayah memasuki status Pengaturan Tertunda saat Anda membuat klaster Multi-wilayah di Wilayah pertama Anda dengan Wilayah saksi. Pembuatan cluster berhenti sampai Anda membuat cluster lain di Region sekunder dan mengintip kedua cluster bersama-sama.
Hapus Tertunda	Hanya untuk klaster Multi-wilayah. Kluster Multi-wilayah memasuki status Penghapusan Tertunda saat Anda menghapus klaster darinya. Cluster bergerak ke status Menghapus setelah Anda menghapus klaster rekan terakhir.

Bekerja dengan cluster Idle dan Inactive

Ketika Aurora DSQL mendeteksi tidak ada aktivitas koneksi pada cluster selama beberapa periode waktu, ia mentransisikan cluster ke status Idle, mengurangi sumber daya yang berjalan untuk meminimalkan kapasitas dan biaya. Jika aktivitas koneksi tetap tidak ada untuk jangka waktu yang lama, klaster Idle secara otomatis bertransisi ke status Tidak Aktif, di mana sumber daya yang sedang berjalan diskalakan ke nol saat data Anda disimpan.

Untuk melanjutkan operasi normal, cukup sambungkan ke cluster seperti biasa. Ketika Anda berhasil terhubung ke cluster, Aurora Aurora DSQL secara otomatis mentransisikan cluster ke status Aktif.

> **Note**
>
> Upaya koneksi pertama ke cluster Idle atau Inactive akan lebih lambat dari biasanya.

Operasi yang membutuhkan status cluster aktif

Beberapa operasi mengharuskan klaster Anda berada dalam status Aktif. Untuk melakukan operasi ini pada klaster Idle atau Inactive, Anda perlu mentransisikan klaster kembali ke Active dengan menghubungkan ke klaster Anda.

Operasi Backup

Mengambil cadangan memerlukan status klaster Aktif. Jika klaster Anda Idle atau Tidak Aktif, pencadangan gagal dengan kesalahan berikut:

```
"Error": {
    "Code": "FailedPrecondition",
    "Message": "Cluster 'cluster-id' is in state 'IDLE' and can't be backed up.
    In order to take a backup of your cluster, it must be in Active state. Please
    connect to your cluster to transition it to Active to perform the backup."
}
```

Untuk melanjutkan dengan cadangan:

1. Connect ke cluster menggunakan klien database pilihan Anda atau konsol Aurora DSQL untuk membangunkannya.
2. Tunggu transisi otomatis ke status Aktif.
3. Memulai backup setelah cluster beroperasi penuh.

> **Note**
>
> Pencadangan yang ada diambil sebelum klaster Idle atau Tidak Aktif tetap valid dan tidak terpengaruh. Upaya pencadangan baru di cluster akan gagal hingga cluster terhubung untuk auto wake up.

Melihat status cluster Aurora DSQL Anda

Untuk melihat status klaster Anda, gunakan Konsol Manajemen AWS AWS CLI, atau Aurora DSQL API.

Konsol

Ikuti langkah-langkah berikut untuk melihat status klaster di Konsol Manajemen AWS:

Untuk melihat status klaster di konsol

1. Buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Pilih Cluster di panel navigasi.
3. Lihat status untuk setiap cluster di dasbor.

AWS CLI

Gunakan AWS CLI perintah berikut untuk memeriksa status cluster tunggal.

```
aws dsql get-cluster --identifier cluster-id --query status --output text
```

Jalankan perintah berikut untuk daftar status semua cluster.

```
for id in $(aws dsql list-clusters --query 'clusters[*].identifier' --output text); do
  cluster_status=$(aws dsql get-cluster --identifier "$id" --query 'status' --output
 text)
  echo "$id    $cluster_status"
done
```

Output sampel ini menunjukkan dua cluster aktif dan satu cluster dalam proses dihapus.

```
aaabbb2bkx555xa7p42qd5cdef    ACTIVE
abcde123efghi77t35abcdefgh    ACTIVE
12abc6lqasc5bbbbbbbbbbbbbb    DELETING
```

Pemrograman dengan Aurora DSQL

Aurora DSQL memberi Anda alat berikut untuk mengelola sumber daya Aurora DSQL Anda secara terprogram.

AWS Command Line Interface (AWS CLI)

Anda dapat membuat dan mengelola sumber daya Anda dengan menggunakan AWS CLI shell dalam baris perintah. AWS CLI Menyediakan akses langsung ke API untuk Layanan AWS, seperti Aurora DSQL. Untuk sintaks dan contoh perintah untuk Aurora DSQL, lihat dsql di Command Reference.AWS CLI

AWS kit pengembangan perangkat lunak (SDK)

AWS menyediakan SDK untuk banyak teknologi populer dan bahasa pemrograman. Mereka memudahkan Anda untuk menelepon Layanan AWS dari dalam aplikasi Anda dalam bahasa atau teknologi itu. Untuk informasi selengkapnya tentang SDK ini, lihat Alat untuk mengembangkan dan mengelola aplikasi. AWS

Aurora DSQL API

API ini adalah antarmuka pemrograman lain untuk Aurora DSQL. Saat menggunakan API ini, Anda harus memformat setiap permintaan HTTPS dengan benar dan menambahkan tanda tangan digital yang valid ke setiap permintaan. Untuk informasi selengkapnya, lihat Referensi API.

CloudFormation

AWS: :DSQL: :Cluster adalah CloudFormation sumber daya yang memungkinkan Anda membuat dan mengelola klaster Aurora DSQL sebagai bagian dari infrastruktur Anda sebagai kode. CloudFormation membantu Anda menentukan seluruh AWS lingkungan Anda dalam kode, membuatnya lebih mudah untuk menyediakan, memperbarui, dan mereplikasi infrastruktur Anda dengan cara yang konsisten dan andal.

Saat Anda menggunakan sumber daya AWS: :DSQL: :Cluster di CloudFormation template, Anda dapat secara deklaratif menyediakan klaster Aurora DSQL bersama sumber daya cloud Anda yang lain. Ini membantu memastikan bahwa infrastruktur data Anda menyebarkan dan mengelola bersama sisa tumpukan aplikasi Anda.

Konektor untuk Aurora DSQL

Aurora DSQL menyediakan konektor khusus yang memperluas driver database yang ada untuk memungkinkan otentikasi dan integrasi IAM yang mulus dengan layanan. AWS Konektor ini dirancang untuk bekerja dengan bahasa dan kerangka kerja pemrograman populer sambil mempertahankan kompatibilitas dengan alur kerja PostgreSQL yang ada.

Konektor tambahan direncanakan untuk rilis masa depan. Untuk informasi terbaru tentang ketersediaan konektor, lihat repositori sampel Aurora DSQL.

Konektor Aurora DSQL untuk Java JDBC

Konektor Aurora DSQL untuk JDBC dirancang sebagai plugin otentikasi yang memperluas fungsionalitas driver PostgreSQL JDBC untuk memungkinkan aplikasi mengautentikasi dengan Aurora DSQL menggunakan kredensil IAM. Konektor tidak terhubung langsung ke database, tetapi menyediakan otentikasi IAM yang mulus di atas driver PostgreSQL JDBC yang mendasarinya.

Konektor Aurora DSQL untuk JDBC dibangun untuk bekerja dengan Driver PostgreSQL JDBC dan menyediakan integrasi tanpa batas dengan persyaratan otentikasi IAM Aurora DSQL.

Dalam hubungannya dengan PostgreSQL JDBC Driver, Aurora DSQL Connector untuk JDBC memungkinkan otentikasi untuk Aurora DSQL. IAM-based Ini memperkenalkan integrasi mendalam dengan layanan AWS otentikasi seperti AWS Identity and Access Management(IAM).

Tentang konektornya

Aurora DSQL adalah layanan database SQL terdistribusi yang menyediakan ketersediaan dan skalabilitas tinggi untuk aplikasi. PostgreSQL-compatible Aurora DSQL memerlukan IAM-based otentikasi dengan token terbatas waktu yang tidak didukung oleh driver JDBC yang ada secara native.

Ide utama di balik Konektor Aurora DSQL untuk JDBC adalah menambahkan lapisan otentikasi di atas driver PostgreSQL JDBC yang menangani pembuatan token IAM, memungkinkan pengguna untuk terhubung ke Aurora DSQL tanpa mengubah alur kerja JDBC yang ada.

Apa itu Autentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan IAM-based otentikasi dengan token terbatas waktu

- Pembuatan Token: Token otentikasi dihasilkan menggunakan AWS kredensil dan memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk JDBC dirancang untuk memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Manfaat Konektor Aurora DSQL untuk JDBC

Meskipun Aurora DSQL menyediakan antarmuka PostgreSQL-compatible , driver PostgreSQL yang ada saat ini tidak mendukung persyaratan otentikasi IAM Aurora DSQL. Konektor Aurora DSQL untuk JDBC memungkinkan pelanggan untuk terus menggunakan alur kerja PostgreSQL yang ada sambil mengaktifkan otentikasi IAM melalui:

- Pembuatan Token Otomatis: Token IAM dihasilkan secara otomatis menggunakan kredensil AWS
- Integrasi Seamless: Bekerja dengan pola koneksi JDBC yang ada
- AWS Credentials Support: Mendukung berbagai penyedia AWS kredensi (default, berbasis profil, dll.)

Menggunakan Konektor Aurora DSQL untuk JDBC dengan penyatuan koneksi

Konektor Aurora DSQL untuk JDBC bekerja dengan pustaka penyatuan koneksi seperti HiKaricp. Konektor menangani pembuatan token IAM selama pembentukan koneksi, memungkinkan kumpulan koneksi beroperasi secara normal.

Fitur utama

Pembuatan Token Otomatis

Token IAM dihasilkan secara otomatis menggunakan AWS kredensil.

Integrasi yang mulus

Bekerja dengan pola koneksi JDBC yang ada tanpa memerlukan perubahan alur kerja.

AWS Credentials Support

Mendukung berbagai penyedia AWS kredensi (default, berbasis profil, dll.).

Kompatibilitas Pengumpulan Koneksi

Bekerja mulus dengan pustaka penyatuan koneksi seperti HikariCP.

Prasyarat

Sebelum Anda mulai, pastikan Anda telah menyelesaikan prasyarat berikut:

- Membuat cluster di Aurora DSQL.
- Menginstal Java Development Kit (JDK). Pastikan Anda memiliki versi 17 atau lebih tinggi.
- Siapkan izin IAM yang sesuai untuk memungkinkan aplikasi Anda terhubung ke Aurora DSQL.
- AWS kredensil dikonfigurasi (via AWS CLI, variabel lingkungan, atau peran IAM).

Menggunakan Konektor Aurora DSQL untuk JDBC

Untuk menggunakan Konektor Aurora DSQL untuk JDBC di aplikasi Java Anda, ikuti langkah-langkah berikut:

1. Tambahkan dependensi berikut ke proyek Maven Anda:

```
<dependencies>
    <!-- Aurora DSQL Connector for JDBC -->
    <dependency>
        <groupId>software.amazon.dsql</groupId>
        <artifactId>aurora-dsql-jdbc-connector</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

   Untuk proyek Gradle, tambahkan dependensi ini:

```
implementation("software.amazon.dsql:aurora-dsql-jdbc-connector:1.0.0")
```

2. Buat koneksi dasar ke cluster Aurora DSQL Anda menggunakan format konektor DSQL AWS PostgreSQL:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DsqlJdbcConnectorExample {
    public static void main(String[] args) {
```

```
        // Using AWS DSQL PostgreSQL Connector prefix
        String jdbcUrl = "jdbc:aws-dsql:postgresql://your-cluster.dsql.us-
east-1.on.aws/postgres?user=admin";

        try (Connection connection = DriverManager.getConnection(jdbcUrl)) {
            // Use the connection
            try (Statement statement = connection.createStatement()) {
                // Create a table
                statement.execute("CREATE TABLE IF NOT EXISTS test_table (id UUID
 PRIMARY KEY DEFAULT gen_random_uuid(), name VARCHAR(100))");

                // Insert data
                statement.execute("INSERT INTO test_table (name) VALUES ('Test
 Name')");

                // Query data
                try (ResultSet resultSet = statement.executeQuery("SELECT * FROM
 test_table")) {
                    while (resultSet.next()) {
                        System.out.println("ID: " + resultSet.getInt("id") + ",
 Name: " + resultSet.getString("name"));
                    }
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Properti konfigurasi

Konektor Aurora DSQL untuk JDBC mendukung properti koneksi berikut:

user

Menentukan pengguna untuk koneksi dan metode pembuatan token yang digunakan. Contoh: `admin`

token-durasi-detik

Durasi dalam hitungan detik untuk validitas token. Untuk informasi selengkapnya tentang batas token, lihat Membuat token otentikasi di Amazon Aurora DSQL.

profile

Digunakan untuk membuat instance ProfileCredentialsProvider untuk pembuatan token dengan nama profil yang disediakan.

region

AWS wilayah untuk koneksi Aurora DSQL. Hal ini bersifat opsional. Ketika disediakan, itu akan mengganti wilayah yang diekstrak dari URL.

basis data

Nama database untuk terhubung ke. Default-nya adalah `postgres`.

Pencatatan log

Aktifkan pencatatan untuk memecahkan masalah apa pun yang mungkin Anda alami saat menggunakan konektor Aurora DSQL JDBC.

Konektor menggunakan sistem logging bawaan (java.util.logging) dari Java. Anda dapat mengonfigurasi level logging dengan membuat `logging.properties` file:

```
# Set root logger level to INFO for clean output
.level = INFO

# Show Aurora DSQL Connector for JDBC FINE logs for detailed debugging
software.amazon.dsql.level = FINE

# Console handler configuration
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Detailed formatter pattern with timestamp and logger name
java.util.logging.SimpleFormatter.format = %1$tH:%1$tM:%1$tS.%1$tL [%4$s] %3$s - %5$s%n
```

Contoh

Untuk contoh dan kasus penggunaan yang lebih komprehensif, lihat Konektor Aurora DSQL untuk repositori JDBC

Konektor Aurora DSQL untuk Python

Konektor Aurora DSQL untuk Python mengintegrasikan Autentikasi IAM untuk menghubungkan aplikasi Python ke cluster Amazon Aurora DSQL. Secara internal, ia menggunakan pustaka klien psycopg, psycopg2, dan asyncpg.

Konektor Aurora DSQL untuk Python dirancang sebagai plugin otentikasi yang memperluas fungsionalitas pustaka klien psycopg, psycopg2, dan asyncpg untuk memungkinkan aplikasi mengautentikasi dengan Amazon Aurora DSQL menggunakan kredenal IAM. Konektor tidak terhubung langsung ke database tetapi menyediakan otentikasi IAM yang mulus di atas pustaka klien yang mendasarinya.

Tentang Konektor

Amazon Aurora DSQL adalah layanan database SQL terdistribusi yang menyediakan ketersediaan dan skalabilitas tinggi untuk aplikasi. PostgreSQL-compatible Aurora DSQL memerlukan IAM-based otentikasi dengan token terbatas waktu yang tidak didukung oleh pustaka Python yang ada secara asli.

Ide di balik Konektor Aurora DSQL untuk Python adalah menambahkan lapisan otentikasi di atas pustaka klien psycopg, psycopg2, dan asyncpg yang menangani pembuatan token IAM, memungkinkan pengguna untuk terhubung ke Aurora DSQL tanpa mengubah alur kerja yang ada.

Apa itu Autentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan IAM-based otentikasi dengan token terbatas waktu
- Pembuatan Token: Token otentikasi dibuat menggunakan kredensi AWS dan memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk Python dirancang untuk memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM Otomatis - Token IAM dihasilkan secara otomatis menggunakan kredensi AWS
- Dibangun di atas psycopg, psycopg2, dan asyncpg - Memanfaatkan pustaka klien psycopg, psycopg2, dan asyncpg
- Integrasi Seamless - Bekerja dengan pola koneksi psycopg, psycopg2, dan asyncpg yang ada tanpa memerlukan perubahan alur kerja
- Region Auto-Discovery - Mengekstrak wilayah AWS dari nama host cluster DSQL
- AWS Credentials Support - Mendukung berbagai penyedia kredensi AWS (default, berbasis profil, kustom)
- Kompatibilitas Pengumpulan Koneksi - Bekerja dengan pengumpulan koneksi bawaan psycopg, psycopg2, dan asyncpg

Panduan memulai cepat

Persyaratan

- Python 3.10 atau lebih tinggi
- Akses ke cluster Aurora DSQL
- Siapkan izin IAM yang sesuai untuk memungkinkan aplikasi Anda terhubung ke Aurora DSQL.
- Kredensi AWS dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

```
pip install aurora-dsql-python-connector
```

Instal psycopg atau psycopg2 atau asyncpg secara terpisah

Konektor Aurora DSQL untuk penginstal Python tidak menginstal pustaka yang mendasarinya. Mereka perlu diinstal secara terpisah, misalnya:

```
# Install psycopg and psycopg pool
pip install "psycopg[binary,pool]"
```

```
# Install psycopg2
pip install psycopg2-binary
```

```
# Install asyncpg
pip install asyncpg
```

Catatan:

Hanya perpustakaan yang dibutuhkan yang harus diinstal. Oleh karena itu, jika klien akan menggunakan psycopg, maka hanya psycopg yang perlu diinstal. Jika klien akan menggunakan psycopg2, maka hanya psycopg2 yang perlu diinstal. Jika klien akan menggunakan asyncpg, maka hanya asyncpg yang perlu diinstal.

Jika klien membutuhkan lebih dari satu, maka semua perpustakaan yang dibutuhkan perlu diinstal.

Penggunaan Dasar

psycopg

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = dsql.connect(**config)
```

```
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = await dsql.connect(**config)
    result = await conn.fetchrow("SELECT 1")
    await conn.close()
    print(result)
```

Hanya menggunakan host

psycopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql
```

```
    conn = await dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

Hanya menggunakan ID cluster

psycopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("your-cluster")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("your-cluster")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster")
```

Catatan:

Dalam skenario “hanya menggunakan ID cluster”, wilayah yang telah ditetapkan sebelumnya pada mesin digunakan, misalnya:

```
aws configure set region us-east-1
```

Jika wilayah belum disetel, atau ID cluster yang diberikan berada di wilayah yang berbeda, koneksi akan gagal. Untuk membuatnya berfungsi, berikan wilayah sebagai parameter seperti pada contoh di bawah ini:

psycopg

```
    import aurora_dsql_psycopg as dsql

    config = {
```

```
        "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
        "region": "us-east-1",
    }

    conn = await dsql.connect("your-cluster", **config)
```

String Koneksi

psycopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/
postgres?user=admin&token_duration_secs=15")
```

Konfigurasi Lanjutan

psycopg

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
```

```
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = await dsql.connect(**config)
    result = await conn.fetchrow("SELECT 1")
    await conn.close()
    print(result)
```

Opsi konfigurasi

Opsi	Tipe	Diperlu	Deskripsi
`host`	`string`	Ya	Nama host cluster DSQL atau ID cluster
`user`	`string`	Tidak	Nama pengguna DSQL. Default: admin
`dbname`	`string`	Tidak	Nama basis data. Default: postgres
`region`	`string`	Tidak	Wilayah AWS (terdeteksi secara otomatis dari nama host jika tidak disediakan)
`port`	`int`	Tidak	Default ke 5432
`custom_credentials_provider`	`CredentialProvider`	Tidak	Penyedia kredensi AWS khusus

Opsi	Tipe	Diperl n	Deskripsi
`profile`	`string`	Tidak	Nama profil IAM. Default: default.
`token_dur ation_secs`	`int`	Tidak	Waktu kedaluwarsa token dalam hitungan detik

Semua opsi koneksi standar dari pustaka psycopg, psycopg2, dan asyncpg yang mendasarinya juga didukung, dengan pengecualian parameter asyncpg krbsrvname dan gsslib yang tidak didukung oleh DSQL.

Menggunakan konektor Aurora DSQL untuk Python dengan penyatuan koneksi

Konektor Aurora DSQL untuk Python bekerja dengan psycopg, psycopg2, dan asyncpg built-in connection pooling. Konektor menangani pembuatan token IAM selama pembentukan koneksi, memungkinkan kumpulan koneksi beroperasi secara normal.

psycopg

Untuk psycopg, konektor mengimplementasikan kelas koneksi bernama DSqlConnection yang dapat diteruskan langsung ke psycopg_pool. ConnectionPool konstruktor. Untuk operasi asinkron, ada juga versi async dari kelas bernama. DSQLAsyncConnection

```
from psycopg_pool import ConnectionPool as PsycopgPool

...
pool = PsycopgPool(
    "",
    connection_class=dsql.DSQLConnection,
    kwargs=conn_params,
    min_size=2,
    max_size=8,
    max_lifetime=3300
)
```

Catatan: Konfigurasi koneksi max_lifetime

Parameter max_lifetime harus disetel ke kurang dari 3600 detik (satu jam), karena ini adalah durasi koneksi maksimum yang diizinkan oleh database Aurora DSQL. Menyetel max_lifetime yang lebih

rendah memungkinkan kumpulan koneksi untuk secara proaktif mengelola daur ulang koneksi, yang lebih efisien daripada menangani kesalahan batas waktu koneksi dari database.

psycopg2

Untuk psycopg2, konektor menyediakan kelas bernama yang mewarisi dari AuroraDSQLThreadedConnectionPool psycopg2.pool. ThreadedConnectionPool. AuroraDSQLThreadedConnectionPool Kelas hanya mengganti metode _connect internal. Sisa implementasi disediakan oleh psycopg2.pool. ThreadedConnectionPool tidak berubah.

```
    import aurora_dsql_psycopg2 as dsql

    pool = dsql.AuroraDSQLThreadedConnectionPool(
            minconn=2,
            maxconn=8,
            **conn_params,
    )
```

asyncpg

Untuk asyncpg, konektor menyediakan fungsi create_pool yang mengembalikan instance Asyncpg.pool.

```
    import asyncio
    import os

    import aurora_dsql_asyncpg as dsql

    pool_params = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'user': "admin",
        "min_size": 2,
        "max_size": 5,
    }

    pool = await dsql.create_pool(**pool_params)
```

Autentikasi

Konektor secara otomatis menangani otentikasi DSQL dengan menghasilkan token menggunakan generator token klien DSQL. Jika wilayah AWS tidak disediakan, wilayah AWS akan diurai secara otomatis dari nama host yang disediakan.

Untuk informasi selengkapnya tentang otentikasi di Aurora DSQL, lihat panduan pengguna.

Admin vs Pengguna Reguler

- Pengguna bernama `"admin"` secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi non-admin
- Token dihasilkan secara dinamis untuk setiap koneksi

Contoh

Untuk kode contoh lengkap, lihat contoh seperti yang ditunjukkan pada bagian di bawah ini. Untuk petunjuk cara menjalankan contoh, silakan lihat contoh file READMDE.

psycopg

Contoh README

Deskripsi	Contoh
Menggunakan Konektor Aurora DSQL untuk Python untuk koneksi dasar	Contoh Koneksi Dasar
Menggunakan Konektor Aurora DSQL untuk Python untuk koneksi asinkron dasar	Contoh Koneksi Asinkron Dasar
Menggunakan Konektor Aurora DSQL untuk Python dengan kumpulan koneksi	Contoh Koneksi Dasar Dengan Connection Pool
	Contoh Koneksi Bersamaan Dengan Connection Pool
Menggunakan Konektor Aurora DSQL untuk Python dengan kumpulan koneksi asinkron	Contoh Koneksi Dasar Dengan Kolam Koneksi Asinkron

psycopg2

Contoh README

Deskripsi	Contoh
Menggunakan Konektor Aurora DSQL untuk Python untuk koneksi dasar	Contoh Koneksi Dasar
Menggunakan Konektor Aurora DSQL untuk Python dengan kumpulan koneksi	Contoh Koneksi Dasar Dengan Connection Pool
	Contoh Koneksi Bersamaan Dengan Connection Pool

asyncpg

Contoh README

Deskripsi	Contoh
Menggunakan Konektor Aurora DSQL untuk Python untuk koneksi dasar	Contoh Koneksi Dasar
Menggunakan Konektor Aurora DSQL untuk Python dengan kumpulan koneksi	Contoh Koneksi Dasar Dengan Connection Pool
	Contoh Koneksi Bersamaan Dengan Connection Pool

Konektor Aurora DSQL untuk Go pgx

Konektor Aurora DSQL untuk Go membungkus pgx dengan otentikasi IAM otomatis. Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi sehingga Anda dapat fokus pada logika aplikasi Anda.

Tentang konektornya

Aurora DSQL memerlukan IAM-based otentikasi dengan token terbatas waktu yang tidak didukung oleh driver Go PostgreSQL yang ada. Konektor Aurora DSQL untuk Go menambahkan lapisan otentikasi di atas driver pgx yang menangani pembuatan token IAM, memungkinkan Anda terhubung ke Aurora DSQL tanpa mengubah alur kerja pgx yang ada.

Apa itu Autentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan IAM-based otentikasi dengan token terbatas waktu
- Generasi Token: Konektor menghasilkan token otentikasi menggunakan AWS kredensyal, dan token ini memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk Go dirancang untuk memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Manfaat Konektor Aurora DSQL untuk Go

Konektor Aurora DSQL untuk Go memungkinkan Anda untuk terus menggunakan alur kerja pgx yang ada sambil mengaktifkan otentikasi IAM melalui:

- Generasi Token Otomatis: Konektor menghasilkan token IAM secara otomatis untuk setiap koneksi
- Connection Pooling: Built-in dukungan untuk `pgxpool` pembuatan token otomatis per koneksi
- Konfigurasi Fleksibel: Dukungan untuk titik akhir penuh atau ID cluster dengan deteksi otomatis wilayah
- AWS Credentials Support: Mendukung AWS profil dan penyedia kredensyal khusus

Fitur utama

Manajemen Token Otomatis

Konektor menghasilkan token IAM secara otomatis untuk setiap koneksi baru menggunakan kredensyal yang telah diselesaikan sebelumnya.

Pengumpulan Koneksi

Penggabungan koneksi melalui `pgxpool` pembuatan token otomatis per koneksi.

Konfigurasi Host Fleksibel

Mendukung titik akhir cluster penuh dan ID cluster dengan deteksi wilayah otomatis.

Keamanan SSL

SSL selalu diaktifkan dengan mode verify-full dan negosiasi TLS langsung.

Prasyarat

- Pergi 1.24 atau yang lebih baru
- AWS kredensil dikonfigurasi
- Kluster Aurora DSQL

Konektor menggunakan rantai kredensi default AWS SDK for Go v2, yang menyelesaikan kredensyal dengan urutan berikut:

1. Variabel lingkungan (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)
2. File kredensial bersama (~/. aws/credentials)
3. File konfigurasi bersama (~/. aws/config)
4. Peran IAM untuk Amazon EC2/ECS/Lambda

Penginstalan

Instal konektor menggunakan modul Go:

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql
```

Quick start

Contoh berikut menunjukkan cara membuat kolam koneksi dan mengeksekusi query:

```
package main

import (
    "context"
    "log"
```

```
    "github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql"
)

func main() {
    ctx := context.Background()

    // Create a connection pool
    pool, err := dsql.NewPool(ctx, dsql.Config{
        Host: "your-cluster.dsql.us-east-1.on.aws",
    })
    if err != nil {
        log.Fatal(err)
    }
    defer pool.Close()

    // Execute a query
    var greeting string
    err = pool.QueryRow(ctx, "SELECT 'Hello, DSQL!'").Scan(&greeting)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(greeting)
}
```

Opsi konfigurasi

Konektor mendukung opsi konfigurasi berikut:

Bidang	Tipe	Default	Deskripsi
Host	string	(wajib)	Titik akhir klaster atau ID klaster
Region	string	(terdeteksi secara otomatis)	AWS wilayah; diperlukan jika Host adalah ID cluster
Pengguna	string	“admin”	Pengguna basis data
Basis Data	string	“postgres”	Nama basis data
Port	int	5432	Port basis data
Profil	string	""	AWS nama profil untuk kredensil

Bidang	Tipe	Default	Deskripsi
TokenDuration Secs	int	900 (15 menit)	Durasi validitas token dalam hitungan detik (maks diperbolehkan: 1 minggu, default: 15 menit)
MaxConns	int32	0	Koneksi kolam maksimum (0 = pgxpool default)
MinConns	int32	0	Koneksi kolam minimum (0 = pgxpool default)
MaxConnLi fetime	Waktu.durasi	55 menit	Masa pakai koneksi maksimum

Format string koneksi

Konektor mendukung format string koneksi PostgreSQL dan DSQL:

```
postgres://[user@]host[:port]/[database][?param=value&...]
dsql://[user@]host[:port]/[database][?param=value&...]
```

Parameter kueri yang didukung:

- `region`- AWS wilayah
- `profile`- nama AWS profil
- `tokenDurationSecs`- Durasi validitas Token dalam hitungan detik

Contoh:

```
// Full endpoint (region auto-detected)
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres")

// Using dsql:// scheme (also supported)
pool, _ := dsql.NewPool(ctx, "dsql://admin@cluster.dsql.us-east-1.on.aws/postgres")

// With explicit region
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/mydb?
region=us-east-1")
```

```
// With AWS profile
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?
profile=dev")
```

Penggunaan lanjutan

Konfigurasi host

Konektor mendukung dua format host:

Titik akhir penuh (wilayah terdeteksi secara otomatis):

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
})
```

ID Cluster (diperlukan wilayah):

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host:   "your-cluster-id",
    Region: "us-east-1",
})
```

Penyetelan konfigurasi kolam

Konfigurasikan kumpulan koneksi untuk beban kerja Anda:

```
pool, err := dsql.NewPool(ctx, dsql.Config{
    Host:              "your-cluster.dsql.us-east-1.on.aws",
    MaxConns:          20,
    MinConns:          5,
    MaxConnLifetime:   time.Hour,
    MaxConnIdleTime:   30 * time.Minute,
    HealthCheckPeriod: time.Minute,
})
```

Penggunaan koneksi tunggal

Untuk skrip sederhana atau saat penyatuan koneksi tidak diperlukan:

```
conn, err := dsql.Connect(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
```

```
})
if err != nil {
    log.Fatal(err)
}
defer conn.Close(ctx)

// Use the connection
rows, err := conn.Query(ctx, "SELECT * FROM users")
```

Penggunaan AWS profil

Tentukan AWS profil untuk kredensil:

```
pool, err := dsql.NewPool(ctx, dsql.Config{
    Host:    "your-cluster.dsql.us-east-1.on.aws",
    Profile: "production",
})
```

Coba lagi OCC

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC). Ketika dua transaksi memodifikasi data yang sama, yang pertama melakukan kemenangan dan yang kedua menerima kesalahan OCC.

`occretry`Paket ini menyediakan pembantu untuk percobaan ulang otomatis dengan backoff eksponensial dan jitter. Instal dengan:

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/occretry
```

Gunakan `WithRetry` untuk penulisan transaksional:

```
err := occretry.WithRetry(ctx, pool, occretry.DefaultConfig(), func(tx pgx.Tx) error {
    _, err := tx.Exec(ctx, "UPDATE accounts SET balance = balance - $1 WHERE id = $2",
 100, fromID)
    if err != nil {
        return err
    }
    _, err = tx.Exec(ctx, "UPDATE accounts SET balance = balance + $1 WHERE id = $2",
 100, toID)
    return err
})
```

Untuk pernyataan DDL atau tunggal, gunakan`ExecWithRetry`:

```
err := occretry.ExecWithRetry(ctx, pool, occretry.DefaultConfig(),
    "CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY, name TEXT)")
```

> **Important**
>
> `WithRetry`mengelola`BEGIN/COMMIT/ROLLBACK`secara internal. Callback Anda menerima transaksi dan harus hanya berisi operasi database dan aman untuk dicoba lagi.

Contoh

Untuk contoh dan kasus penggunaan yang lebih komprehensif, lihat contoh Aurora DSQL Connector for Go.

Contoh	Deskripsi
example_preferred	Direkomendasikan: Kolam koneksi dengan kueri bersamaan
transaksi	Penanganan transaksi dengan BEGIN/COMMIT/ROLLBACK
occ_coba lagi	Menangani konflik OCC dengan backoff eksponensial
connection_string	Menggunakan string koneksi untuk konfigurasi

Konektor Aurora DSQL untuk Node.js

Konektor Aurora DSQL untuk node-postgres dan Konektor Aurora DSQL untuk adalah plugin otentikasi yang memperluas fungsionalitas node-postgres dan Postgres.js klien untuk memungkinkan aplikasi mengautentikasi dengan Aurora DSQL menggunakan kredensil IAM. Postgres.js

Konektor Aurora DSQL untuk node-postgres

Konektor Aurora DSQL untuk node-postgres adalah konektor Node.js yang dibangun di atas node-postgres yang mengintegrasikan Autentikasi IAM untuk menghubungkan aplikasi ke cluster DSQL Amazon Aurora. JavaScript/TypeScript

Konektor Aurora DSQL dirancang sebagai plugin otentikasi yang memperluas fungsionalitas Klien dan Pool node-postgres untuk memungkinkan aplikasi mengautentikasi dengan Amazon Aurora DSQL menggunakan kredensil IAM.

Tentang Konektor

Amazon Aurora DSQL adalah database terdistribusi cloud-native dengan kompatibilitas PostgreSQL. Meskipun memerlukan otentikasi IAM dan token terikat waktu, driver database Node.js tradisional tidak memiliki dukungan bawaan ini.

Konektor Aurora DSQL untuk node-postgres menjembatani kesenjangan ini dengan menerapkan middleware otentikasi yang bekerja mulus dengan node-postgres. Pendekatan ini memungkinkan pengembang untuk mempertahankan kode node-postgres mereka yang ada sambil mendapatkan akses berbasis IAM yang aman ke cluster Aurora DSQL melalui manajemen token otomatis.

Apa itu Autentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan otentikasi berbasis IAM dengan token terbatas waktu
- Pembuatan Token: Token otentikasi dibuat menggunakan kredensi AWS dan memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk node-postgres dirancang untuk memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM Otomatis - Menangani pembuatan dan penyegaran token DSQL
- Dibangun di atas node-postgres - Memanfaatkan klien PostgreSQL populer untuk Node.js
- Integrasi Seamless - Bekerja dengan pola koneksi node-postgres yang ada
- Region Auto-Discovery - Mengekstrak wilayah AWS dari nama host cluster DSQL
- Full TypeScript Support - Menyediakan keamanan tipe penuh
- AWS Credentials Support - Mendukung berbagai penyedia kredensi AWS (default, berbasis profil, kustom)
- Kompatibilitas Pengumpulan Koneksi - Bekerja dengan mulus dengan penyatuan koneksi bawaan

Contoh Aplikasi

Ada contoh aplikasi yang disertakan dalam contoh yang menunjukkan cara menggunakan Aurora DSQL Connector untuk node-postgres. Untuk menjalankan contoh yang disertakan, silakan lihat contoh README.

Panduan memulai cepat

Persyaratan

- Node.js 20+
- Akses ke cluster Aurora DSQL
- Siapkan izin IAM yang sesuai untuk memungkinkan aplikasi Anda terhubung ke Aurora DSQL.
- Kredensi AWS dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

```
npm install @aws/aurora-dsql-node-postgres-connector
```

Dependensi Sebaya

```
npm install @aws-sdk/credential-providers @aws-sdk/dsql-signer pg tsx
npm install --save-dev @types/pg
```

Penggunaan

Koneksi Klien

```
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
});
await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Koneksi Kolam

```
import { AuroraDSQLPool } from "@aws/aurora-dsql-node-postgres-connector";

const pool = new AuroraDSQLPool({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
  max: 3,
```

```
  idleTimeoutMillis: 60000,
});

const result = await pool.query("SELECT NOW()");
```

Penggunaan Lanjutan

```
import { fromNodeProviderChain } from "@aws-sdk/credential-providers";
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "example.dsql.us-east-1.on.aws",
  user: "admin",
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
});

await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Opsi konfigurasi

Opsi	Tipe	Dipe n	Deskripsi
`host`	`string`	Ya	Nama host kluster DSQL
`username`	`string`	Ya	Nama pengguna DSQL
`database`	`string`	Tida	Nama basis data
`region`	`string`	Tida	Wilayah AWS (terdeteksi secara otomatis dari nama host jika tidak disediakan)
`port`	`number`	Tida	Default ke 5432
`customCredentialsProvider`	`AwsCredentialIdentity / AwsCredentialIdentityProvider`	Tida	Penyedia kredensi AWS kustom

Opsi	Tipe	Dipe n	Deskripsi
`profile`	`string`	Tida	Nama profil IAM. Default ke “default”
`tokenDuration Secs`	`number`	Tida	Waktu kedaluwarsa token dalam hitungan detik

Semua parameter lain dari Klien/Pool didukung.

Autentikasi

Konektor secara otomatis menangani otentikasi DSQL dengan menghasilkan token menggunakan generator token klien DSQL. Jika wilayah AWS tidak disediakan, wilayah AWS akan diurai secara otomatis dari nama host yang disediakan.

Untuk informasi selengkapnya tentang otentikasi di Aurora DSQL, lihat panduan pengguna.

Admin vs Pengguna Reguler

- Pengguna bernama “admin” secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi reguler
- Token dihasilkan secara dinamis untuk setiap koneksi

Konektor Aurora DSQL untuk Postgres.js

Konektor Aurora DSQL untuk Postgres.js adalah konektor yang dibangun di atas Postgres.jsyang Node.js mengintegrasikan Otentikasi IAM untuk menghubungkan aplikasi ke JavaScript cluster Amazon Aurora DSQL.

Konektor Aurora DSQL untuk Postgres.js dirancang sebagai plugin otentikasi yang memperluas fungsionalitas Postgres.js klien untuk memungkinkan aplikasi mengautentikasi dengan Amazon Aurora DSQL menggunakan kredensil IAM. Konektor tidak terhubung langsung ke database, tetapi menyediakan otentikasi IAM yang mulus di atas driver yang mendasarinya. Postgres.js

Tentang Konektor

Amazon Aurora DSQL adalah layanan database SQL terdistribusi yang menyediakan ketersediaan dan skalabilitas tinggi untuk aplikasi. PostgreSQL-compatible Aurora DSQL memerlukan IAM-based

otentikasi dengan token terbatas waktu yang tidak didukung oleh driver yang ada Node.js secara native.

Ide di balik Konektor Aurora DSQL untuk Postgres.js adalah menambahkan lapisan otentikasi di atas Postgres.js klien yang menangani pembuatan token IAM, memungkinkan pengguna untuk terhubung ke Aurora DSQL tanpa mengubah alur kerja yang ada. Postgres.js

Konektor Aurora DSQL untuk Postgres.js bekerja dengan sebagian besar versi. Postgres.js Pengguna menyediakan versi mereka sendiri dengan menginstal Postgres.js secara langsung.

Apa itu Autentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan IAM-based otentikasi dengan token terbatas waktu
- Pembuatan Token: Token otentikasi dibuat menggunakan kredensi AWS dan memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk dirancang untuk Postgres.js memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM Otomatis - Menangani pembuatan dan penyegaran token DSQL
- Dibangun di atas Postgres.js - Memanfaatkan klien PostgreSQL cepat untuk Node.js
- Integrasi Seamless - Bekerja dengan pola Postgres.js koneksi yang ada
- Region Auto-Discovery - Mengekstrak wilayah AWS dari nama host cluster DSQL
- Full TypeScript Support - Menyediakan keamanan tipe penuh
- AWS Credentials Support - Mendukung berbagai penyedia kredensi AWS (default, berbasis profil, kustom)
- Kompatibilitas Pengumpulan Koneksi - Bekerja dengan mulus dengan Postgres.js 'penyatuan koneksi bawaan

Panduan memulai cepat

Persyaratan

- Node.js 20+
- Akses ke cluster Aurora DSQL

- Siapkan izin IAM yang sesuai untuk memungkinkan aplikasi Anda terhubung ke Aurora DSQL.
- Kredensi AWS dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

```
npm install @aws/aurora-dsql-postgresjs-connector
# Postgres.js is a peer-dependency, so users must install it themselves
npm install postgres
```

Penggunaan Dasar

```
import { auroraDSQLPostgres } from '@aws/aurora-dsql-postgresjs-connector';

const sql = auroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  username: 'admin',

});

// Execute queries
const result = await sql`SELECT current_timestamp`;
console.log(result);

// Clean up
await sql.end();
```

Menggunakan ID cluster alih-alih host

```
const sql = auroraDSQLPostgres({
  host: 'your-cluster-id',
  region: 'us-east-1',
  username: 'admin',

});
```

String Koneksi

```
const sql = AuroraDSQLPostgres(
  'postgres://admin@your-cluster.dsql.us-east-1.on.aws'
);
```

```
const result = await sql`SELECT current_timestamp`;
```

Konfigurasi Lanjutan

```
import { fromNodeProviderChain } from '@aws-sdk/credential-providers';

const sql = AuroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  database: 'postgres',
  username: 'admin',
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
  tokenDurationSecs: 3600,                            // Token expiration (seconds)

  // Standard Postgres.js options
  max: 20,                                // Connection pool size
  ssl: { rejectUnauthorized: false }    // SSL configuration
});
```

Opsi konfigurasi

Opsi	Tipe	Diperlu	Deskripsi
`host`	`string`	Ya	Nama host cluster DSQL atau ID cluster
`database`	`string?`	Tidak	Nama basis data
`username`	`string?`	Tidak	Nama pengguna database (menggunakan admin jika tidak disediakan)
`region`	`string?`	Tidak	Wilayah AWS (terdeteksi secara otomatis dari nama host jika tidak disediakan)
`customCredentialsProvider`	`AwsCredentialIdentityProvider?`	Tidak	Penyedia kredensi AWS khusus
`tokenDurationSecs`	`number?`	Tidak	Waktu kedaluwarsa token dalam hitungan detik

Semua Postgres.jsopsi standar juga didukung.

Autentikasi

Konektor secara otomatis menangani otentikasi DSQL dengan menghasilkan token menggunakan generator token klien DSQL. Jika wilayah AWS tidak disediakan, wilayah AWS akan diurai secara otomatis dari nama host yang disediakan.

Untuk informasi selengkapnya tentang otentikasi di Aurora DSQL, lihat panduan pengguna.

Admin vs Pengguna Reguler

- Pengguna bernama “admin” secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi reguler
- Token dihasilkan secara dinamis untuk setiap koneksi

Penggunaan sampel

JavaScript contoh menggunakan Konektor Aurora DSQL untuk Postgres.js tersedia di. GitHub Untuk petunjuk tentang cara menjalankan contoh, lihat direktori examples.

Deskripsi	Contoh
Penggabungan koneksi dengan kueri bersamaan , termasuk pembuatan tabel, penyisipan, dan pembacaan di beberapa pekerja	Contoh kolam koneksi (lebih disukai)
Operasi CRUD (buat tabel, sisipkan, pilih, hapus) tanpa penyatuan koneksi	Contoh tanpa kolam koneksi

Konektor Aurora DSQL untuk Ruby pg

Konektor Aurora DSQL untuk Ruby adalah konektor Ruby yang dibangun di atas pg yang mengintegrasikan otentikasi IAM untuk menghubungkan aplikasi Ruby ke cluster Amazon Aurora DSQL.

Konektor menangani pembuatan token, konfigurasi SSL, dan penyatuan koneksi sehingga Anda dapat fokus pada logika aplikasi Anda.

Tentang konektornya

Amazon Aurora DSQL memerlukan otentikasi IAM dengan token terbatas waktu yang tidak didukung oleh driver Ruby PostgreSQL yang ada. Konektor Aurora DSQL untuk Ruby menambahkan lapisan otentikasi di atas permata pg yang menangani pembuatan token IAM, memungkinkan Anda untuk terhubung ke Aurora DSQL tanpa mengubah alur kerja pg yang ada.

Apa itu otentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan otentikasi berbasis IAM dengan token terbatas waktu
- Generasi Token: Konektor menghasilkan token otentikasi menggunakan AWS kredensil, dan token ini memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk Ruby memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM otomatis - Menangani pembuatan dan penyegaran token Aurora DSQL
- Dibangun di pg - Membungkus permata PostgreSQL populer untuk Ruby
- Integrasi mulus - Bekerja dengan alur kerja pg gem yang ada
- Penggabungan koneksi - Dukungan bawaan melalui **`connection_pool`** permata dengan penegakan max_lifetime
- Deteksi otomatis wilayah - Ekstrak AWS wilayah dari nama host cluster Aurora DSQL
- AWS dukungan kredensil - Mendukung AWS profil dan penyedia kredensil khusus
- Coba lagi OCC - Coba lagi kontrol konkurensi optimis dengan backoff eksponensial

Contoh aplikasi

Untuk contoh lengkap, lihat contoh aplikasi di GitHub.

Panduan memulai cepat

Persyaratan

- Ruby 3.1 atau yang lebih baru
- Akses ke cluster Aurora DSQL
- AWS kredensyal yang dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

Tambahkan ke Gemfile Anda:

```
gem "aurora-dsql-ruby-pg"
```

Atau instal langsung:

```
gem install aurora-dsql-ruby-pg
```

Penggunaan

Koneksi kolam

```
require "aurora_dsql_pg"

# Create a connection pool with OCC retry enabled
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  occ_max_retries: 3
)

# Read
pool.with do |conn|
  result = conn.exec("SELECT 'Hello, DSQL!'")
  puts result[0]["?column?"]
end

# Write — you must wrap writes in a transaction
pool.with do |conn|
  conn.transaction do
```

```
    conn.exec_params("INSERT INTO users (id, name) VALUES (gen_random_uuid(), $1)",
 ["Alice"])
  end
end

pool.shutdown
```

Koneksi tunggal

Untuk skrip sederhana atau saat penyatuan koneksi tidak diperlukan:

```
conn = AuroraDsql::Pg.connect(host: "your-cluster.dsql.us-east-1.on.aws")
conn.exec("SELECT 1")
conn.close
```

Penggunaan lanjutan

Konfigurasi host

Konektor mendukung titik akhir cluster penuh (wilayah terdeteksi secara otomatis) dan cluster IDs (wilayah diperlukan):

```
# Full endpoint (region auto-detected)
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws"
)

# Cluster ID (region required)
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster-id",
  region: "us-east-1"
)
```

AWS profil

Tentukan AWS profil untuk kredensyal:

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  profile: "production"
)
```

Format string koneksi

Konektor mendukung format string koneksi PostgreSQL:

```
postgres://[user@]host[:port]/[database][?param=value&...]
postgresql://[user@]host[:port]/[database][?param=value&...]
```

Parameter kueri yang didukung:`region`,`profile`,`tokenDurationSecs`.

```
# Full endpoint with profile
pool = AuroraDsql::Pg.create_pool(
  "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?profile=dev"
)
```

Coba lagi OCC

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC). Ketika dua transaksi memodifikasi data yang sama, yang pertama melakukan kemenangan dan yang kedua menerima kesalahan OCC.

Coba lagi OCC adalah opt-in. Setel `occ_max_retries` saat membuat kumpulan untuk mengaktifkan percobaan ulang otomatis dengan backoff eksponensial dan jitter pada: `pool.with`

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  occ_max_retries: 3
)

pool.with do |conn|
  conn.transaction do
    conn.exec_params("UPDATE accounts SET balance = balance - $1 WHERE id = $2", [100,
 from_id])
    conn.exec_params("UPDATE accounts SET balance = balance + $1 WHERE id = $2", [100,
 to_id])
  end
end
```

> **Warning**
>
> `pool.with`TIDAK secara otomatis membungkus blok Anda dalam transaksi. Anda harus menelepon `conn.transaction` diri Anda sendiri untuk operasi menulis. Pada konflik

OCC konektor mengeksekusi ulang seluruh blok, sehingga seharusnya hanya berisi operasi database dan aman untuk dicoba lagi.

Untuk melewati percobaan ulang pada panggilan individual, teruskan`retry_occ: false`:

```
pool.with(retry_occ: false) do |conn|
  conn.exec("SELECT 1")
end
```

Opsi konfigurasi

Bidang	Tipe	Default	Deskripsi
host	String	(wajib)	Titik akhir klaster atau ID klaster
region	String	(terdeteksi secara otomatis)	AWS wilayah; diperlukan jika host adalah ID cluster
user	String	“admin”	Pengguna basis data
basis data	String	“postgres”	Nama basis data
port	Bilangan Bulat	5432	Port basis data
profile	String	kosong	AWS nama profil untuk kredensyal
token_duration	Bilangan Bulat	900 (15 menit)	Durasi validitas token dalam hitungan detik (maks diperbolehkan: 1 minggu, default: 15 menit)
credentials_provider	Aws: :Kredensyal	kosong	Penyedia kredensyal kustom
max_seumur hidup	Bilangan Bulat	3300 (55 menit)	Masa pakai koneksi maksimum dalam hitungan detik
application_name	String	kosong	Awalan ORM untuk application_name

Bidang	Tipe	Default	Deskripsi
logger	Pencatat	kosong	Logger untuk peringatan coba lagi OCC
occ_max_m encoba lagi	Bilangan Bulat	nil (dinonakt ifkan)	Max OCC mencoba ulang`pool.with` ; memungkinkan coba lagi saat disetel

`create_pool`juga menerima `pool:` kata kunci dengan hash opsi yang Anda berikan langsung ke. `ConnectionPool.new` Jika Anda menghilangkan`pool:`, konektor default ke. `{size: 5, timeout: 5}` Kunci yang Anda berikan hanya mengganti default tertentu tersebut.

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  pool: { size: 10, timeout: 10 }
)
```

Autentikasi

Konektor secara otomatis menangani otentikasi Aurora DSQL dengan menghasilkan token menggunakan kredensil. AWS Jika Anda tidak menyediakan AWS wilayah, konektor menguraikannya dari nama host.

Untuk informasi lebih lanjut tentang otentikasi di Aurora DSQL, lihat. Otentikasi dan otorisasi untuk Aurora DSQL

Admin vs pengguna biasa

- Pengguna bernama “admin” secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi reguler
- Konektor menghasilkan token secara dinamis untuk setiap koneksi

Konektor Aurora DSQL untuk PHP **PDO_PGSQL**

Konektor Aurora DSQL untuk PHP adalah konektor PHP yang dibangun di atas PDO_PGSQLyang mengintegrasikan otentikasi IAM untuk menghubungkan aplikasi PHP ke cluster Amazon Aurora DSQL.

Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi sehingga Anda dapat fokus pada logika aplikasi Anda.

Tentang konektornya

Konektor Aurora DSQL untuk PHP menambahkan lapisan otentikasi di atasnya `PDO_PGSQL` yang menangani pembuatan token IAM, memungkinkan Anda untuk terhubung ke Aurora DSQL menggunakan alur kerja PDO yang ada. Amazon Aurora DSQL memerlukan otentikasi IAM dengan token terbatas waktu, dan konektor secara otomatis mengelola pembuatan token untuk Anda.

Apa itu otentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan otentikasi berbasis IAM dengan token terbatas waktu
- Generasi Token: Konektor menghasilkan token otentikasi menggunakan AWS kredensyal, dan token ini memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk PHP memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM otomatis - Menangani pembuatan dan penyegaran token Aurora DSQL
- Dibangun di atas **`PDO_PGSQL`** - Membungkus ekstensi PostgreSQL standar untuk PHP
- Integrasi mulus - Bekerja dengan alur kerja PDO yang ada
- Penegakan SSL - Selalu menggunakan SSL dengan `verify-full` mode dan negosiasi TLS langsung
- Deteksi otomatis wilayah - Ekstrak AWS wilayah dari nama host cluster Aurora DSQL
- AWS dukungan kredensyal - Mendukung AWS profil dan penyedia kredensyal khusus
- Coba lagi OCC - Coba lagi kontrol konkurensi optimis dengan backoff eksponensial dan jitter
- Pencatatan PSR-3 - Pencatatan yang kompatibel untuk diagnostik coba lagi

Contoh aplikasi

Untuk contoh lengkap, lihat contoh aplikasi di GitHub.

Panduan memulai cepat

Persyaratan

- PHP 8.2 atau yang lebih baru
- `ext-pdo_pgsql`perpanjangan
- Akses ke cluster Aurora DSQL
- AWS kredensyal yang dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

Tambahkan paket ke proyek Anda:

```
composer require awslabs/aurora-dsql-pdo-pgsql
```

Penggunaan

Koneksi berbasis konfigurasi

```
<?php

require_once 'vendor/autoload.php';

use Aws\AuroraDsql\PdoPgsql\AuroraDsql;
use Aws\AuroraDsql\PdoPgsql\DsqlConfig;

$config = new DsqlConfig(
    host: 'your-cluster.dsql.us-east-1.on.aws',
    occMaxRetries: 3
);
$pdo = AuroraDsql::connect($config);

// Read
$stmt = $pdo->query('SELECT 1 AS result');
$row = $stmt->fetch(PDO::FETCH_ASSOC);
echo "Connected: {$row['result']}\n";

// Transactional write with automatic OCC retry
$id = $pdo->transaction(function (PDO $conn): string {
    $stmt = $conn->prepare('INSERT INTO users (name) VALUES (?) RETURNING id');
    $stmt->execute(['Alice']);
```

```
    return $stmt->fetchColumn();
});
```

Format string koneksi

Untuk skrip sederhana atau ketika Anda lebih suka sintaks string koneksi, konektor mendukung `postgres://` dan `postgresql://` koneksi string dengan parameter kueri khusus Aurora DSQL:

```
$pdo = AuroraDsql::connectFromDsn(
    'postgres://admin@your-cluster.dsql.us-east-1.on.aws/postgres?region=us-east-1'
);
```

Parameter kueri yang didukung: `region`,`profile`,`tokenDurationSecs`,,`ormPrefix`.

Coba lagi OCC

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC). Ketika dua transaksi memodifikasi data yang sama, yang pertama melakukan kemenangan dan yang kedua menerima kesalahan OCC.

Coba lagi OCC adalah opt-in. Setel `occMaxRetries` dalam konfigurasi untuk mengaktifkan percobaan ulang otomatis dengan backoff dan jitter eksponensial:

```
$config = new DsqlConfig(
    host: 'your-cluster.dsql.us-east-1.on.aws',
    occMaxRetries: 3
);
$pdo = AuroraDsql::connect($config);

// Single statements are automatically retried via exec()
$pdo->exec("CREATE INDEX ASYNC ON users (email)");

// Multi-statement transactions are retried via transaction()
$pdo->transaction(function (PDO $conn) {
    $conn->exec("UPDATE accounts SET balance = balance - 100 WHERE id = 1");
    $conn->exec("UPDATE accounts SET balance = balance + 100 WHERE id = 2");
});
```

> **Important**
>
> `transaction()`mengelolabeginTransaction()/commit()/rollBack()secara internal. Callback Anda seharusnya hanya berisi operasi database dan aman untuk dicoba lagi.

Opsi konfigurasi

Konektor juga menerima `postgres://` dan `postgresql://` menghubungkan string dengan parameter kueri untuk konfigurasi. Parameter kueri yang didukung: `region`,`profile`,`tokenDurationSecs`,, dan`ormPrefix`.

Bidang	Tipe	Default	Deskripsi
`host`	`string`	(wajib)	Titik akhir cluster atau ID cluster 26-char
`region`	`?string`	`null`(terdeteksi otomatis)	AWS wilayah; diperlukan jika `host` adalah ID cluster
`user`	`string`	`"admin"`	Pengguna basis data
`database`	`string`	`"postgres"`	Nama basis data
`port`	`int`	`5432`	Port basis data
`profile`	`?string`	`null`	AWS nama profil untuk kredensyal
`credentialsProvider`	`?\Closure`	`null`	Penyedia AWS kredensyal kustom
`tokenDurationSecs`	`int`	`900`(15 menit)	Durasi validitas token dalam hitungan detik
`occMaxRetries`	`?int`	`null`(dinonaktifkan)	Percobaan ulang OCC maks default untuk dan `exec()` `transaction()`
`ormPrefix`	`?string`	`null`	Awalan ORM diawali `application_name`
`logger`	`?LoggerInterface`	`null`	PSR-3 logger untuk peringatan dan diagnostik coba lagi

Autentikasi

Konektor secara otomatis menangani otentikasi Aurora DSQL dengan menghasilkan token menggunakan kredensyal. AWS Jika Anda tidak memberikan AWS wilayah, konektor menguraikannya dari nama host.

Untuk informasi lebih lanjut tentang otentikasi di Aurora DSQL, lihat. Otentikasi dan otorisasi untuk Aurora DSQL

Admin vs pengguna biasa

- Pengguna bernama “admin” secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi reguler
- Konektor menghasilkan token secara dinamis untuk setiap koneksi

Konektor Aurora DSQL untuk .NET Npgsql

Konektor Aurora DSQL untuk .NET adalah konektor.NET yang dibangun di atas Npgsql yang mengintegrasikan otentikasi IAM untuk menghubungkan aplikasi.NET ke cluster Amazon Aurora DSQL.

Konektor menangani pembuatan token, konfigurasi SSL, dan penyatuan koneksi sehingga Anda dapat fokus pada logika aplikasi Anda.

Tentang konektornya

Amazon Aurora DSQL memerlukan otentikasi IAM dengan token terbatas waktu yang tidak didukung oleh driver.NET PostgreSQL yang ada. Konektor Aurora DSQL untuk.NET menambahkan lapisan otentikasi di atas Npgsql yang menangani pembuatan token IAM, memungkinkan Anda untuk terhubung ke Aurora DSQL tanpa mengubah alur kerja Npgsql yang ada.

Apa itu otentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan otentikasi berbasis IAM dengan token terbatas waktu
- Generasi Token: Konektor menghasilkan token otentikasi menggunakan AWS kredensyal, dan token ini memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk.NET memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM otomatis - Menangani pembuatan dan penyegaran token Aurora DSQL
- Dibangun di Npgsql - Membungkus driver PostgreSQL populer untuk.NET
- Integrasi mulus - Bekerja dengan alur kerja Npgsql yang ada
- Penggabungan koneksi - Dukungan bawaan melalui `NpgsqlDataSource` dengan penegakan seumur hidup maksimal
- Deteksi otomatis wilayah - Ekstrak AWS wilayah dari nama host cluster Aurora DSQL
- AWS dukungan kredensyal - Mendukung AWS profil dan penyedia kredensyal khusus
- Coba lagi OCC - Coba lagi kontrol konkurensi optimis dengan backoff eksponensial
- Penegakan SSL - Selalu menggunakan SSL dengan `verify-full` mode dan negosiasi TLS langsung

Contoh aplikasi

Untuk contoh lengkap, lihat contoh aplikasi di GitHub.

Panduan memulai cepat

Persyaratan

- .NET 8.0 atau yang lebih baru
- Akses ke cluster Aurora DSQL
- AWS kredensyal yang dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

Tambahkan paket ke proyek Anda:

```
dotnet add package Amazon.AuroraDsql.Npgsql
```

Penggunaan

Koneksi kolam

```
using Amazon.AuroraDsql.Npgsql;

// Create a connection pool
await using var ds = await AuroraDsql.CreateDataSourceAsync(new DsqlConfig
{
    Host = "your-cluster.dsql.us-east-1.on.aws",
    OccMaxRetries = 3
});

// Read
await using (var conn = await ds.OpenConnectionAsync())
{
    await using var cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT 'Hello, DSQL!'";
    var greeting = await cmd.ExecuteScalarAsync();
    Console.WriteLine(greeting);
}

// Transactional write with OCC retry
await ds.WithTransactionRetryAsync(async conn =>
{
    await using var cmd = conn.CreateCommand();
    cmd.CommandText = "INSERT INTO users (id, name) VALUES (gen_random_uuid(), @name)";
    cmd.Parameters.AddWithValue("name", "Alice");
    await cmd.ExecuteNonQueryAsync();
});
```

Koneksi tunggal

Untuk skrip sederhana atau saat Anda tidak memerlukan penyatuan koneksi:

```
await using var conn = await AuroraDsql.ConnectAsync(new DsqlConfig
{
    Host = "your-cluster.dsql.us-east-1.on.aws"
});

await using var cmd = conn.CreateCommand("SELECT 1");
await cmd.ExecuteScalarAsync();
```

Coba lagi OCC

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC). Ketika dua transaksi memodifikasi data yang sama, yang pertama melakukan kemenangan dan yang kedua menerima kesalahan OCC.

Coba lagi OCC adalah opt-in. Setel `OccMaxRetries` dalam konfigurasi untuk mengaktifkan percobaan ulang otomatis dengan backoff eksponensial dan jitter. Gunakan `WithTransactionRetryAsync` untuk penulisan transaksional:

```
await ds.WithTransactionRetryAsync(async conn =>
{
    await using var cmd = conn.CreateCommand();

    cmd.CommandText = "UPDATE accounts SET balance = balance - 100 WHERE id = @from";
    cmd.Parameters.AddWithValue("from", fromId);
    await cmd.ExecuteNonQueryAsync();

    cmd.CommandText = "UPDATE accounts SET balance = balance + 100 WHERE id = @to";
    cmd.Parameters.Clear();
    cmd.Parameters.AddWithValue("to", toId);
    await cmd.ExecuteNonQueryAsync();
});
```

Untuk pernyataan DDL atau tunggal, gunakan`ExecWithRetryAsync`:

```
await ds.ExecWithRetryAsync("CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY,
 name TEXT)");
```

> **Important**
>
> `WithTransactionRetryAsync`mengelola`BEGIN`/`COMMIT`/`ROLLBACK`secara internal dan membuka koneksi baru untuk setiap upaya. Callback Anda seharusnya hanya berisi operasi database dan aman untuk dicoba lagi.

Opsi konfigurasi

Konektor juga menerima `postgres://` dan `postgresql://` menghubungkan string dengan `region` dan `profile` parameter kueri.

Bidang	Tipe	Default	Deskripsi
`Host`	`string`	(wajib)	Titik akhir cluster atau ID cluster 26-char
`Region`	`string?`	(terdeteksi secara otomatis)	AWS wilayah; diperlukan jika `Host` adalah ID cluster
`User`	`string`	`"admin"`	Pengguna basis data
`Database`	`string`	`"postgres"`	Nama basis data
`Port`	`int`	`5432`	Port basis data
`Profile`	`string?`	`null`	AWS nama profil untuk kredensyal
`CustomCredentialsProvider`	`AWSCredentials?`	`null`	Penyedia AWS kredensyal kustom
`TokenDurationSecs`	`int?`	`null`(SDK default, 900-an)	Durasi validitas token dalam hitungan detik
`OccMaxRetries`	`int?`	`null`(dinonaktifkan)	Percobaan ulang OCC maks default untuk metode coba lagi pada sumber data
`OrmPrefix`	`string?`	`null`	Awalan ORM diawali `application_name`
`LoggerFactory`	`ILoggerFactory?`	`null`	Pabrik logger untuk peringatan dan diagnostik coba lagi
`ConfigureConnectionString`	`Action<NpgsqlConnectionStringBuilder>?`	`null`	Callback untuk mengganti pengaturan pool atau mengatur properti string koneksi Npgsql tambahan. SSL dan `Enlist` merupakan invarian keamanan dan tidak dapat diganti.

Autentikasi

Konektor secara otomatis menangani otentikasi Aurora DSQL dengan menghasilkan token menggunakan kredensyal. AWS Jika Anda tidak menyediakan AWS wilayah, konektor menguraikannya dari nama host.

Untuk informasi lebih lanjut tentang otentikasi di Aurora DSQL, lihat. Otentikasi dan otorisasi untuk Aurora DSQL

Admin vs pengguna biasa

- Pengguna bernama “admin” secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi reguler
- Konektor menghasilkan token secara dinamis untuk setiap koneksi

Konektor Aurora DSQL untuk Karat SQLx

Konektor Aurora DSQL untuk Rust adalah konektor Rust yang dibangun di atasnya SQLxyang mengintegrasikan otentikasi IAM untuk menghubungkan aplikasi Rust ke cluster Amazon Aurora DSQL.

Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi sehingga Anda dapat fokus pada logika aplikasi Anda.

Tentang konektor

Konektor Aurora DSQL untuk Rust menambahkan lapisan otentikasi di atasnya SQLx yang menangani pembuatan token IAM, memungkinkan Anda untuk terhubung ke Aurora DSQL tanpa mengubah alur kerja yang ada. SQLx

Apa itu otentikasi Aurora DSQL?

Dalam Aurora DSQL, otentikasi melibatkan:

- Otentikasi IAM: Semua koneksi menggunakan otentikasi berbasis IAM dengan token terbatas waktu
- Generasi Token: Konektor menghasilkan token otentikasi menggunakan AWS kredensil, dan token ini memiliki masa pakai yang dapat dikonfigurasi

Konektor Aurora DSQL untuk Rust memahami persyaratan ini dan secara otomatis menghasilkan token otentikasi IAM saat membuat koneksi.

Fitur

- Otentikasi IAM otomatis - Menangani pembuatan dan penyegaran token Aurora DSQL
- Dibangun di atas SQLx - Membungkus driver PostgreSQL async populer untuk Rust
- Integrasi mulus - Bekerja dengan alur kerja yang ada SQLx
- Penggabungan koneksi - Dukungan kumpulan opt-in dengan penyegaran token latar belakang melalui fitur `pool`
- Deteksi otomatis wilayah - Ekstrak AWS wilayah dari nama host cluster Aurora DSQL
- AWS dukungan kredensil - Mendukung AWS profil dan rantai kredensi default
- Coba lagi OCC - Coba lagi kontrol konkurensi optimis dengan backoff eksponensial dan jitter

Contoh aplikasi

Untuk contoh lengkap, lihat contoh aplikasi di GitHub.

Panduan memulai cepat

Persyaratan

- Rust 1.80 atau lebih baru
- Memulai dengan Aurora DSQL
- AWS kredensil yang dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Penginstalan

Tambahkan ke`Cargo.toml`:

```
[dependencies]
aurora-dsql-sqlx-connector = "0.1.2"
```

Untuk sebagian besar aplikasi, aktifkan keduanya `pool` dan `occ` fitur:

```
[dependencies]
```

```
aurora-dsql-sqlx-connector = { version = "0.1.2", features = ["pool", "occ"] }
```

Bendera fitur

Fitur	Default	Deskripsi
`pool`	Tidak	SQLx pembantu kolam renang dengan penyegaran token latar belakang
`occ`	Tidak	Pembantu coba lagi OCC (,) `retry_on_occ` `is_occ_error`

Penggunaan

Koneksi kolam renang

```
use sqlx::Row;

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let pool = aurora_dsql_sqlx_connector::pool::connect(
        "postgres://admin@your-cluster.dsql.us-east-1.on.aws/postgres"
    ).await?;

    // Read
    let row = sqlx::query("SELECT 'Hello, DSQL!' as greeting")
        .fetch_one(&pool)
        .await?;
    let greeting: &str = row.get("greeting");
    println!("{}", greeting);

    // Write — you must wrap writes in a transaction
    let mut tx = pool.begin().await?;
    sqlx::query("INSERT INTO users (id, name) VALUES (gen_random_uuid(), $1)")
        .bind("Alice")
        .execute(&mut *tx)
        .await?;
    tx.commit().await?;

    pool.close().await;
    Ok(())
```

```
}
```

Koneksi tunggal

Untuk skrip sederhana atau saat Anda tidak memerlukan penyatuan koneksi:

```
use sqlx::Row;

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let mut conn = aurora_dsql_sqlx_connector::connection::connect(
        "postgres://admin@your-cluster.dsql.us-east-1.on.aws/postgres"
    ).await?;

    let row = sqlx::query("SELECT 1 as value")
        .fetch_one(&mut conn)
        .await?;
    let value: i32 = row.get("value");
    println!("Result: {}", value);

    Ok(())
}
```

Setiap panggilan untuk `connection::connect()` menghasilkan token IAM baru. Untuk operasi yang lebih lama dari durasi token, buat koneksi baru.

Penggunaan lanjutan

Konfigurasi host

Konektor mendukung titik akhir cluster penuh (wilayah terdeteksi secara otomatis) dan cluster IDs (wilayah diperlukan):

```
// Full endpoint (region auto-detected)
let opts = DsqlConnectOptions::from_connection_string(
    "postgres://admin@your-cluster.dsql.us-east-1.on.aws/postgres"
)?;

// Cluster ID (region required)
let opts = DsqlConnectOptions::from_connection_string(
    "postgres://admin@your-cluster-id/postgres?region=us-east-1"
)?;
```

AWS profil

Tentukan AWS profil untuk kredensil:

```
let pool = aurora_dsql_sqlx_connector::pool::connect(
    "postgres://admin@your-cluster.dsql.us-east-1.on.aws/postgres?profile=production"
).await?;
```

Format string koneksi

Konektor mendukung format string koneksi PostgreSQL:

```
postgres://[user@]host[:port]/[database][?param=value&...]
postgresql://[user@]host[:port]/[database][?param=value&...]
```

Parameter kueri yang didukung: `region`,`profile`,`tokenDurationSecs`,,`ormPrefix`.

Konfigurasi kolam renang

Untuk pengaturan kolam khusus, teruskan `PgPoolOptions` ke`connect_with()`:

```
use aurora_dsql_sqlx_connector::DsqlConnectOptions;
use sqlx::postgres::PgPoolOptions;

let config = DsqlConnectOptions::from_connection_string(
    "postgres://admin@your-cluster.dsql.us-east-1.on.aws/postgres"
)?;

let pool = aurora_dsql_sqlx_connector::pool::connect_with(
    &config,
    PgPoolOptions::new().max_connections(20),
).await?;
```

Konfigurasi terprogram

Gunakan `DsqlConnectOptionsBuilder` untuk konfigurasi terprogram:

```
use aurora_dsql_sqlx_connector::{DsqlConnectOptionsBuilder, Region};
use sqlx::postgres::PgConnectOptions;
```

```
let pg = PgConnectOptions::new()
    .host("your-cluster.dsql.us-east-1.on.aws")
    .username("admin")
    .database("postgres");

let opts = DsqlConnectOptionsBuilder::default()
    .pg_connect_options(pg)
    .region(Some(Region::new("us-east-1")))
    .build()?;

let mut conn = aurora_dsql_sqlx_connector::connection::connect_with(&opts).await?;
```

Coba lagi OCC

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC). Ketika dua transaksi memodifikasi data yang sama, yang pertama melakukan kemenangan dan yang kedua menerima kesalahan OCC.

Coba lagi OCC adalah opt-in. Aktifkan `occ` fitur dan gunakan `retry_on_occ` untuk mengaktifkan percobaan ulang otomatis dengan backoff dan jitter eksponensial:

```
use aurora_dsql_sqlx_connector::{retry_on_occ, OCCRetryConfig};

let config = OCCRetryConfig::default(); // max_attempts: 3, exponential backoff

retry_on_occ(&config, || async {
    let mut tx = pool.begin().await?;

    sqlx::query("UPDATE accounts SET balance = balance - 100 WHERE id = $1")
        .bind(account_id)
        .execute(&mut *tx)
        .await?;

    tx.commit().await?;
    Ok(())
}).await?;
```

> **Warning**
>
> `retry_on_occ`mengeksekusi kembali seluruh penutupan pada konflik OCC, sehingga penutupan harus hanya berisi operasi database dan aman untuk dicoba lagi.

Opsi konfigurasi

Bidang	Tipe	Default	Deskripsi
`host`	`String`	(wajib)	Titik akhir cluster atau ID cluster
`region`	`Option<Region>`	(terdeteksi otomatis)	AWS wilayah; diperlukan jika host adalah ID cluster
`user`	`String`	`"admin"`	Pengguna basis data
`database`	`String`	`"postgres"`	Nama database
`port`	`u16`	`5432`	Port basis data
`profile`	`Option<String>`	`None`	AWS nama profil untuk kredensil
`tokenDurationSecs`	`u64`	`900`(15 menit)	Durasi validitas token dalam hitungan detik
`ormPrefix`	`Option<String>`	`None`	Awalan ORM untuk application_name (misalnya, menghasilkan) `"diesel"` `"diesel:aurora-dsql-rust-sqlx/{version}"`

Autentikasi

Konektor secara otomatis menangani otentikasi Aurora DSQL dengan menghasilkan token menggunakan kredensil. AWS Jika Anda tidak memberikan AWS wilayah, konektor menguraikannya dari nama host.

Untuk informasi lebih lanjut tentang otentikasi di Aurora DSQL, lihat. Otentikasi dan otorisasi untuk Aurora DSQL

Pembuatan token

- Kumpulan koneksi: Tugas latar belakang menyegarkan token pada 80% dari durasi token. Panggil `pool.close().await` untuk menghentikan tugas penyegaran dan melepaskan sumber daya kumpulan.
- Koneksi tunggal: Konektor menghasilkan token baru pada waktu koneksi.
- Pembuatan Token adalah operasi presigning SiGv4 lokal dengan biaya yang dapat diabaikan.

Admin vs pengguna biasa

- Pengguna bernama “admin” secara otomatis menggunakan token otentikasi admin
- Semua pengguna lain menggunakan token otentikasi reguler
- Konektor menghasilkan token secara dinamis untuk setiap koneksi

Mengakses Aurora DSQL dengan klien PostgreSQL-compatible

Aurora DSQL menggunakan protokol kawat PostgreSQL. Anda dapat terhubung ke PostgreSQL menggunakan berbagai alat dan klien, AWS CloudShell seperti, psql, DBeaver, dan. DataGrip Tabel berikut merangkum bagaimana Aurora DSQL memetakan parameter koneksi PostgreSQL umum:

PostgreSQL	Aurora DSQL	Catatan
Peran (juga dikenal sebagai Pengguna atau Grup)	Peran Database	Aurora DSQL menciptakan peran untuk Anda bernama. `admin` Saat membuat peran basis data kustom, Anda harus menggunakan peran tersebut untuk mengaitkannya dengan `admin` peran IAM untuk mengautentikasi saat menghubungkan ke klaster Anda. Untuk informasi selengkapnya, lihat Menggunakan peran database dan otentikasi IAM.
Host (juga dikenal sebagai hostname atau hostspec)	Titik Akhir klaster	Aurora DSQL Kluster wilayah tunggal menyediakan satu titik akhir terkelola dan secara otomatis mengarahkan lalu lintas jika tidak tersedia di dalam Wilayah.

PostgreSQL	Aurora DSQL	Catatan
Port	N/A — gunakan default `5432`	Ini adalah PostgreSQL default.
Database (dbname)	menggunakan `postgres`	Aurora DSQL membuat database ini untuk Anda saat Anda membuat cluster.
Modus SSL	SSL selalu diaktifkan di sisi server	Di Aurora DSQL, Aurora DSQL mendukung Mode SSL. `require` Koneksi tanpa SSL ditolak oleh Aurora DSQL.
Kata Sandi	Token Otentikasi	Aurora DSQL membutuhkan token otentikasi sementara alih-alih kata sandi berumur panjang. Untuk mempelajari selengkapnya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

Saat menghubungkan, Aurora DSQL memerlukan token otentikasi IAM yang ditandatangani sebagai pengganti kata sandi tradisional. Token sementara ini dihasilkan menggunakan AWS Signature Version 4 dan hanya digunakan selama pembuatan koneksi. Setelah terhubung, sesi tetap aktif sampai berakhir atau klien terputus.

Jika Anda mencoba membuka sesi baru dengan token kedaluwarsa, permintaan koneksi gagal dan token baru harus dibuat. Untuk informasi selengkapnya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

Parameter sesi yang didukung

Aurora DSQL secara otomatis mengoptimalkan penyimpanan, manajemen koneksi, dan pemrosesan transaksi, sehingga mendukung serangkaian parameter sesi yang terfokus. Aurora DSQL menerima parameter yang tercantum dalam tabel berikut. Parameter di luar set ini mengembalikan kesalahan berikut:

```
ERROR: setting configuration parameter "parameter_name" not supported
```

Aurora DSQL mendukung parameter sesi PostgreSQL berikut. Kecuali dinyatakan lain, Anda dapat mengatur ini menggunakan `SETSET SESSION`,, atau sebagai opsi string koneksi. Anda juga dapat

menggunakan `SET LOCAL` untuk mengatur salah satu parameter ini selama satu transaksi. Gunakan `RESET` untuk mengembalikan parameter ke nilai defaultnya.

Parameter sesi yang didukung

Parameter	Kategori	Deskripsi
`application_name`	Identifikasi klien	Menetapkan nama aplikasi yang dilaporkan dalam metadata koneksi. Berguna untuk mengidentifikasi koneksi dalam pemantauan.
`client_encoding`	Lokalisasi	Mengatur pengkodean karakter sisi klien. Database menggunakan UTF-8 secara internal.
`datestyle`	Pemformatan keluaran	Menetapkan format tampilan untuk nilai tanggal dan waktu (misalnya,`ISO, MDY`).
`extra_float_digits`	Pemformatan keluaran	Menetapkan jumlah digit yang ditampilkan untuk nilai floating-point. Beberapa driver, seperti JDBC, menggunakan parameter ini.
`intervalstyle`	Pemformatan keluaran	Menetapkan format tampilan untuk nilai interval.
`timezone`	Lokalisasi	Mengatur zona waktu untuk sesi tersebut. Aurora DSQL menyimpan semua tanggal dan waktu yang sadar zona waktu secara internal di UTC. Parameter ini mengontrol bagaimana Aurora DSQL menampilkan nilai ke klien.
`search_path`	Resolusi skema	Menetapkan urutan pencarian skema untuk nama objek yang tidak memenuhi syarat.
`enable_bitmapscan`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan perencana kueri jenis paket pemindaian bitmap.
`enable_hashjoin`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan tipe paket hash-join oleh perencana kueri.

Parameter	Kategori	Deskripsi
`enable_indexonlyscan`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan perencana kueri jenis paket pemindaian hanya indeks.
`enable_indexscan`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan tipe paket pemindaian indeks oleh perencana kueri.
`enable_material`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan materialisasi perencana kueri.
`enable_mergejoin`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan tipe paket gabungan gabungan oleh perencana kueri.
`enable_nestloop`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan paket gabungan loop bersarang oleh perencana kueri.
`enable_seqscan`	Perencana kueri	Mengaktifkan atau menonaktifkan penggunaan tipe rencana pemindaian sekuensial oleh perencana kueri.
`disable_sync_create_index`	Aurora DSQL-specific	Mengontrol apakah `CREATE INDEX` berjalan secara asinkron. Defaultnya adalahon, artinya Aurora DSQL membuat indeks secara asinkron. Setel `off` untuk membuat indeks secara sinkron pada tabel kosong.
`role`	Identitas sesi	Menetapkan peran saat ini. Di Aurora DSQL, Anda hanya dapat mengatur parameter ini menggunakan `SET LOCAL` dalam blok transaksi.

Aurora DSQL secara otomatis mengelola aspek-aspek berikut dari database Anda, sehingga Anda tidak perlu mengkonfigurasi parameter sesi PostgreSQL yang sesuai:

- Parameter koneksi dan jaringan — Aurora DSQL mengelola siklus hidup koneksi dan pengaturan TCP secara internal, termasuk,, dan. `tcp_keepalives_idle` `tcp_keepalives_interval` `tcp_keepalives_count`
- Parameter memori dan sumber daya — Aurora DSQL secara otomatis mengelola alokasi memori, termasuk dan. `work_mem` `shared_buffers`
- Parameter batas waktu — Aurora DSQL memberlakukan batas durasi transaksinya sendiri, termasuk,, dan. `statement_timeout` `lock_timeout` `idle_in_transaction_session_timeout`
- Parameter replikasi dan WAL — Aurora DSQL menangani replikasi secara otomatis melalui arsitektur Multi-region bawaannya, termasuk dan. `wal_level` `synchronous_commit`
- Parameter logging — Aurora DSQL mengelola logging secara internal, termasuk dan. `log_statement` `client_min_messages`
- Parameter transaksi — Aurora DSQL menggunakan tingkat `REPEATABLE READ` isolasi tetap untuk semua transaksi. Jika pengemudi Anda perlu menentukan tingkat isolasi, gunakan`BEGIN ISOLATION LEVEL REPEATABLE READ`.

Akses Aurora DSQL menggunakan klien SQL

Aurora DSQL mendukung beberapa PostgreSQL-compatible klien untuk menghubungkan ke cluster Anda. Bagian berikut menjelaskan cara menghubungkan menggunakan PostgreSQL AWS CloudShell dengan atau baris perintah lokal Anda, serta GUI-based alat seperti DBeaver dan. JetBrains DataGrip Setiap klien memerlukan token otentikasi yang valid seperti yang dijelaskan di bagian sebelumnya.

Topik

- Gunakan DBeaver untuk mengakses Aurora DSQL
- Gunakan JetBrains DataGrip untuk mengakses Aurora DSQL
- Gunakan terminal interaktif PostgreSQL (psql) untuk mengakses Aurora DSQL
- Gunakan driver Aurora DSQL untuk SQLTools
- Pemecahan masalah

Gunakan DBeaver untuk mengakses Aurora DSQL

DBeaver adalah klien SQL universal yang dapat digunakan untuk mengelola database apa pun yang memiliki driver JDBC. Ini banyak digunakan di kalangan pengembang dan administrator database karena kemampuan tampilan, pengeditan, dan manajemen datanya yang kuat. Menggunakan opsi konektivitas cloud DBeaver, Anda dapat menghubungkan DBeaver ke Aurora DSQL secara asli.

DBeaver Pro

Produk DBeaver PRO menawarkan integrasi asli dengan Aurora DSQL pada versi 25.3. Ikuti instruksi dari Dokumentasi DBeaver untuk terhubung ke cluster Aurora DSQL Anda.

Edisi Komunitas DBeaver

DBeaver Community Edition adalah versi gratis dan open-source. Kunjungi halaman unduhan untuk petunjuk penginstalan. Untuk terhubung ke DSQL dari DBeaver Community Edition, Anda perlu menginstal Plugin Aurora DSQL untuk DBeaver.

Plugin Aurora DSQL untuk DBeaver dibangun di atas Konektor Aurora DSQL untuk JDBC dan memungkinkan otentikasi IAM ke cluster Aurora DSQL. Ini mudah dipasang melalui UI DBeaver dan menghilangkan kebutuhan untuk menulis kode pembuatan token atau secara manual memasok token IAM yang valid, menyederhanakan otentikasi sambil menghilangkan risiko keamanan yang terkait dengan kata sandi tradisional yang dibuat pengguna.

Fitur

- Dukungan Autentikasi IAM: Sambungkan ke cluster Aurora DSQL menggunakan kredensil AWS IAM untuk otentikasi aman dan bebas kata sandi
- Manajemen Driver Otomatis: Menginstal dan mengonfigurasi Konektor Aurora DSQL dengan mulus untuk JDBC
- Opsi Koneksi Fleksibel: Pilih antara Host-based atau konfigurasi koneksi JDBC URL-based

Plugin Aurora DSQL untuk Instalasi DBeaver

1. Dengan DBeaver dibuka, Buka menu Drop down Bantuan → Instal Perangkat Lunak Baru
2. Klik Tambah untuk menambahkan repositori baru
3. Masukkan:
   - Nama: `Aurora DSQL Plugin`
   - Lokasi: `https://awslabs.github.io/aurora-dsql-dbeaver-plugin/update-site/`

4. Periksa Konektor Aurora DSQL untuk JDBC
5. Klik Berikutnya, terima lisensi, dan selesaikan instalasi
6. Mulai ulang DBeaver saat diminta

Buat Koneksi Aurora DSQL

1. Klik Koneksi Database Baru
2. Pilih Aurora DSQL
3. Di bawah Server, pilih salah satu dari berikut ini untuk Connect by setting
   - Tuan rumah
     - untuk mengaktifkan input teks antarmuka pengguna untuk bidang berikut:
       - Titik akhir: Titik Akhir Cluster DSQL
       - Nama Pengguna: Nama pengguna DSQL (mis. Admin)
       - Profil AWS: misalnya default - Profil standar yang digunakan saat tidak ada profil tertentu yang ditentukan
       - AWS Region (Opsional): harus cocok dengan wilayah tempat kluster DSQL Anda berada, jika tidak, otentikasi akan gagal
   - URL
     - URL JDBC dalam format ini:

       ```
       jdbc:aws-dsql:postgresql://{cluster_endpoint}/{database}?
       user=admin&profile=default&region=us-east-1
       ```

     - Catatan: Dalam mode ini, hanya input URL yang diaktifkan. Untuk menambahkan parameter ke string koneksi JDBC, gunakan format parameter kueri URL yang dimulai dengan? sebagai parameter pertama dan menambahkan & untuk parameter berikutnya.
4. Klik Uji Koneksi untuk memverifikasi koneksi Aurora DSQL berfungsi
5. Klik Selesai

Pemecahan masalah

Masalah Windows Trust Store

Pengguna Windows mungkin mengalami masalah saat mengunduh Konektor Aurora DSQL untuk driver JDBC dari Maven Central.

Penyebab: Windows Trust Store mungkin tidak menyertakan sertifikat yang diperlukan untuk mengakses repositori Maven Central.

Solusi:

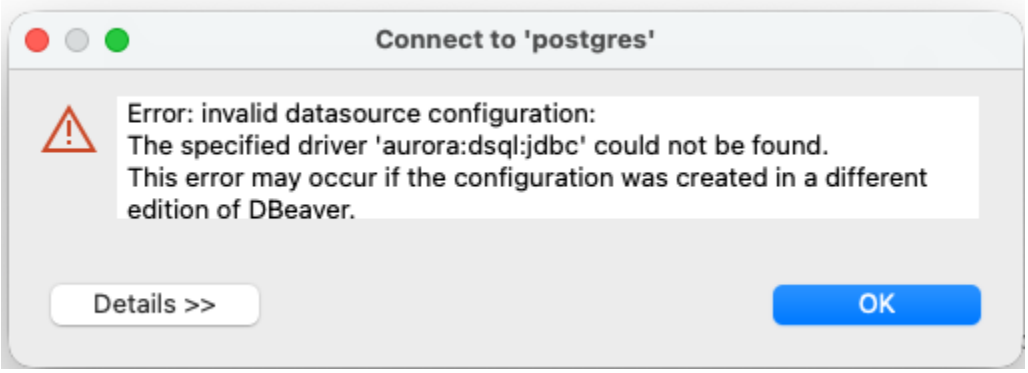
1. Jalankan DBeaver sebagai “Administrator”
2. Hapus centang pada pengaturan ini - Windows > Preferensi > Koneksi > “Gunakan toko Windows Trust”

Kesalahan Pengemudi Hilang

Jika Anda melihat ikon driver yang hilang atau kesalahan koneksi, Aurora DSQL (Plugin Komunitas) mungkin tidak diinstal dalam versi DBeaver Anda saat ini. Lihat di bawah ini beberapa contoh kesalahan dan cara memperbaikinya:

- Membuat koneksi baru dengan driver yang hilang:

- Mencoba terhubung tanpa driver:



Penyebab: Ketika beberapa versi DBeaver diinstal, pengaturan koneksi dibagikan tetapi driver diinstal per aplikasi.

Solusi: Instal ulang Aurora DSQL (plugin Komunitas) dengan mengikuti langkah-langkah instalasi di atas.

> **Important**
>
> Fitur administratif yang disediakan oleh DBeaver untuk database PostgreSQL (seperti Session Manager dan Lock Manager) tidak berlaku untuk database Aurora DSQL karena arsitekturnya yang unik. Meskipun dapat diakses, layar ini tidak memberikan informasi yang dapat dipercaya tentang kesehatan atau status database.

Gunakan JetBrains DataGrip untuk mengakses Aurora DSQL

JetBrains DataGrip adalah IDE lintas platform untuk bekerja dengan SQL dan database, termasuk PostgreSQL. DataGrip termasuk GUI yang kuat dengan editor SQL cerdas. Untuk mengunduh DataGrip, buka halaman unduhan di situs JetBrainsweb.

Untuk mengatur koneksi Aurora DSQL baru di JetBrains DataGrip

1. Pilih Sumber Data Baru dan pilih PostgreSQL.
2. Di tab Sumber Data/Umum, masukkan informasi berikut:
   - Host — Gunakan endpoint cluster Anda.

     Port - Aurora DSQL menggunakan PostgreSQL default: `5432`

     Database - Aurora DSQL menggunakan PostgreSQL default dari `postgres`

     Otentikasi — Pilih`User & Password` .

     Nama Pengguna — Masukkan`admin`.

     Kata sandi — Buat token dan tempelkan ke bidang ini.

     URL - Jangan ubah bidang ini. Ini akan diisi secara otomatis berdasarkan bidang lain.
3. Kata sandi — Berikan ini dengan membuat token otentikasi. Salin output yang dihasilkan dari generator token dan tempel ke bidang kata sandi.

> **Note**
>
> Anda harus mengatur mode SSL di koneksi klien. Aurora DSQL mendukung. `PGSSLMODE=require and PGSSLMODE=verify-full` Aurora DSQL memberlakukan komunikasi SSL di sisi server dan menolak koneksi non-SSL. Untuk `verify-full` opsi,

Anda perlu menginstal sertifikat SSL secara lokal. Untuk informasi selengkapnya lihat sertifikat SSL/TLS.

4. Anda harus terhubung ke cluster Anda dan dapat mulai menjalankan pernyataan SQL:

> **Important**
>
> Beberapa tampilan yang disediakan oleh DataGrip database PostgreSQL (seperti Sessions) tidak berlaku untuk database Aurora DSQL karena arsitekturnya yang unik. Meskipun dapat diakses, layar ini tidak memberikan informasi yang dapat dipercaya tentang sesi aktual yang terhubung ke database.

Gunakan terminal interaktif PostgreSQL (psql) untuk mengakses Aurora DSQL

Gunakan AWS CloudShell untuk mengakses Aurora DSQL dengan terminal interaktif PostgreSQL (psql)

Gunakan prosedur berikut untuk mengakses Aurora DSQL dengan terminal interaktif PostgreSQL dari. AWS CloudShell Untuk informasi lebih lanjut, lihat Apa itu AWS CloudShell.

Untuk terhubung menggunakan AWS CloudShell

1. Masuk ke konsol Aurora DSQL.
2. Pilih cluster yang ingin Anda buka CloudShell. Jika Anda belum membuat cluster, ikuti langkah-langkah di Langkah 1: Buat cluster Aurora DSQL Single-region atauBuat klaster Multi-wilayah.
3. Pilih Connect with Query Editor dan kemudian pilih Connect with CloudShell.
4. Pilih apakah Anda ingin terhubung sebagai admin atau dengan peran basis data kustom.
5. Pilih Luncurkan CloudShell dan pilih Jalankan di CloudShell dialog berikut.

Gunakan CLI lokal untuk mengakses Aurora DSQL dengan terminal interaktif PostgreSQL (psql)

Gunakan`psql`, front-end berbasis terminal untuk utilitas PostgreSQL, untuk memasukkan kueri secara interaktif, menerbitkannya ke PostgreSQL, dan melihat hasil kueri.

> **Note**
>
> Untuk meningkatkan waktu respons kueri, gunakan klien PostgreSQL versi 17. Jika Anda menggunakan CLI di lingkungan yang berbeda, pastikan Anda secara manual mengatur Python versi 3.8+ dan psql versi 14+.

Unduh installer sistem operasi Anda dari halaman Unduhan PostgreSQL. Untuk informasi selengkapnya*psql*, lihat Aplikasi Klien PostgreSQL di situs web PostgreSQL.

Jika Anda sudah AWS CLI menginstal, gunakan contoh berikut untuk terhubung ke cluster Anda.

```
# Aurora DSQL requires a valid IAM token as the password when connecting.
# Aurora DSQL provides tools for this and here we're using Python.
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token \
  --region us-east-1 \
  --expires-in 3600 \
  --hostname your_cluster_endpoint)

# Aurora DSQL requires SSL and will reject your connection without it.
export PGSSLMODE=require

# Connect with psql, which automatically uses the values set in PGPASSWORD and
 PGSSLMODE.
# Quiet mode suppresses unnecessary warnings and chatty responses but still outputs
 errors.
psql --quiet \
  --username admin \
  --dbname postgres \
  --host your_cluster_endpoint
```

Gunakan driver Aurora DSQL untuk SQLTools

Driver Aurora DSQL untuk SQLTools adalah ekstensi Visual Studio Code untuk Amazon Aurora DSQL yang terintegrasi dengan SQLTools. Ini memungkinkan pengembang untuk terhubung ke dan menanyakan database Aurora DSQL langsung dari VS Code. Driver tersedia untuk instalasi dari Visual Studio Marketplace dan Open VSX Registry. Kiro, Kursor dan VSCode-based IDE lainnya dapat menggunakan Open VSX Registry untuk menginstal driver mengikuti prosedur instalasi standar yang dijelaskan di halaman ini.

Fitur

- Otentikasi IAM Otomatis
- Operasi basis data standar seperti skema penelusuran, tabel, dan mengeksekusi kueri SQL.

Penginstalan

1. Buka tampilan Ekstensi.
2. Cari “Driver Aurora DSQL untuk SQLTools”.
3. Klik “Instal”.

Catatan:

Ekstensi SQLTools akan diinstal secara otomatis jika belum ada.

Autentikasi

Di Aurora DSQL semua koneksi menggunakan IAM-based otentikasi dengan token terbatas waktu. Driver secara otomatis menangani otentikasi Aurora DSQL menggunakan Konektor Aurora DSQL untuk node-postgres.

Untuk informasi selengkapnya tentang otentikasi di Aurora DSQL, lihat panduan pengguna.

Buat Koneksi Aurora DSQL

Prasyarat

- Kredensi AWS dikonfigurasi (melalui AWS CLI, variabel lingkungan, atau peran IAM)

Langkah-langkah

1. Klik ikon SQLTools di sidebar kiri.
2. Di panel SQLTools, arahkan kursor ke CONNECTIONS dan klik ikon Add New Connection.
3. Di tab Pengaturan SQLTools pilih Driver Aurora DSQL dari daftar.
4. Isi parameter koneksi.
   - Wilayah AWS
     - Opsional - wilayah akan diurai dari titik akhir cluster Aurora DSQL.
     - Diperlukan ketika hanya ID cluster ditentukan dalam bidang DSQL Cluster.

- Profil AWS
  - Digunakan untuk pembuatan token.
  - Menggunakan profil default jika tidak ditentukan.

5. Klik “tombol Uji Koneksi” untuk menguji koneksi.
6. Klik Simpan Koneksi.

Pemecahan masalah

Kedaluwarsa kredensi otentikasi untuk Klien SQL

Sesi yang ditetapkan tetap diautentikasi selama maksimal 1 jam atau sampai pemutusan eksplisit atau batas waktu sisi klien terjadi. Jika koneksi baru perlu dibuat, token otentikasi baru harus dibuat dan disediakan di bidang Kata Sandi koneksi. Mencoba membuka sesi baru (misalnya, untuk membuat daftar tabel baru, atau membuka konsol SQL baru) memaksa upaya otentikasi baru. Jika token otentikasi yang dikonfigurasi dalam pengaturan Koneksi tidak lagi valid, sesi baru itu akan gagal dan semua sesi yang dibuka sebelumnya akan menjadi tidak valid. Ingatlah hal ini ketika memilih durasi token otentikasi IAM Anda dengan `expires-in` opsi, yang dapat diatur ke 15 menit secara default dan dapat diatur ke nilai maksimum tujuh hari.

Selain itu, lihat bagian Pemecahan Masalah dari dokumentasi Aurora DSQL.

Alat konektivitas kluster Amazon Aurora DSQL

Aurora DSQL kompatibel dengan banyak driver database pihak ketiga dan pustaka ORM. AWS menyediakan dua jenis alat untuk menyederhanakan bekerja dengan Aurora DSQL:

- Konektor — Plugin otentikasi yang memperluas driver database untuk menangani pembuatan token IAM secara otomatis. Gunakan konektor saat bekerja langsung dengan driver database.
- Adaptor dan dialek — Ekstensi untuk kerangka kerja ORM tertentu yang menyediakan otentikasi IAM dan peningkatan kompatibilitas Aurora DSQL. Gunakan adaptor saat bekerja dengan kerangka kerja ORM yang didukung.

Adaptor dan dialek Aurora DSQL

Tabel berikut menunjukkan adaptor dan dialek yang tersedia untuk Aurora DSQL.

Bahasa pemrograman	ORM/Framework	Tautan repositori
Java	Hibernasi	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy
Python	Kura-kura ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm

Sampel driver database

Tabel berikut menunjukkan kode contoh untuk menghubungkan ke Aurora DSQL menggunakan driver database pihak ketiga.

Bahasa pemrograman	Driver	Contoh tautan repositori
C++	libpq	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/libpq
C# (.NET)	Npgsql	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/npgsql
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx

Bahasa pemrograman	Driver	Contoh tautan repositori
Java	HikariCP+PgJdBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	node-postgres (AWS Lambda)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/lambda
JavaScript	simpul-postgres	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/node-postgres
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
Python	asyncpg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/asyncpg
Python	Psycopg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycopg2	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg2
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Karat	SqlX	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx

ORM dan sampel kerangka kerja

Tabel berikut menunjukkan kode contoh untuk menggunakan pustaka dan kerangka kerja ORM pihak ketiga dengan Aurora DSQL.

Bahasa pemrograman	ORM/Framework	Contoh tautan repositori
Java	Hibernasi	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate/examples/pet-clinic-app
Java	Liquibase	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/liquibase
Java	Sepatu Bot Musim Semi	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/spring_boot
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django/examples/pet-clinic-app
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy/examples/pet-clinic-app
Python	Kura-kura ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm/example
Ruby	Rel	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails

Bahasa pemrograman	ORM/Framework	Contoh tautan repositori
TypeScript	Prisma	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/prisma-multi-region
TypeScript	Sequelize	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	TypeORM	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/type-orm

Memuat data ke Aurora DSQL

Baik Anda bermigrasi dari database yang ada, mengimpor file dari Amazon Simple Storage Service, atau memuat data dari sistem lokal Anda, Aurora DSQL menyediakan beberapa pendekatan untuk memasukkan data Anda. Bagian ini mencakup alat dan teknik yang direkomendasikan untuk memuat data dari semua ukuran, dari gigabyte hingga ratusan terabyte.

Memilih pendekatan pemuatan

Aurora DSQL mendukung perintah pemuatan data PostgreSQL standar, tetapi memuat data secara efisien dalam skala memerlukan penanganan paralelisasi, manajemen koneksi, dan pemulihan kesalahan. Tabel berikut merangkum opsi Anda:

Pendekatan	Terbaik untuk	Pertimbangan-pertimbangan
Aurora DSQL Loader - Utilitas open source yang memudahkan untuk memparalelkan	Sebagian besar skenario pemuatan data, terutama migrasi dan impor massal	Menangani paralelisasi, penyatuan koneksi, resolusi konflik, dan otentikasi IAM secara otomatis. Tersedia sebagai kode sumber atau biner.

Pendekatan	Terbaik untuk	Pertimbangan-pertimbangan
sisipan saat menggunakan Aurora DSQL		
PostgreSQL - Perintah `\copy` meta-sisi klien `psql`	Beban sederhana saat Anda sudah terhubung melalui `psql`	Membaca file pada klien dan mengalirkan data melalui koneksi; Anda mengelola paralelisasi sendiri
`INSERT`transaksi - Standar `SQL DML`	Kumpulan data kecil atau sisipan berbasis aplikasi	Pendekatan paling sederhana tetapi paling lambat untuk data massal

Untuk sebagian besar tugas pemuatan data, gunakan Aurora DSQL Loader. Ini menangani kompleksitas operasional pemuatan data ke dalam database terdistribusi, termasuk eksekusi paralel di beberapa koneksi dan coba ulang otomatis operasi yang gagal.

Aurora DSQL Loader

Aurora DSQL Loader adalah utilitas baris perintah open-source yang dirancang untuk memuat data secara efisien ke dalam cluster Aurora DSQL. Ini mengelola penyatuan koneksi, memparalelkan transfer data di beberapa pekerja, dan menangani konflik dan mencoba ulang secara otomatis.

Fitur utama

Aurora DSQL Loader menyediakan kemampuan berikut:

Pemuatan paralel

Thread pekerja yang dapat dikonfigurasi memungkinkan pemuatan data bersamaan di beberapa koneksi untuk meningkatkan kinerja.

Pengumpulan koneksi

Mengelola kumpulan koneksi ke cluster Aurora DSQL Anda, menangani autentikasi IAM dan siklus hidup koneksi secara otomatis.

Dukungan beberapa format file

Mendukung CSV (nilai yang dipisahkan koma), TSV (nilai yang dipisahkan tab), dan format kolom Apache Parquet. Loader secara otomatis mendeteksi format file berdasarkan ekstensi URI sumber.

Inferensi skema otomatis

Saat digunakan dengan `--if-not-exists` bendera, loader dapat secara otomatis membuat tabel dengan jenis kolom yang sesuai berdasarkan data.

Penanganan konflik

Jika tabel target Anda memiliki batasan unik, konfigurasikan cara loader menangani konflik menggunakan `--on-conflict` opsi: lewati duplikat, tambahkan catatan, atau kembalikan kesalahan.

Toleransi kesalahan

Percobaan ulang otomatis dan kemampuan dimulainya kembali pekerjaan memastikan bahwa beban yang terputus dapat berlanjut dari titik penghentiannya daripada memulai kembali sepenuhnya.

Sumber lokal dan S3

Muat data dari jalur sistem file lokal atau langsung dari bucket Amazon S3 menggunakan S3. URIs

Prasyarat

Sebelum menggunakan Aurora DSQL Loader, pastikan Anda memiliki yang berikut:

- Cluster Aurora DSQL aktif dengan titik akhir yang valid.
- AWS kredensil yang dikonfigurasi melalui peran AWS CLI (aws configure), AWS Single Sign-On (aws sso login), atau IAM.
- Izin IAM: `dsql:DbConnectAdmin` atau `dsql:DbConnect` di cluster Aurora DSQL Anda.
- Untuk sumber S3, izin yang sesuai untuk membaca dari bucket sumber.

Penginstalan

Unduh rilis terbaru dari halaman GitHub rilis. Binari pra-bangun tersedia untuk platform umum. Untuk petunjuk tentang membangun dari sumber, lihat repositori Aurora DSQL Loader.

Contoh penggunaan

Contoh berikut menunjukkan kasus penggunaan umum untuk Aurora DSQL Loader.

Example Memuat file CSV lokal

Contoh ini memuat file CSV dari sistem file lokal Anda ke dalam tabel yang ada:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table
```

Example Memuat data dari Amazon S3

Contoh ini memuat file Parket dari bucket Amazon S3:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/data.parquet \
  --table my_table
```

Example Pembuatan tabel otomatis

Contoh ini membuat tabel baru secara otomatis berdasarkan skema data:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --if-not-exists
```

Example Memvalidasi sebelum memuat

Contoh ini memvalidasi konfigurasi Anda tanpa benar-benar memuat data:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
```

```
  --dry-run
```

Example Melanjutkan beban yang terputus

Jika operasi pemuatan terganggu, Anda dapat melanjutkannya menggunakan ID pekerjaan dari proses sebelumnya:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --resume-job-id job-id \
  --manifest-dir ./loader-state
```

> **Note**
>
> Saat melanjutkan, loader melewatkan sebagian besar pekerjaan yang sudah selesai tetapi dapat mencoba kembali beberapa catatan. Jika tabel target Anda memiliki batasan unik, gunakan `--on-conflict` opsi untuk menangani duplikat — misalnya, `DO NOTHING` untuk melewatinya atau untuk meningkatkan. `DO UPDATE`

Opsi baris perintah

Aurora DSQL Loader mendukung opsi baris perintah berikut:

`--endpoint`

(Wajib) Titik akhir cluster Aurora DSQL. Contoh: *`cluster-id`*`.dsql.`*`region`*`.on.aws`

`--source-uri`

(Wajib) Jalur ke file data. Bisa berupa jalur file lokal atau URI S3 (misalnya,`s3://`*`bucket-name`*`/`*`file.parquet`*).

`--table`

(Wajib) Nama tabel target dalam database Aurora DSQL Anda.

`--if-not-exists`

(Opsional) Secara otomatis membuat tabel target jika tidak ada. Loader menyimpulkan skema dari data.

`--dry-run`

(Opsional) Validasi konfigurasi dan data tanpa benar-benar memuatnya ke dalam database.

`--resume-job-id`

(Opsional) Lanjutkan operasi beban yang sebelumnya terputus menggunakan ID pekerjaan yang ditentukan.

`--manifest-dir`

(Opsional) Direktori untuk menyimpan status pekerjaan dan manifes, digunakan untuk dimulainya kembali pekerjaan.

`--on-conflict`

(Opsional) Menentukan cara menangani konflik saat memasukkan baris yang melanggar kendala unik pada tabel target. Nilai yang valid adalah `error` (mengembalikan kesalahan), `do-nothing` (lewati baris duplikat), atau `do-update` (perbarui baris yang ada dengan nilai baru).

Untuk daftar lengkap opsi dan parameter konfigurasi tambahan, jalankan:

```
aurora-dsql-loader load --help
```

Praktik terbaik

- Gunakan dry-run untuk validasi — Selalu uji konfigurasi pemuatan Anda `--dry-run` sebelum memuat data ke dalam tabel produksi.
- Tentukan batasan unik untuk dimulainya kembali - Jika Anda perlu melanjutkan beban yang terputus, tentukan batasan unik pada tabel target Anda dan gunakan opsi untuk menangani catatan yang sudah dimuat. `--on-conflict`
- Gunakan Parket untuk kumpulan data besar — Format kolom Parket biasanya memberikan kompresi yang lebih baik dan pemuatan yang lebih cepat untuk kumpulan data besar dibandingkan dengan CSV atau TSV.
- Pertahankan direktori manifes — Simpan direktori manifes untuk memuat pekerjaan hingga Anda mengonfirmasi pemuatan berhasil diselesaikan, memungkinkan dimulainya kembali jika diperlukan.
- Pra-buat tabel bila memungkinkan - Tentukan tabel target dengan tipe data kolom eksplisit dan kunci utama sebelum memuat data. Skema yang dibuat sebelumnya memberi Anda kontrol atas presisi dan pengindeksan tipe, yang biasanya menghasilkan kinerja kueri yang lebih baik dibandingkan dengan skema yang disimpulkan secara otomatis.

Pemecahan masalah

Kesalahan autentikasi

Verifikasi AWS kredensil Anda dikonfigurasi dengan benar dan identitas IAM Anda memiliki persyaratan `dsql:DbConnect` atau `dsql:DbConnectAdmin` izin pada kluster target.

Kesalahan akses S3

Pastikan identitas IAM Anda memiliki izin baca S3 yang sesuai untuk bucket sumber dan objek.

Kesalahan inferensi skema

Saat menggunakan`--if-not-exists`, pastikan file data Anda memiliki tipe kolom yang konsisten. Jenis campuran dalam kolom dapat menyebabkan inferensi skema gagal.

Gandakan kesalahan kunci pada resume

Jika Anda mengalami kesalahan kunci duplikat saat melanjutkan pemuatan, tambahkan batasan unik ke tabel target sehingga loader dapat digunakan untuk melewati catatan yang sudah dimuat. `ON CONFLICT DO NOTHING`

Untuk informasi pemecahan masalah tambahan, lihat repositori Aurora DSQL Loader. GitHub

Jalur migrasi

Bagian berikut menjelaskan cara memigrasikan data dari sistem sumber umum ke Aurora DSQL.

Migrasi dari PostgreSQL

Untuk memigrasikan data dari database PostgreSQL yang ada ke Aurora DSQL:

1. Ekspor data Anda dari PostgreSQL ke format CSV atau Parket. Anda dapat menggunakan perintah `COPY` PostgreSQL untuk mengekspor setiap tabel:

```
COPY my_table TO '/path/to/my_table.csv' WITH (FORMAT csv, HEADER true);
```

2. Buat tabel target di Aurora DSQL. Anda dapat membuat skema secara manual atau menggunakan `--if-not-exists` flag loader untuk menyimpulkan skema dari data Anda.
3. Muat data yang diekspor menggunakan Aurora DSQL Loader:

```
aurora-dsql-loader load \
```

```
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri /path/to/my_table.csv \
  --table my_table
```

> **Tip**
>
> Untuk migrasi besar, pertimbangkan untuk mengekspor ke format Parket untuk kompresi yang lebih baik dan pemuatan yang lebih cepat. Alat seperti DuckDB dapat mengonversi file CSV ke Parquet secara efisien.

Migrasi dari MySQL

Untuk memigrasikan data dari MySQL ke Aurora DSQL:

1. Ekspor data Anda dari MySQL ke format CSV `SELECT INTO OUTFILE` menggunakan atau alat mysqldump seperti dengan opsi: `--tab`

   ```
   SELECT * FROM my_table
   INTO OUTFILE '/path/to/my_table.csv'
   FIELDS TERMINATED BY ','
   ENCLOSED BY '"'
   LINES TERMINATED BY '\n';
   ```

2. Buat tabel target di Aurora DSQL dengan tipe data yang kompatibel dengan PostgreSQL yang sesuai.
3. Muat data yang diekspor menggunakan Aurora DSQL Loader:

   ```
   aurora-dsql-loader load \
     --endpoint cluster-id.dsql.region.on.aws \
     --source-uri /path/to/my_table.csv \
     --table my_table
   ```

> **Note**
>
> MySQL dan PostgreSQL memiliki sistem tipe data yang berbeda. Tinjau skema Anda dan sesuaikan tipe data sesuai kebutuhan saat membuat tabel di Aurora DSQL.

Memuat dari Amazon S3

Jika data Anda sudah ada di Amazon S3, Anda dapat memuatnya secara langsung tanpa mengunduh ke sistem lokal Anda. Aurora DSQL Loader mendukung S3 secara asli: URIs

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/path/to/data.parquet \
  --table my_table
```

Pastikan identitas IAM Anda memiliki `s3:GetObject` izin pada objek sumber.

Menggunakan PostgreSQL\ copy

Jika Anda sudah terhubung ke Aurora DSQL melalui `psql` sesi yang menangani otentikasi IAM, Anda dapat menggunakan `\copy` meta-command sisi klien untuk memuat data dari sistem file lokal Anda. Berbeda dengan `COPY` pernyataan sisi server, `\copy` membaca file di mesin klien dan mengalirkan data melalui koneksi yang ada, sehingga tidak diperlukan akses file sisi server. Pendekatan ini bekerja dengan baik untuk beban ulir tunggal yang sederhana.

Example Memuat file CSV dengan\ copy

```
\copy my_table FROM '/path/to/data.csv' WITH (FORMAT csv, HEADER true);
```

Saat menggunakan `\copy` secara langsung, Anda bertanggung jawab untuk:

- Mengelola paralelisasi jika memuat banyak file atau kumpulan data besar
- Menangani manajemen koneksi dan penyegaran token otentikasi
- Menerapkan logika coba lagi untuk operasi yang gagal

Praktik terbaik untuk transaksi INSERT

Saat menggunakan `INSERT` pernyataan untuk memuat data ke Aurora DSQL, ikuti praktik berikut untuk meningkatkan throughput dan keandalan:

- Batch rows menjadi multi-row INSERTs — Kelompokkan beberapa baris ke dalam satu `INSERT` pernyataan untuk mengurangi perjalanan pulang pergi. Misalnya, `INSERT INTO my_table VALUES (1, 'a'), (2, 'b'), (3, 'c')` lebih efisien daripada tiga pernyataan terpisah.

- Gunakan kueri berparameter - Gunakan pernyataan yang disiapkan dengan pengikatan parameter alih-alih rangkaian string. Ini menghindari risiko injeksi SQL dan memungkinkan database untuk menggunakan kembali rencana kueri.
- Pertahankan transaksi kecil — Aurora DSQL menggunakan kontrol konkurensi yang optimis, sehingga transaksi besar yang menyentuh banyak baris lebih mungkin menghadapi konflik. Bertujuan untuk transaksi beberapa ratus baris daripada ribuan.
- Implementasikan logika coba ulang — Kesalahan sementara seperti konflik kontrol konkurensi optimis (OCC) diharapkan dalam sistem terdistribusi. Terapkan backoff eksponensial dengan coba lagi untuk transaksi yang gagal.
- Paralelisasi di seluruh koneksi - Buka beberapa koneksi dan distribusikan sisipan di atasnya. Setiap koneksi dapat memproses subset data yang berbeda secara bersamaan.

Untuk sebagian besar kasus penggunaan, Aurora DSQL Loader menyediakan pendekatan yang lebih sederhana dan lebih kuat untuk pemuatan data.

Sumber daya tambahan

- Aurora DSQL Loader aktif GitHub — Kode sumber, dokumentasi, dan pelacakan masalah
- Menghasilkan token otentikasi di Amazon Aurora DSQL— Pelajari tentang token otentikasi IAM untuk Aurora DSQL
- Mengakses Aurora DSQL dengan klien PostgreSQL-compatible— Connect ke Aurora DSQL menggunakan berbagai klien dan alat

AI generatif untuk Aurora DSQL

Bagian ini memberikan petunjuk terperinci tentang cara menggunakan alat AI Generatif dengan Aurora DSQL

AWS Labs Aurora DSQL MCP Server

Server AWS Labs Model Context Protocol (MCP) untuk Aurora DSQL

Fitur

- Mengubah pertanyaan dan perintah yang dapat dibaca manusia menjadi kueri Postgres-compatible SQL terstruktur dan menjalankannya terhadap database Aurora DSQL yang dikonfigurasi.

- Read-only secara default, transaksi diaktifkan dengan `--allow-writes`
- Penggunaan kembali koneksi antara permintaan untuk meningkatkan kinerja
- Built-in akses ke dokumentasi Aurora DSQL, pencarian, dan rekomendasi praktik terbaik
- Validasi SQL untuk kompatibilitas Aurora DSQL dengan perbaikan otomatis opsional

Alat yang Tersedia

Operasi Database

- readonly_query - Jalankan kueri SQL read-only terhadap cluster DSQL Anda
- transact - Jalankan operasi tulis dalam transaksi (membutuhkan`--allow-writes`)
- get_schema - Ambil informasi skema tabel

Dokumentasi dan Rekomendasi

- dsql_search_documentation - Cari dokumentasi Aurora DSQL
  - Parameter: `search_phrase` (wajib), `limit` (opsional)
- dsql_read_documentation - Baca halaman dokumentasi DSQL tertentu
  - Parameter: `url` (wajib), `start_index` (opsional), `max_length` (opsional)
- dsql_recommend - Dapatkan rekomendasi untuk praktik terbaik DSQL
  - Parameter: `url` (diperlukan)

SQL Validasi

- dsql_lint - Validasi SQL untuk kompatibilitas Aurora DSQL dan masalah perbaikan otomatis opsional
  - Parameter: `sql` (wajib), `fix` (opsional)

Prasyarat

1. Akun AWS dengan Aurora DSQL Cluster
2. Server MCP ini hanya dapat dijalankan secara lokal pada host yang sama dengan klien LLM Anda.
3. Siapkan kredensil AWS dengan akses ke layanan AWS
   - Anda memerlukan akun AWS dengan peran termasuk izin ini:
     - `dsql:DbConnectAdmin`- Connect ke cluster DSQL sebagai pengguna admin
     - `dsql:DbConnect`- Connect ke cluster DSQL dengan peran database kustom (hanya diperlukan jika menggunakan pengguna non-admin)

- Konfigurasikan kredensional AWS dengan `aws configure` atau variabel lingkungan

Penginstalan

Untuk sebagian besar alat, memperbarui konfigurasi dengan mengikuti petunjuk Instalasi Default harus cukup.

Instruksi terpisah diuraikan untuk Claude Code dan Codex.

Instalasi Default: Memperbarui File Konfigurasi MCP yang Relevan

Menggunakan **uv**

1. Instal uv dari Astral atau README GitHub
2. Instal Python menggunakan `uv python install 3.10`

Konfigurasikan server MCP dalam konfigurasi klien MCP Anda (Menemukan File Konfigurasi MCP)

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "command": "uvx",
      "args": [
        "awslabs.aurora-dsql-mcp-server@latest",
        "--cluster_endpoint",
        "[your dsql cluster endpoint, e.g. abcdefghijklmnopqrst234567.dsql.us-
east-1.on.aws]",
        "--region",
        "[your dsql cluster region, e.g. us-east-1]",
        "--database_user",
        "[your dsql username, e.g. admin]",
        "--profile",
        "[your aws profile, e.g. default]"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR"
      },
      "disabled": false,
      "autoApprove": []
    }
```

```
  }
}
```

Instalasi Windows

Untuk pengguna Windows, format konfigurasi server MCP sedikit berbeda:

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "disabled": false,
      "timeout": 60,
      "type": "stdio",
      "command": "uv",
      "args": [
        "tool",
        "run",
        "--from",
        "awslabs.aurora-dsql-mcp-server@latest",
        "awslabs.aurora-dsql-mcp-server.exe"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR",
        "AWS_PROFILE": "your-aws-profile",
        "AWS_REGION": "us-east-1"
      }
    }
  }
}
```

Menemukan File Konfigurasi Klien MCP

Untuk beberapa alat pengembangan Agentic yang paling umum, Anda dapat menemukan konfigurasi klien MCP Anda di jalur file berikut:

- Kiro:
  - Konfigurasi Pengguna: `~/.kiro/settings/mcp.json`
  - Config Ruang Kerja: `/path/to/workspace/.kiro/settings/mcp.json`
- Kode Claude: Lihat Instalasi Kode Claude untuk bantuan penyiapan terperinci
  - Konfigurasi Pengguna: di `~/.claude.json` `"mcpServers"`
  - Konfigurasi Proyek: `/path/to/project/.mcp.json`

  - Config Lokal: di `~/.claude.json` `"projects" -> "path/to/project" -> "mcpServers"`
- Kursor:
  - Global: `~/.cursor/mcp.json`
  - Proyek: `/path/to/project/.cursor/mcp.json`
- Kodeks: `~/.codex/config.toml`
  - Setiap server MCP dikonfigurasi dengan `[mcp_servers.<server-name>]` tabel di file konfigurasi. Lihat Petunjuk Instalasi Codex Kustom
- Lengkung:
  - Pengeditan File: `~/.warp/mcp_settings.json`
  - Editor Aplikasi: `Settings > AI > Manage MCP Servers` dan tempel json
- CLI Pengembang Amazon Q: `~/.aws/amazonq/mcp.json`
- Cline: Biasanya jalur VS Code bersarang - `~/.vscode-server/path/to/cline_mcp_settings.json`

Kode Claude

Prasyarat

Penting: Manajemen server MCP hanya tersedia melalui pengalaman terminal Claude Code CLI, bukan mode panel asli VS Code.

Instal Claude Code CLI terlebih dahulu dengan mengikuti proses yang direkomendasikan instalasi asli Claude.

Memilih Lingkup yang Tepat

Claude Code menawarkan 3 cakupan berbeda: lokal (default), proyek, dan pengguna dan detail ruang lingkup mana yang harus dipilih berdasarkan sensitivitas kredensi dan perlu dibagikan. Lihat dokumentasi Kode Claude di Lingkup Instalasi MCP untuk detail selengkapnya.

1. Local-scopedserver mewakili tingkat konfigurasi default dan disimpan di `~/.claude.json` bawah jalur proyek Anda. Keduanya bersifat pribadi untuk Anda dan hanya dapat diakses dalam direktori proyek saat ini. Ini adalah default `scope` saat membuat server MCP.
2. Project-scopedserver mengaktifkan kolaborasi tim sementara masih hanya dapat diakses di direktori proyek. Project-scoped server menambahkan `.mcp.json` file di direktori root proyek Anda. File ini dirancang untuk diperiksa ke kontrol versi, memastikan semua anggota tim memiliki akses ke alat dan layanan MCP yang sama. Saat Anda menambahkan server cakupan proyek,

Claude Code secara otomatis membuat atau memperbarui file ini dengan struktur konfigurasi yang sesuai.

3. User-scopedserver disimpan `~/.claude.json` dan menyediakan aksesibilitas lintas proyek, membuatnya tersedia di semua proyek di komputer Anda sambil tetap pribadi ke akun pengguna Anda.

Menggunakan Claude CLI (disarankan)

Menggunakan sesi `claude` CLI interaktif memungkinkan pengalaman pemecahan masalah yang lebih baik, jadi ini adalah jalur yang disarankan.

```
claude mcp add amazon-aurora-dsql \
  --scope [one of local, project, or user] \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Pemecahan Masalah: Menggunakan Kode Claude dengan Bedrock di Akun AWS yang berbeda

Jika Anda telah mengonfigurasi Claude Code dengan akun atau profil AWS Bedrock yang berbeda dari profil yang diperlukan untuk terhubung ke kluster dsql Anda, Anda harus memberikan argumen lingkungan tambahan:

```
  --env AWS_PROFILE="[dsql profile, eg. default]" \
  --env AWS_REGION="[dsql cluster region, eg. us-east-1]" \
```

Modifikasi Langsung dalam File Konfigurasi

Kode Claude Memerlukan penamaan alfanumerik, jadi kami sarankan untuk memberi nama server Anda:. `aurora-dsql-mcp-server`

Local-Scope

Perbarui `~/.claude.json` dalam bidang khusus proyek`mcpServers`:

```
{
  "projects": {
    "/path/to/project": {
      "mcpServers": {}
    }
  }
}
```

Project-Scope

Perbarui `/path/to/project/root/.mcp.json` di `mcpServers` bidang:

```
{
  "mcpServers": {}
}
```

User-Scope

Perbarui `~/.claude.json` dalam bidang khusus proyek`mcpServers`:

```
{
  "mcpServers": {}
}
```

kodeks

Opsi 1: Codex CLI

Jika Anda memiliki Codex CLI diinstal, Anda dapat menggunakan perintah codex mcp untuk mengkonfigurasi server MCP Anda.

```
codex mcp add amazon-aurora-dsql \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Opsi 2: config.toml

Untuk kontrol yang lebih halus atas opsi server MCP, Anda dapat mengedit file `~/.codex/config.toml` konfigurasi secara manual. Setiap server MCP dikonfigurasi dengan `[mcp_servers.<server-name>]` tabel di file konfigurasi.

```
[mcp_servers.amazon-aurora-dsql]
command = "uvx"
args = [
  "awslabs.aurora-dsql-mcp-server@latest",
  "--cluster_endpoint", "<DSQL_CLUSTER_ID>.dsql.<AWS_REGION>.on.aws",
  "--region", "<AWS_REGION>",
  "--database_user", "<DATABASE_USERNAME>"
]

[mcp_servers.amazon-aurora-dsql.env]
FASTMCP_LOG_LEVEL = "ERROR"
```

Memverifikasi Instalasi

Untuk Amazon Q Developer CLI, Kiro CLI, CLI/TUI Claude, atau CLI/TUI Codex, `/mcp` jalankan untuk melihat status server MCP.

Untuk IDE Kiro, Anda juga dapat menavigasi ke `MCP SERVERS` tab Panel Kiro yang menunjukkan semua server MCP yang dikonfigurasi dan indikator status koneksinya.

Opsi Konfigurasi Server

`--allow-menulis`

Secara default, server mcp dsql tidak mengizinkan operasi tulis (“mode hanya-baca”). Setiap pemanggilan alat transaksi akan gagal dalam mode ini. Untuk menggunakan alat transaksi, izinkan penulisan dengan melewati `--allow-writes` parameter.

Sebaiknya gunakan akses hak istimewa paling rendah saat menghubungkan ke DSQL. Misalnya, pengguna harus menggunakan peran yang hanya-baca jika memungkinkan. Mode hanya-baca memiliki upaya terbaik penegakan sisi klien untuk menolak mutasi.

`--cluster_titik akhir`

Ini adalah parameter wajib untuk menentukan cluster untuk terhubung ke. Ini harus menjadi titik akhir penuh cluster Anda, misalnya, `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`

`--database_pengguna`

Ini adalah parameter wajib untuk menentukan pengguna untuk terhubung sebagai. Misalnya`admin`, atau`my_user`. Perhatikan bahwa kredensil AWS yang Anda gunakan harus memiliki izin untuk masuk sebagai pengguna tersebut. Untuk informasi selengkapnya tentang pengaturan dan penggunaan peran database di DSQL, lihat Menggunakan peran database dengan peran IAM.

`--profil`

Anda dapat menentukan profil aws yang akan digunakan untuk kredensil Anda. Perhatikan bahwa ini tidak didukung untuk instalasi docker.

Menggunakan variabel `AWS_PROFILE` lingkungan dalam konfigurasi MCP Anda juga didukung:

```
"env": {
  "AWS_PROFILE": "your-aws-profile"
}
```

Jika tidak ada yang disediakan, server MCP default menggunakan profil “default” di file konfigurasi AWS Anda.

`--wilayah`

Ini adalah parameter wajib untuk menentukan wilayah database DSQL Anda.

`--pengetahuan-server`

Parameter opsional untuk menentukan titik akhir server MCP jarak jauh untuk alat pengetahuan DSQL (pencarian dokumentasi, pembacaan, dan rekomendasi). Secara default sudah dikonfigurasi sebelumnya.

Contoh:

```
--knowledge-server https://custom-knowledge-server.example.com
```

Catatan: Untuk keamanan, hanya gunakan endpoint server pengetahuan tepercaya. Server harus menjadi titik akhir HTTPS.

`--knowledge-timeout`

Parameter opsional untuk menentukan batas waktu dalam hitungan detik untuk permintaan ke server pengetahuan.

Default: `30.0`

Contoh:

```
--knowledge-timeout 60.0
```

Tingkatkan nilai ini jika Anda mengalami batas waktu saat mengakses dokumentasi di jaringan yang lambat.

Aurora DSQL Steering: Keterampilan dan Kekuatan

Bagian ini menjelaskan cara mengonfigurasi kemudi AI untuk Aurora DSQL menggunakan keterampilan dan kekuatan. File konfigurasi berbasis penurunan harga ini memberikan konteks dan panduan yang diterapkan asisten AI secara otomatis saat membuat kode untuk meningkatkan kualitas pengembangan agen.

Ikhtisar

Keterampilan dan kekuatan adalah kemampuan modular yang memperluas fungsionalitas asisten AI untuk Aurora DSQL. Mereka mengemas instruksi, metadata, dan sumber daya yang digunakan asisten AI secara otomatis saat bekerja dengan database Aurora DSQL.

Mengapa Menggunakan Keterampilan dan Kekuatan

Keterampilan dan kekuatan memberikan beberapa manfaat utama untuk pengembangan Aurora DSQL:

- Spesialisasi asisten AI - Menyediakan keahlian khusus domain untuk Aurora DSQL termasuk praktik terbaik, pola SQL, dan pengoptimalan basis data terdistribusi. Postgres-compatible
- Kurangi pengulangan - Buat sekali, gunakan secara otomatis. Menghilangkan kebutuhan untuk berulang kali memberikan panduan yang sama di beberapa percakapan.
- Efisiensi konteks - Keterampilan memuat sesuai permintaan daripada mengkonsumsi konteks di muka. AI memuat informasi secara bertahap sesuai kebutuhan.
- Pembelajaran berkelanjutan - Saat fitur Aurora DSQL berkembang, asisten AI mengakses pola yang diperbarui secara otomatis saat keterampilan diperbarui.

Jalur Pengaturan yang Direkomendasikan

Pilih jalur penyiapan yang cocok dengan lingkungan pengembangan Anda:

- the section called “Keterampilan CLI” (Agent-Agnostic)
- the section called “Kekuatan Kiro”
- the section called “Keterampilan Claude”
- the section called “Keterampilan Gemini”
- the section called “Keterampilan Codex”

Keterampilan DSQL juga dapat digunakan dengan agen pengkodean AI lainnya dengan menyalin folder keterampilan ke alat atau direktori. `rules` `skills`

Keterampilan CLI

Skill DSQL dapat diinstal menggunakan Skills CLI. Metode penyiapan agnostik agen ini bekerja dengan sebagian besar asisten pengkodean AI dan memungkinkan Anda menginstal keterampilan ke beberapa agen sekaligus.

Pengaturan

Jalankan perintah berikut untuk menginstal skill Aurora DSQL:

```
npx skills add awslabs/mcp --skill dsql
```

CLI akan memandu Anda melalui:

- Memilih agen - Pilih agen mana yang akan dipasang (Kiro, Claude Code, Cursor, Copilot, Gemini, Codex, Roo, Cline,, Windsurf, dll.) OpenCode
- Lingkup instalasi - Pilih antara:
  - Proyek: Instal di direktori saat ini (berkomitmen dengan proyek Anda)
  - Global: Instal di direktori home (tersedia di semua proyek)
- Metode instalasi - Pilih antara:
  - Symlink (Disarankan): Satu sumber kebenaran, pembaruan mudah
  - Salin ke semua agen: Salinan independen untuk setiap agen

Keterampilan Mengelola

Periksa dan perbarui keterampilan kapan saja menggunakan:

```
npx skills check
npx skills update
```

Kekuatan Kiro

Kekuatan Kiro adalah paket terpadu yang menggabungkan alat MCP dengan keahlian kerangka kerja dan instruksi kemudi. Setiap daya mencakup dokumen titik masuk yang menjelaskan alat MCP yang tersedia dan pemicu aktivasi, konfigurasi server MCP, dan panduan khusus alur kerja tambahan yang dimuat sesuai permintaan.

Daya diaktifkan secara dinamis berdasarkan konteks pengguna. Alih-alih memuat semua alat di muka, kekuatan mempertahankan penggunaan dasar mendekati nol sampai kata kunci yang relevan memicu aktivasi.

Pengaturan

Untuk mengatur daya Kiro untuk Aurora DSQL:

1. Instal langsung dari Kiro Powers Registry
2. Setelah dialihkan ke Power di IDE, baik:
   - Pilih tombol Coba Daya. Disarankan untuk pengguna yang ingin AI memandu penyiapan server MCP atau pengalaman orientasi interaktif dengan Aurora DSQL untuk membuat cluster baru.
   - Buka obrolan Kiro baru dan tanyakan apa pun yang berhubungan dengan Aurora DSQL. Opsional memperbarui MCP Config dengan detail cluster Anda yang ada untuk menguji koneksi server MCP sehingga dapat digunakan di luar kotak dengan daya. Agen Kiro akan secara otomatis mengaktifkan daya jika mengidentifikasi kekuatan sebagai berharga untuk menyelesaikan tugas pengguna.

Keterampilan Claude

Keterampilan Claude adalah kemampuan modular yang memperluas fungsionalitas Claude. Setiap keterampilan mengemas instruksi, metadata, dan sumber daya opsional yang digunakan Claude secara otomatis jika relevan. Keterampilan berbasis sistem file dan dimuat sesuai permintaan untuk meminimalkan penggunaan konteks.

Pengaturan Sederhana dengan CLI Keterampilan

Keterampilan dapat diinstal ke Kode Claude menggunakan. the section called “Keterampilan CLI” Untuk menentukan hanya Claude Code sebagai agen untuk menginstal, gunakan:

```
npx skills add awslabs/mcp --skill dsql --agent claude-code
```

Alternatif: Pengaturan Langsung Menggunakan Klon Git

Pengaturan alternatif mengambil klon jarang dari direktori dsql-skill dan menghubungkan klon ini ke folder. `~/.claude/skills/` Hal ini memungkinkan perubahan pada keterampilan yang akan ditarik setiap kali keterampilan perlu diperbarui.

Prasyarat

- Git diinstal

Langkah Penyiapan

1. Buat direktori repo dasar

```
mkdir -p .dsql_skill_repos
```

2. Sparse clone skill dari repositori MCP

Kloning hanya `dsql-skill` folder (tidak ada file lain):

```
cd .dsql_skill_repos
git clone --filter=blob:none --no-checkout https://github.com/awslabs/mcp.git
cd mcp
git sparse-checkout init --cone
git sparse-checkout set src/aurora-dsql-mcp-server/skills/dsql-skill
git checkout
cd ../..
```

3. Symlink keterampilan ke Direktori Keterampilan

Tambahkan direktori skill (default: global/user -scoped):

```
mkdir -p ~/.claude/skills
```

> **Note**
>
> Jika Anda ingin menjadikan ini keterampilan cakupan proyek, gunakan direktori root proyek Anda sebagai gantinya. `.claude/skills/`

Tambahkan symlink:

```
ln -s "$(pwd)/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/dsql-skill"
 ~/.claude/skills/dsql-skill
```

4. Verifikasi pengaturan

```
# Should show SKILL.md and other skill files
ls -la ~/.claude/skills/dsql-skill/
```

5. Verifikasi Penggunaan Keterampilan

Setelah keterampilan dikonfigurasi, Anda harus memiliki perintah keterampilan baru:`/dsql`. Anda mungkin harus memulai ulang Kode Claude setelah menambahkan keterampilan agar dapat dideteksi. Anda dapat menggunakan perintah ini dari Claude Code CLI atau panel seperti yang diinginkan.

Memperbarui Keterampilan

Untuk menarik perubahan terbaru dari repositori:

```
cd .dsql_skill_repos/mcp
git pull
```

Struktur Direktori

Setelah menyiapkan keterampilan global, Anda akan melihat direktori ini:

```
.dsql_skill_repos/
### mcp/                              # Sparse git checkout
    ### src/
        ### aurora-dsql-mcp-server/
            ### skills/
                ### dsql-skill/
                    ### SKILL.md
                    ### ...

~/.claude/
### skills/
    ### dsql-skill -> /path/to/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/
dsql-skill
```

> **Note**
>
> Tambahkan `.dsql_skill_repos/` ke Anda `.gitignore` jika Anda tidak ingin melacaknya. Checkout yang jarang hanya menyimpan folder keterampilan, meminimalkan penggunaan disk.

Keterampilan Gemini

Untuk menambahkan keterampilan Aurora DSQL langsung di Gemini, tentukan ruang lingkup: `workspace` (terkandung untuk proyek) atau `user` (default, global) dan gunakan penginstal keterampilan.

Pengaturan

```
gemini skills install https://github.com/awslabs/mcp.git --path src/aurora-dsql-mcp-
server/skills/dsql-skill --scope $SCOPE
```

Ganti `$SCOPE` dengan salah satu `workspace` atau`user`.

Anda kemudian dapat menggunakan perintah `/dsql` keterampilan dengan Gemini, dan Gemini akan secara otomatis mendeteksi kapan keterampilan harus digunakan.

Keterampilan Codex

Gunakan penginstal keterampilan dari Codex CLI atau TUI menggunakan keterampilan. `$skill-installer`

Pengaturan

```
$skill-installer install dsql skill: https://github.com/awslabs/mcp/tree/main/src/
aurora-dsql-mcp-server/skills/dsql-skill
```

Mulai ulang Codex untuk mengambil keterampilan. Keterampilan kemudian dapat diaktifkan menggunakan`$dsql`.

Memulai dengan Aurora DSQL Query Editor

Dengan Aurora DSQL Query Editor, Anda dapat terhubung dengan aman ke cluster Aurora DSQL Anda dan menjalankan kueri SQL langsung dari Konsol Manajemen tanpa menginstal atau

mengonfigurasi klien eksternal. AWS Ini menyediakan ruang kerja intuitif dengan penyorotan sintaks bawaan, pelengkapan otomatis, dan bantuan kode cerdas. Anda dapat dengan cepat menjelajahi objek skema, mengembangkan dan menjalankan kueri SQL, dan melihat hasil, semuanya dalam satu antarmuka.

Topik ini memandu Anda melalui langkah-langkah untuk terhubung ke klaster, menjalankan kueri, melihat hasil, dan menjelajahi kemampuan lanjutan seperti rencana eksekusi.

> **Note**
>
> Editor Kueri tersedia di semua Wilayah di mana Aurora DSQL didukung. Untuk detail selengkapnya tentang ketersediaan regional, lihat Layanan AWS Regional.

Prasyarat

Sebelum Anda memulai, pastikan Anda memenuhi persyaratan berikut:

- Anda memiliki setidaknya satu cluster Aurora DSQL yang tersedia. Untuk detail selengkapnya tentang membuat cluster, lihatLangkah 1: Buat cluster Aurora DSQL Single-region.
- Titik akhir cluster Anda dapat diakses oleh publik. Editor Kueri tidak mendukung klaster yang memiliki akses publik yang diblokir oleh kebijakan berbasis sumber daya atau kluster yang dikelola melalui titik akhir VPC. Untuk detail selengkapnya tentang pembatasan akses, lihat Memblokir akses publik dengan kebijakan berbasis sumber daya di Aurora DSQL danMengelola dan menghubungkan ke cluster DSQL Amazon Aurora menggunakan AWS PrivateLink.
- Pengguna atau peran IAM Anda memiliki izin yang diperlukan untuk mengakses dan terhubung ke cluster. Untuk detail selengkapnya tentang izin, lihatMenggunakan peran database dan otentikasi IAM.

Bekerja dengan Query Editor

Buka Editor Kueri

Untuk membuka Query Editor

1. Buka konsol Aurora DSQL.
2. Di panel navigasi, pilih Editor Kueri.

Atau, dari halaman Clusters, pilih cluster yang ingin Anda kueri dan pilih Connect with Query editor untuk meluncurkan editor secara langsung.

> **Note**
>
> Status kerja dan koneksi tidak disimpan. Jika Anda menavigasi jauh dari konsol Aurora DSQL, menutup tab browser, atau keluar, koneksi, teks kueri, dan hasil Anda akan hilang.

Connect ke sebuah cluster

Untuk terhubung ke cluster

1. Jika tidak ada koneksi cluster, editor akan menampilkan Tidak ada cluster yang terhubung. Pilih Connect atau pilih + (Add) di panel Cluster Explorer untuk terhubung ke cluster yang ada.
2. (Opsional) Hubungkan ke beberapa cluster atau ke cluster yang sama dengan menggunakan peran yang berbeda.

Jelajahi objek cluster

Cluster Explorer menampilkan semua koneksi cluster yang tersedia dan memungkinkan Anda menelusuri objek seperti database, skema, tabel, dan tampilan. Ini juga menyediakan tindakan umum seperti Refresh, Create table, dan opsi khusus konteks lainnya.

Jalankan kueri

Untuk menjalankan kueri

1. Di panel tab editor kueri, masukkan pernyataan SQL Anda. Contoh:

   ```
   SELECT * FROM public.orders LIMIT 10;
   ```

2. Verifikasi Konteks Cluster Aktif yang ditampilkan di kanan atas tab kueri. Ini menunjukkan koneksi cluster yang terkait dengan tab kueri saat ini.
3. (Opsional) Gunakan dropdown koneksi untuk meninjau semua koneksi yang tersedia atau beralih ke cluster yang berbeda. Mengubah pembaruan koneksi tempat kueri Anda di tab itu dijalankan.
4. Pilih Jalankan untuk menjalankan kueri.

> **Note**
>
> Setiap kueri dapat mengembalikan hingga 10.000 baris di panel hasil. Untuk kumpulan data yang lebih besar, perbaiki kueri Anda dengan filter atau batas.

Meninjau hasil dan rencana eksekusi

Setelah kueri berjalan, tinjau output di panel Results di bagian bawah editor. Secara default, setiap eksekusi kueri menampilkan tab Hasil (Tabel), menampilkan keluaran kueri tabular.

Untuk mendapatkan rencana eksekusi kueri, jalankan `EXPLAIN ANALYZE` atau `EXPLAIN ANALYZE VERBOSE` dapatkan wawasan tambahan tentang kinerja kueri. Untuk detail selengkapnya tentang rencana eksekusi, lihatMembaca Aurora DSQL JELASKAN rencana.

> **Tip**
>
> `EXPLAIN ANALYZE VERBOSE`Perintah tersebut menampilkan perkiraan penggunaan DPU termasuk nilai Compute, Read, Write, dan Total DPU, memberikan visibilitas langsung ke sumber daya yang dikonsumsi oleh pernyataan SQL individual.

Editor Kueri: Menggunakan JupyterLab dengan Aurora DSQL

Panduan ini memberikan petunjuk langkah demi langkah tentang cara menghubungkan dan menanyakan Amazon Aurora DSQL menggunakan Python. JupyterLab JupyterLab adalah lingkungan komputasi interaktif populer yang menggabungkan kode, teks, dan visualisasi dalam satu dokumen. Ini banyak digunakan untuk ilmu data dan aplikasi penelitian.

Petunjuk di bawah ini akan mencakup dasar-dasar penggunaan Aurora DSQL baik dalam instalasi lokal JupyterLab maupun menggunakan Amazon SageMaker AI, layanan pembelajaran mesin yang dikelola sepenuhnya yang menyediakan lingkungan yang dihosting dengan UI untuk alur kerja data.

Memulai

Persyaratan

- Cluster Aurora DSQL

- Kredensi AWS dikonfigurasi (hanya instalasi lokal)
- Python versi 3.9 atau lebih tinggi (hanya instalasi lokal)

Menggunakan lokal JupyterLab

Untuk memulai JupyterLab, pengguna harus terlebih dahulu menginstal aplikasi menggunakan pip Python:

```
pip install jupyterlab
```

JupyterLab kemudian dapat dibuka dengan berlari**`jupyter lab`**. Ini akan membuka JupyterLab aplikasi di localhost:8888, dapat diakses di browser. Pastikan Anda memiliki kredensi AWS yang dikonfigurasi di lingkungan lokal Anda sebelum melanjutkan.

Menggunakan Amazon SageMaker AI

Di konsol AWS, lanjutkan ke halaman konsol Amazon SageMaker AI dan kemudian ke bagian Notebook di bawah Aplikasi dan IDE. Dari sana Anda dapat memilih Buat instance notebook untuk mulai membuat SageMaker lingkungan. Pilih jenis dan platform instance sebelum mengklik Buat instance notebook.

Lihat dokumentasi penyiapan Amazon SageMaker AI untuk informasi selengkapnya tentang opsi penyiapan dan instans.

> **Note**
>
> Peringatan: Menggunakan Amazon SageMaker AI dapat mengakibatkan biaya ke akun AWS Anda.

Setelah SageMaker instans menjadi aktif, Anda dapat membukanya dari bagian instance Notebook dengan Buka JupyterLab. Sebelum memulai dengan Aurora DSQL di notebook Anda, Anda harus menyediakan akses ke cluster DSQL Anda dalam peran IAM instans. SageMaker Cara termudah untuk melakukannya adalah dengan mengikuti tautan ke peran IAM di halaman instance notebook. Dari sana Anda dapat mengedit Kebijakan yang dilampirkan ke peran SageMaker IAM Anda. Lihat Otentikasi dan otorisasi untuk informasi selengkapnya tentang mengonfigurasi kebijakan IAM untuk mengizinkan akses ke Aurora DSQL.

Menghubungkan ke Aurora DSQL menggunakan JupyterLab

Setelah Anda menyiapkan JupyterLab instance, langkah-langkah untuk terhubung ke Aurora DSQL sama secara lokal dan di AI. SageMaker Buat notebook Python 3 kosong, di mana Anda dapat menambahkan sel dengan kode Python.

Di sel Python, unduh sertifikat root Amazon dari toko kepercayaan resmi:

```
import urllib.request
urllib.request.urlretrieve('https://www.amazontrust.com/repository/AmazonRootCA1.pem',
 'root.pem')
```

Untuk terhubung ke Aurora DSQL, pertama-tama instal Konektor Aurora DSQL untuk Python dan driver Psycopg di sel Python, lalu impor:

```
pip install aurora_dsql_python_connector psycopg
```

```
import aurora_dsql_psycopg as dsql
```

Dengan konektor yang diimpor, Anda kemudian dapat membuat konfigurasi DSQL dan terhubung. Konektor Python Aurora DSQL akan secara otomatis menangani pembuatan token otentikasi pada setiap koneksi.

```
config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin"
    }

conn = dsql.connect(**config)
```

Setelah menjalankan kode Anda, Anda sekarang harus memiliki koneksi Psycopg ke Aurora DSQL. Anda kemudian dapat menjalankan kueri menggunakan kursor Psycopg dan memberikan kueri SQL Anda. Lihat dokumentasi Psycopg untuk informasi lebih lanjut tentang penggunaan Psycopg dengan database. Postgres-compatible Kueri ini akan menghasilkan daftar tupel di. `results_list`

```
with conn:
    with conn.cursor() as cur:
```

```
        cur.execute("SELECT * FROM table")
        results_list = cur.fetchall()
```

Anda kemudian dapat menggunakan kerangka kerja Python seperti Pandas untuk menganalisis atau memvisualisasikan hasil kueri Anda, misalnya:

```
pip install pandas

import pandas as pd

df = pd.DataFrame(tuples_list)
print(df)
print(f"Total records: {len(df)}")
```

Contoh buku catatan

Notebook sampel menggunakan Aurora DSQL tersedia di repositori sampel Aurora DSQL.

Sumber bacaan lebih lanjut

Dokumentasi penyiapan Amazon SageMaker AI

Konektor Aurora DSQL untuk Python

Dokumentasi panda

Cadangkan dan pulihkan untuk Amazon Aurora DSQL

Amazon Aurora DSQL membantu Anda memenuhi kepatuhan peraturan dan persyaratan kelangsungan bisnis melalui integrasi dengan AWS Backup, layanan perlindungan data yang dikelola sepenuhnya yang memudahkan untuk memusatkan dan mengotomatiskan pencadangan di seluruh AWS layanan, di cloud, dan di tempat. Layanan ini merampingkan pembuatan cadangan, pengelolaan, dan pemulihan untuk klaster DSQL Aurora Wilayah Tunggal dan Multi-wilayah.

Fitur utama meliputi:

- Manajemen cadangan terpusat melalui Konsol Manajemen AWS, SDK, atau AWS CLI
- Cadangan klaster penuh
- Jadwal pencadangan otomatis dan kebijakan retensi
- Cross-Region dan kemampuan lintas akun
- Konfigurasi WORM (tulis-sekali, baca-banyak) untuk semua cadangan yang Anda simpan

Untuk informasi selengkapnya tentang fitur AWS Backup Vault Lock dan daftar lengkap AWS Backup fitur yang tersedia untuk Aurora DSQL, lihat Manfaat kunci Vault AWS Backup dan ketersediaan fitur di Panduan Pengembang.AWS Backup

Memulai dengan AWS Backup

AWS Backup membuat salinan lengkap cluster Aurora DSQL Anda. Anda dapat mulai menggunakan AWS Backup untuk Aurora DSQL dengan mengikuti langkah-langkah dalam Memulai dengan: AWS Backup

1. Buat cadangan sesuai permintaan untuk perlindungan segera.
2. Buat rencana cadangan untuk pencadangan terjadwal otomatis.
3. Konfigurasikan periode retensi dan penyalinan lintas wilayah.
4. Siapkan pemantauan dan pemberitahuan untuk aktivitas pencadangan.

Memulihkan backup

Saat Anda memulihkan cluster Aurora DSQL, AWS Backup selalu buat cluster baru untuk melestarikan data sumber Anda.

Memulihkan kluster wilayah tunggal

Untuk memulihkan cluster Aurora DSQL Single-region, gunakan konsol: atau https://console.aws.amazon.com/backupCLI untuk memilih titik pemulihan (cadangan) yang ingin Anda pulihkan. Konfigurasikan pengaturan untuk cluster baru yang akan dibuat dari cadangan Anda. Untuk petunjuk mendetail, lihat Mengembalikan klaster DSQL Aurora Wilayah Tunggal.

Memulihkan klaster Multi-wilayah

Memulihkan cluster Multi-region Aurora DSQL didukung melalui konsol: dan file. https://console.aws.amazon.com/backup AWS CLI Untuk petunjuk mendetail, lihat Mengembalikan klaster DSQL Aurora Multi-wilayah.

Untuk mengembalikan ke cluster Aurora DSQL Multi-wilayah, Anda dapat menggunakan cadangan yang diambil dalam satu. Wilayah AWS Namun, sebelum Anda memulai proses pemulihan, Anda harus memastikan ada salinan identik dari cadangan Anda di semua Wilayah AWS untuk klaster Multi-wilayah Anda. Jika Anda belum memiliki salinan tersebut, Anda harus terlebih dahulu menyalin cadangan ke yang lain Wilayah AWS yang mendukung klaster Multi-wilayah.

Kami merekomendasikan untuk membuat salinan cadangan sebagai kunci Wilayah AWS untuk mengaktifkan opsi pemulihan bencana yang kuat dan memenuhi persyaratan kepatuhan. Untuk melihat tersedia Wilayah AWS untuk Aurora DSQL, lihat. the section called “Wilayah AWS ketersediaan”

Untuk petunjuk terperinci tentang langkah-langkah ini, lihat dokumentasi pemulihan Amazon Aurora DSQL.

Pemantauan dan kepatuhan

AWS Backup memberikan visibilitas komprehensif ke dalam operasi pencadangan dan pemulihan dengan sumber daya berikut.

- Dasbor terpusat untuk melacak pencadangan dan pemulihan pekerjaan
- Integrasi dengan CloudWatch dan CloudTrail
- AWS Backup Audit Manager untuk pelaporan dan audit kepatuhan.

Lihat Pencatatan operasi Aurora DSQL menggunakan AWS CloudTrail untuk mempelajari lebih lanjut tentang mencatat catatan tindakan yang diambil oleh pengguna, peran, atau Layanan AWS saat menggunakan Aurora DSQL.

Sumber daya tambahan

Untuk mempelajari lebih lanjut tentang AWS Backup fitur dan dan menggunakannya bersama-sama dengan Aurora DSQL, lihat sumber daya berikut:

- Kebijakan terkelola untuk AWS Backup
- Pemulihan DSQL Amazon Aurora
- Layanan yang didukung oleh Wilayah AWS
- Enkripsi untuk backup di AWS Backup

Dengan menggunakan AWS Backup Aurora DSQL, Anda menerapkan strategi pencadangan yang kuat, sesuai, dan otomatis yang melindungi sumber daya basis data penting Anda sambil meminimalkan overhead administratif. Baik Anda mengelola satu cluster atau penyebaran Multi-wilayah yang kompleks, AWS Backup menyediakan alat yang Anda butuhkan untuk memastikan data Anda tetap aman dan dapat dipulihkan.

Ubah aliran pengambilan data (Pratinjau)

> **Important**
>
> Fitur ini disediakan sebagai AWS Pratinjau dan dapat berubah sewaktu-waktu. Untuk informasi selengkapnya, lihat bagian 2, Beta dan Pratinjau, di Ketentuan AWS Layanan. Untuk mempelajari lebih lanjut tentang harga untuk aliran CDC, lihat halaman harga Aurora DSQL.
>
> Sebelum ketersediaan umum, kami akan menambahkan jenis operasi baru (`"op":"u"`untuk pembaruan) ke muatan aliran Anda. Untuk memastikan aplikasi Anda menangani perubahan ini tanpa modifikasi, perlakukan op nilai yang tidak dikenal sebagai peningkatan dengan menerapkan payload. `after` Lihat Memahami catatan CDC untuk detail.

Amazon Aurora DSQL mengubah data capture (CDC) mengalirkan perubahan database yang dilakukan dalam waktu dekat langsung ke Amazon Kinesis Data Streams. Aurora DSQL memberikan setiap perubahan tingkat baris yang berkomitmen sebagai catatan JSON terstruktur ke aliran data Kinesis yang Anda konfigurasikan.

CDC berguna ketika Anda ingin:

- Jaga agar sistem hilir tetap sinkron — Mereplikasi perubahan pada indeks pencarian, cache, gudang data, atau sistem analitik tanpa pekerjaan batch.
- Membangun arsitektur berbasis peristiwa — Memicu alur kerja, pemberitahuan, atau tindakan layanan mikro sebagai respons terhadap perubahan database.
- Pertahankan jejak audit — Tangkap setiap perubahan yang dilakukan untuk kepatuhan, debugging, atau analisis historis.
- Memisahkan produsen dari konsumen — Biarkan database fokus pada transaksi sementara sistem hilir memproses perubahan dengan kecepatan mereka sendiri.

Cara kerjanya

Aurora DSQL membaca transaksi yang berkomitmen, memformat setiap perubahan baris sebagai catatan JSON terstruktur, dan mengirimkannya ke aliran data Kinesis yang Anda konfigurasikan. CDC secara otomatis menangkap setiap`INSERT`,`UPDATE`, dan `DELETE` di semua tabel pengguna di cluster. Terapkan logika pemfilteran di aplikasi hilir Anda dengan menggunakan `source.schema`

dan `source.table` bidang di setiap rekaman CDC untuk fokus pada tabel atau mengubah kebutuhan aplikasi Anda.

Aliran CDC sepenuhnya dikelola. Aurora DSQL mengelola semua infrastruktur yang diperlukan untuk menangkap peristiwa perubahan, memantau kesehatan aliran, dan melaporkan status melalui operasi dan metrik `GetStream` API. CloudWatch

Aliran CDC menggunakan model bring-your-own-target. Anda membuat dan mengelola aliran data Kinesis di akun Anda, dan Aurora DSQL mengasumsikan peran IAM yang Anda konfigurasikan untuk menulis catatan CDC atas nama Anda. Anda bertanggung jawab atas kapasitas target, enkripsi, dan pengaturan retensi. Untuk target terbaru yang didukung, lihat `TargetDefinition` parameter CreateStreamdi Referensi API DSQL Amazon Aurora. Untuk daftar lengkap operasi API aliran CDC, lihat Referensi API DSQL Amazon Aurora.

Topik di halaman ini

- Pemesanan dan pengiriman semantik
- Multi-Region Konfigurasi aliran CDC
- Memproses catatan CDC di hilir

Topik terkait

- Memulai dengan aliran CDC
- Mengkonfigurasi IAM
- Memahami catatan CDC
- Pemantauan aliran

Pemesanan dan pengiriman semantik

Jaminan pengiriman

Aurora DSQL CDC menjamin bahwa setiap perubahan yang dilakukan mencapai target setidaknya satu kali. Aurora DSQL dapat memberikan catatan lebih dari satu kali. Rancang aplikasi Anda untuk menangani duplikat. Anda dapat mengidentifikasi duplikat dengan membandingkan `source.ts_ns` dan nilai kunci primer — duplikat memiliki nilai yang sama dengan pengiriman asli.

Memesan

Streaming CDC menggunakan `UNORDERED` mode. Dalam praktiknya, catatan tiba dalam perkiraan urutan komit karena Aurora DSQL membaca dan menerbitkan perubahan secara berurutan. Namun, Aurora DSQL tidak menjamin pemesanan yang ketat. Secara khusus:

- Aurora DSQL dapat mengirimkan catatan dari transaksi yang berbeda dalam urutan apa pun.
- Catatan untuk kunci utama yang sama dari transaksi yang berbeda dapat keluar dari perintah komit.
- Catatan dari satu transaksi dapat menyatu dengan catatan dari transaksi lain. Gunakan `source.txId` bidang untuk mengelompokkan catatan berdasarkan transaksi saat alur kerja Anda membutuhkannya.

Setiap catatan CDC menyertakan `source.ts_ns` bidang yang berisi stempel waktu komit transaksi dalam nanodetik. Gunakan bidang ini untuk membuat perintah komit di sisi penerima.

Strategi konsumen

Karena catatan dapat keluar dari perintah komit dan dapat muncul lebih dari satu kali, aplikasi Anda harus memperhitungkan kedua kondisi tersebut.

> **Important**
>
> Tentukan kunci utama pada semua tabel yang berpartisipasi dalam CDC. Tanpa kunci utama, aplikasi Anda tidak dapat menghapus duplikasi catatan atau menghubungkan penghapusan dengan baris yang terpengaruh.

Last-writer-wins (tampilan terwujud, cache)

Lacak `source.ts_ns` nilai tertinggi per kunci primer. Buang catatan apa pun dengan nilai `source.ts_ns` kurang dari atau sama dengan nilai yang dilacak. Ini menyaring duplikat dan catatan di luar pesanan, menjaga status terbaru untuk setiap kunci. Saat Anda memproses delete (`op: "d"`), simpan batu nisan untuk kunci utama yang mempertahankan `source.ts_ns` nilai alih-alih menghapus entri. Batu nisan memastikan bahwa sisipan atau pembaruan dengan yang lebih awal `source.ts_ns` yang tiba setelah penghapusan tidak salah mengembalikan baris.

Every-change pemrosesan (pencatatan audit, sumber acara)

Hapus duplikat dengan membandingkan `source.ts_ns` dikombinasikan dengan nilai kunci primer. Buffer catatan yang masuk dan urutkan berdasarkan `source.ts_ns` sebelum diproses untuk merekonstruksi perintah komit.

Multi-Region Konfigurasi aliran CDC

Aliran CDC adalah sumber daya regional. Setiap aliran milik satu AWS Wilayah dan memberikan perubahan pada aliran data Kinesis di Wilayah yang sama. Pada klaster Multi-wilayah, aliran CDC di salah satu Wilayah menangkap tulisan yang berkomitmen dari semua Wilayah di cluster. Ini berarti Anda hanya perlu satu aliran untuk menangkap setiap perubahan, terlepas dari mana penulisan itu berasal. Untuk mengirimkan catatan CDC di lebih dari satu Wilayah, buat aliran terpisah di setiap Wilayah. Setiap aliran secara independen menangkap set lengkap perubahan komited di seluruh cluster.

Semua sumber daya—cluster Aurora DSQL, aliran data Kinesis, peran layanan IAM, dan prinsip pemanggilan—harus berada di akun dan Wilayah yang sama. AWS

Memproses catatan CDC di hilir

Setelah data CDC tiba di aliran data Kinesis Anda, Anda dapat memprosesnya secara langsung atau meruteknya ke tujuan lain dengan AWS menggunakan layanan integrasi. Tabel berikut merangkum pola pemrosesan umum.

Pola pemrosesan umum untuk catatan CDC

Pola	Cara kerjanya
Konsumsi langsung	Membaca catatan dari Kinesis dengan menggunakan Amazon Kinesis Client Library (KCL), SDK, AWS atau konsumen Kinesis Data Streams. Lihat Mengembangkan konsumen KCL di Panduan Pengembang Amazon Kinesis Data Streams.
AWS Lambda	Konfigurasikan fungsi Lambda sebagai sumber peristiwa untuk aliran data Kinesis Anda untuk memproses setiap kumpulan catatan CDC saat data tersebut tiba. Lihat Menggunakan AWS

Pola	Cara kerjanya
	Lambda dengan Amazon Kinesis di Panduan Pengembang AWS Lambda.
Amazon Data Firehose	Kirimkan data CDC dari Kinesis ke Amazon S3, Amazon Redshift, Amazon Service OpenSearch, atau tujuan lain untuk analitik dan arsip. Lihat Mengirim data ke aliran pengiriman di Panduan Pengembang Amazon Data Firehose.
Self-managed konsumen	Jalankan Apache Kafka Connect dengan konektor sumber Kinesis, Apache Flink, atau kerangka kerja pemrosesan aliran lainnya untuk mengubah dan merutekan catatan. Untuk mengaktifkan Apache Flink AWS, lihat Mengonfigurasi input aplikasi di Amazon Managed Service for Apache Flink Developer Guide.

Setiap catatan CDC menyertakan bidang-bidang seperti`source.schema`,`source.table`, dan `op` yang dapat Anda gunakan untuk merutekan dan memfilter catatan dalam logika pemrosesan Anda. Untuk skema rekaman lengkap, lihatMemahami catatan CDC.

Memulai dengan aliran CDC

> **Important**
>
> Fitur ini disediakan sebagai AWS Pratinjau dan dapat berubah sewaktu-waktu. Untuk informasi selengkapnya, lihat bagian 2, Beta dan Pratinjau, di Ketentuan AWS Layanan. Untuk mempelajari lebih lanjut tentang harga untuk aliran CDC, lihat halaman harga Aurora DSQL.
> Sebelum ketersediaan umum, kami akan menambahkan jenis operasi baru (`"op": "u"`untuk pembaruan) ke payload streaming Anda. Untuk memastikan aplikasi Anda menangani perubahan ini tanpa modifikasi, perlakukan op nilai yang tidak dikenal sebagai

peningkatan dengan menerapkan payload. `after` Lihat Memahami catatan CDC untuk detail.

Panduan ini memandu Anda melalui setiap langkah yang diperlukan untuk memulai streaming perubahan tingkat baris yang berkomitmen dari cluster Aurora DSQL ke aliran data Amazon Kinesis. Pada akhir panduan ini, Anda telah membuat pipeline CDC yang berfungsi dan skrip Python yang membaca dan mencetak catatan perubahan.

Prasyarat

Sebelum Anda mulai, konfirmasikan hal berikut:

- Anda telah membuat klaster Aurora DSQL dalam status. `ACTIVE` Jika klaster Anda menganggur, sambungkan dengan PostgreSQL-compatible klien mana pun untuk membangunkannya sebelum Anda membuat aliran CDC. `CreateStream`mengembalikan kesalahan validasi jika cluster tidak dalam `ACTIVE` status.
- Aurora DSQL mengharuskan semua sumber daya CDC — cluster, aliran data Amazon Kinesis, peran layanan IAM, dan prinsip pemanggilan—berada di akun yang sama. AWS
- Aliran data Amazon Kinesis Anda berada di AWS Wilayah yang sama dengan cluster Aurora DSQL Anda.
- Anda telah menginstal dan mengonfigurasi kredensi AWS CLI dengan yang memiliki izin untuk membuat peran IAM dan aliran data Amazon Kinesis.

Langkah 1: Buat aliran data Amazon Kinesis

Buat aliran data Kinesis di AWS akun dan Wilayah yang sama dengan cluster Aurora DSQL Anda. Catatan CDC lebih besar dari baris Aurora DSQL yang sesuai karena format JSON mencakup nama kolom, metadata, dan overhead encoding.

Mengukur aliran data Kinesis

Aurora DSQL CDC memberikan baris penuh pada setiap perubahan. Pembaruan yang menyentuh satu kolom menghasilkan catatan yang berisi setiap kolom di baris. Hapus catatan adalah pengecualian — mereka hanya menyertakan kolom kunci utama.

Perkirakan ukuran rekor rata-rata

Ukur ukuran baris rata-rata pada disk untuk memahami volume yang akan dihasilkan CDC dan untuk mengantisipasi catatan yang terlalu besar. Query berikut mengembalikan ukuran tuple rata-rata dalam byte untuk tabel:

```
SELECT avg(pg_column_size(t.*)) FROM your_table t;
```

Amplop catatan CDC menambahkan nama kolom, metadata, dan overhead encoding di atas ukuran baris. Untuk format rekaman yang tepat, lihatRekam muatan. Untuk bagaimana Aurora DSQL menangani catatan yang melebihi batas ukuran rekaman Kinesis, lihat. Menangani catatan besar Untuk set lengkap batas layanan Kinesis, lihat Kuota dan batas Amazon Kinesis Data Streams di Panduan Pengembang Amazon Kinesis Data Streams.

> **Important**
>
> Saat Anda membuat aliran data Kinesis, atur yang berikut ini:
>
> - `MaxRecordSizeInKiBke 10240` (10 MiB). Kinesis default maksimum 1 MiB tidak selalu cukup besar untuk catatan Aurora DSQL CDC. Setiap catatan yang melebihi ukuran catatan Kinesis yang dikonfigurasi menyebabkan aliran CDC menjadi terganggu. `KINESIS_OVERSIZE_RECORD` Aurora DSQL membagi catatan besar menjadi fragmen yang masing-masing dapat mendekati 10 MiB, sehingga aliran data Kinesis perlu menerima catatan sebesar itu. Lihat perinciannya di Menangani catatan besar.
> - `StreamModekeON_DEMAND`. On-demand mode menskalakan kapasitas pecahan secara otomatis dan melindungi Anda dari kekurangan penyediaan selama lonjakan yang tidak terduga. Kinesis masih dapat kembali `WriteProvisionedThroughputExceeded` selama ledakan skala detik yang tajam saat kapasitas meningkat. Rencanakan acara pelambatan singkat.

Buat CloudWatch alarm di `IncomingBytes` dan `WriteProvisionedThroughputExceeded` di `AWS/Kinesis` namespace. Kinesis throttling memperlambat pengiriman CDC dan meningkatkan kelambatan replikasi. Untuk DSQL-side metrik Aurora dan panduan alarm, lihat. Memantau praktik terbaik

Contoh berikut menggunakan AWS CLI. Jika AWS CLI versi Anda tidak mendukung `--max-record-size-in-ki-b` parameter, gunakan AWS SDK untuk memanggil operasi CreateStreamKinesis.

```
aws kinesis create-stream \
  --stream-name my-cdc-stream \
  --stream-mode-details StreamMode=ON_DEMAND \
  --max-record-size-in-ki-b 10240 \
  --region region
```

Tunggu hingga streaming menjadi aktif:

```
aws kinesis describe-stream-summary \
  --stream-name my-cdc-stream \
  --region region \
  --query 'StreamDescriptionSummary.StreamStatus'
```

Perintah kembali `"ACTIVE"` ketika aliran sudah siap.

Rekam aliran ARN dari output. Anda membutuhkannya dalam langkah-langkah berikut. ARN memiliki formatnya. `arn:aws:kinesis:region:account-id:stream/my-cdc-stream`

Langkah 2: Buat peran IAM untuk Aurora DSQL

Aurora DSQL mengasumsikan peran IAM untuk menulis catatan CDC ke aliran data Kinesis Anda. Pada langkah ini, Anda membuat peran dengan kebijakan kepercayaan dan melampirkan kebijakan izin. Untuk penjelasan lengkap tentang setiap elemen kebijakan, lihatMengkonfigurasi IAM.

Buat file kebijakan kepercayaan

Simpan JSON berikut sebagai`trust-policy.json`. Ganti *your-account-idregion*,, dan *cluster-id* dengan nilai-nilai Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DSQLAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "dsql.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:SourceAccount": "your-account-id"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:dsql:region:your-account-
id:cluster/cluster-id/stream/*"
                }
            }
        }
    ]
}
```

Buat peran

Jalankan perintah berikut untuk membuat peran IAM:

```
aws iam create-role \
  --role-name dsql-cdc-role \
  --assume-role-policy-document file://trust-policy.json
```

Buat file kebijakan izin

Simpan JSON berikut sebagai`permissions-policy.json`. Ganti nilai placeholder dengan ARN aliran data Kinesis Anda. `KMSAccess`Pernyataan ini hanya diperlukan jika aliran data Kinesis Anda menggunakan kunci yang dikelola AWS KMS pelanggan, tetapi Anda dapat memasukkannya terlebih dahulu sehingga menambahkan kunci yang dikelola pelanggan nanti tidak merusak aliran CDC Anda. Untuk penjelasan lengkap tentang setiap kondisi, lihatKebijakan izin peran layanan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:your-account-id:stream/my-cdc-stream"
        },
        {
```

```
            "Sid": "KMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "kinesis.region.amazonaws.com",
                    "kms:EncryptionContext:aws:kinesis:arn":
 "arn:aws:kinesis:region:your-account-id:stream/my-cdc-stream",
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        }
    ]
}
```

Lampirkan kebijakan izin

Jalankan perintah berikut:

```
aws iam put-role-policy \
  --role-name dsql-cdc-role \
  --policy-name dsql-cdc-kinesis-access \
  --policy-document file://permissions-policy.json
```

Rekam peran ARN dari output. `create-role` ARN memiliki formatnya. `arn:aws:iam::`*`your-account-id`*`:role/`*`dsql-cdc-role`*

Langkah 3: Buat aliran CDC

Gunakan AWS CLI untuk membuat aliran CDC yang menghubungkan cluster Aurora DSQL Anda ke aliran data Kinesis. Ganti nilai placeholder dengan Kinesis stream ARN dari Langkah 1, ARN peran IAM dari Langkah 2, dan pengidentifikasi cluster Anda.

```
aws dsql create-stream \
  --cluster-identifier cluster-id \
  --target-definition '{"kinesis":{"streamArn":"kinesis-stream-arn","roleArn":"role-
arn"}}' \
  --ordering UNORDERED \
  --format JSON \
```

```
  --tags '{"Name":"my-cdc-stream"}' \
  --region region
```

Respons termasuk pengenal aliran dan status. `CREATING` Pembuatan streaming biasanya memakan waktu satu hingga tiga menit.

Tunggu hingga streaming menjadi aktif

Polling status streaming hingga mencapai`ACTIVE`:

```
aws dsql get-stream \
  --cluster-identifier cluster-id \
  --stream-identifier stream-id \
  --region region \
  --query 'status'
```

Anda juga dapat menggunakan `StreamActive` pelayan di AWS SDK untuk melakukan polling secara otomatis.

Setelah streaming mencapai`ACTIVE`, Aurora DSQL mulai mengirimkan perubahan tingkat baris yang berkomitmen ke aliran data Kinesis Anda.

> **Note**
>
> Setiap cluster Aurora DSQL memiliki jumlah maksimum aliran CDC. Jika Anda mencapai batas ini, `CreateStream` kembalikan a`ServiceQuotaExceededException`. Untuk batas default, lihat Kuota dan batas.

Langkah 4: Verifikasi bahwa catatan mengalir

Masukkan baris ke dalam tabel di cluster Aurora DSQL Anda. Contoh:

```
CREATE TABLE IF NOT EXISTS test_cdc (
    id INT PRIMARY KEY,
    message TEXT
);

INSERT INTO test_cdc VALUES (1, 'hello cdc');
```

Baca dari aliran data Kinesis untuk memverifikasi bahwa catatan CDC tiba:

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator \
  --stream-name my-cdc-stream \
  --shard-id shardId-000000000000 \
  --shard-iterator-type TRIM_HORIZON \
  --region region \
  --query 'ShardIterator' --output text)

aws kinesis get-records \
  --shard-iterator "$SHARD_ITERATOR" \
  --region region
```

Setiap `Data` bidang catatan berisi muatan JSON. Saat Anda menggunakan AWS CLI, muatannya ada Base64-encoded dalam respons. Saat Anda menggunakan `boto3` SDK, SDK menerjemahkannya secara otomatis. JSON yang diterjemahkan terlihat seperti berikut:

```
{
    "type": "full",
    "op": "c",
    "before": null,
    "after": {"id": 1, "message": "hello cdc"},
    "source": {
        "version": "1.0",
        "ts_ms": 1705318200000,
        "ts_ns": 1705318200000000000,
        "txId": "ffthunp5stx6ffs2vyfqoatmfu",
        "schema": "public",
        "table": "test_cdc",
        "db": "postgres",
        "cluster": "cluster-id"
    },
    "ts_ms": 1705318200125,
    "ts_ns": 1705318200125483291
}
```

Untuk deskripsi lengkap dari setiap bidang, lihatMemahami catatan CDC.

Langkah 5: Konsumsi catatan dengan skrip Python

Skrip Python berikut membaca catatan CDC dari aliran data Kinesis dan mencetak setiap peristiwa perubahan. Skrip menggunakan klien `boto3` Amazon Kinesis untuk mengulangi pecahan dan

memecahkan kode setiap catatan. Karena Aurora DSQL CDC menggunakan pengiriman paling tidak sekali, skrip mungkin mencetak catatan yang sama lebih dari satu kali.

```
"""
Read CDC records from an Amazon Kinesis data stream.

Usage:
    pip install boto3
    python consume_cdc.py --stream-name my-cdc-stream --region us-east-1
"""
from __future__ import annotations

import argparse
import json

import boto3


def consume_cdc(stream_name: str, region: str) -> None:
    kinesis = boto3.client("kinesis", region_name=region)

    # List all shards (paginate if the stream has many shards)
    shard_ids: list[str] = []
    paginator = kinesis.get_paginator("list_shards")
    for page in paginator.paginate(StreamName=stream_name):
        shard_ids.extend(s["ShardId"] for s in page["Shards"])
    print(f"Reading from {stream_name} ({len(shard_ids)} shard(s))")

    for shard_id in shard_ids:
        iterator_response = kinesis.get_shard_iterator(
            StreamName=stream_name,
            ShardId=shard_id,
            ShardIteratorType="TRIM_HORIZON",
        )
        shard_iterator = iterator_response["ShardIterator"]

        while shard_iterator:
            records_response = kinesis.get_records(
                ShardIterator=shard_iterator, Limit=100
            )
            shard_iterator = records_response.get("NextShardIterator")

            for record in records_response["Records"]:
```

```
                # boto3 decodes Base64 automatically; record["Data"] is bytes.
                payload = json.loads(record["Data"])

                # A record's "type" field identifies its structure.
                # "full": inlined record with before/after values.
                # "chunked": main record that references fragments for a split image.
                # "fragment": one piece of a chunked image; reassemble in production
code.
                # For details, see cdc-record-format.html#cdc-oversized-records.
                record_type = payload.get("type", "full")
                if record_type == "fragment":
                    print(f"[FRAGMENT] chunk_id={payload['chunk_id']}
index={payload['index']}")
                    continue

                source = payload["source"]
                op = payload["op"]
                ts_ns = source["ts_ns"]
                tx_id = source["txId"]
                table = f"{source['schema']}.{source['table']}"

                # Aurora DSQL currently emits "c" for both inserts and updates. A
subsequent
                # release will emit "u" for updates, and "c" for inserts. Design your
                # consumer to handle all three values; this map stays correct across
the
                # transition.
                op_labels = {"c": "INSERT/UPDATE", "u": "UPDATE", "d": "DELETE"}
                print(
                    f"[{op_labels.get(op, op)}] {table} "
                    f"txId={tx_id} ts_ns={ts_ns} type={record_type}"
                )
                if payload.get("after"):
                    print(f"  after:  {json.dumps(payload['after'])}")
                if payload.get("before"):
                    print(f"  before: {json.dumps(payload['before'])}")
                if record_type == "chunked":
                    print(f"  chunked: {json.dumps(payload['chunked'])}")

            if not records_response["Records"]:
                break  # No more records in this shard


if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser(
        description="Consume DSQL CDC records from Kinesis"
    )
    parser.add_argument("--stream-name", required=True, help="Kinesis stream name")
    parser.add_argument("--region", required=True, help="AWS Region")
    args = parser.parse_args()
    consume_cdc(args.stream_name, args.region)
```

Jalankan skrip .

```
pip install boto3
python consume_cdc.py \
  --stream-name my-cdc-stream \
  --region region
```

Skrip mencetak setiap peristiwa perubahan saat tiba. Anda melihat output yang mirip dengan berikut ini:

```
Reading from my-cdc-stream (4 shard(s))
[INSERT/UPDATE] public.test_cdc txId=ffthunp5stx6ffs2vyfqoatmfu
 ts_ns=1705318200000000000 type=full
  after:  {"id": 1, "message": "hello cdc"}
```

Menambahkan deduplikasi last-writer-win

Karena Aurora DSQL CDC menggunakan pengiriman paling sedikit sekali, aplikasi produksi harus menghapus duplikasi dan memesan catatan. Contoh kode berikut menunjukkan pendekatan tanda air tinggi: untuk setiap kunci utama, ia melacak yang tertinggi yang `source.ts_ns` terlihat sejauh ini dan membuang catatan apa pun dengan stempel waktu yang sama atau sebelumnya. Setel PK_COLUMNS ke nama kolom kunci utama dari tabel yang sedang Anda proses. Untuk strategi yang menangani beberapa tabel atau penghapusan, lihatStrategi konsumen.

```
# Set PK_COLUMNS to the primary key column(s) of your table.
PK_COLUMNS = ["id"]

# Maps each primary key value to the highest ts_ns seen for that key.
high_water: dict[tuple, int] = {}

def process_record(payload: dict) -> bool:
    """Return True if the record is new, False if it's a duplicate or stale.
```

```
    Skip fragment records; reassemble them into a full image before calling this.
    """
    if payload.get("type") == "fragment":
        return False  # Fragments are reassembled upstream, not deduplicated here.

    source = payload["source"]
    ts_ns = source["ts_ns"]
    op = payload["op"]

    # For inserts/updates the row is in "after"; for deletes it's in "before".
    row = payload.get("after") or payload.get("before") or {}
    pk = tuple(row.get(col) for col in PK_COLUMNS)

    prev_ts = high_water.get(pk, -1)
    if ts_ns <= prev_ts:
        return False  # Duplicate or out-of-order record

    high_water[pk] = ts_ns
    return True
```

Mengelola aliran CDC

Daftar aliran

Untuk mencantumkan semua aliran CDC untuk klaster, gunakan operasi: `ListStreams`

```
aws dsql list-streams \
  --cluster-identifier cluster-id \
  --region region
```

Menghapus aliran

Untuk menghapus aliran CDC, jalankan perintah berikut:

```
aws dsql delete-stream \
  --cluster-identifier cluster-id \
  --stream-identifier stream-id \
  --region region
```

Anda dapat menggunakan `StreamNotExists` pelayan untuk melakukan polling `GetStream` hingga a `ResourceNotFoundException` dikembalikan, menunjukkan bahwa Aurora DSQL telah sepenuhnya menghapus aliran.

Mengkonfigurasi IAM

> **Important**
>
> Fitur ini disediakan sebagai AWS Pratinjau dan dapat berubah sewaktu-waktu. Untuk informasi selengkapnya, lihat bagian 2, Beta dan Pratinjau, di Ketentuan AWS Layanan. Untuk mempelajari lebih lanjut tentang harga untuk aliran CDC, lihat halaman harga Aurora DSQL.
> Sebelum ketersediaan umum, kami akan menambahkan jenis operasi baru (`"op": "u"`untuk pembaruan) ke muatan aliran Anda. Untuk memastikan aplikasi Anda menangani perubahan ini tanpa modifikasi, perlakukan op nilai yang tidak dikenal sebagai peningkatan dengan menerapkan payload. `after` Lihat Memahami catatan CDC untuk detail.

Aliran CDC memerlukan dua set izin IAM yang terpisah:

- Izin pemanggil - Prinsip IAM yang memanggil operasi API aliran CDC (`CreateStream`,, `GetStreamDeleteStream`,`ListStreams`) memerlukan izin untuk tindakan ini dan untuk. `iam:PassRole`
- Peran layanan — Peran IAM yang diasumsikan Aurora DSQL saat runtime untuk menulis catatan CDC ke target Anda. Anda membuat peran ini, melampirkan kebijakan kepercayaan yang memungkinkan prinsipal layanan Aurora DSQL untuk menerapkannya, dan melampirkan kebijakan izin yang memberikan akses tulis ke target.

> **Note**
>
> Peran layanan CDC terpisah dari kebijakan berbasis sumber daya apa pun di cluster Aurora DSQL Anda. Kebijakan berbasis sumber daya klaster mengontrol prinsipal mana yang dapat terhubung dan menanyakan klaster. Kontrol peran layanan CDC yang menargetkan Aurora DSQL dapat menulis catatan CDC.

Izin penelepon

Prinsip IAM yang memanggil operasi API aliran CDC memerlukan izin untuk tindakan yang relevan `dsql` dan. `iam:PassRole` `CreateStream`Operasi memerlukan `iam:PassRole` karena melewati peran layanan ARN ke Aurora DSQL. Berikut ini adalah contoh kebijakan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DSQLStreamActions",
            "Effect": "Allow",
            "Action": [
                "dsql:CreateStream",
                "dsql:GetStream",
                "dsql:ListStreams",
                "dsql:DeleteStream"
            ],
            "Resource": [
                "arn:aws:dsql:region:your-account-id:cluster/cluster-id",
                "arn:aws:dsql:region:your-account-id:cluster/cluster-id/stream/*"
            ]
        },
        {
            "Sid": "PassServiceRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::your-account-id:role/dsql-cdc-role",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "dsql.amazonaws.com"
                }
            }
        }
    ]
}
```

`Resource`Elemen ini mencakup ARN cluster (diperlukan oleh `CreateStream` dan`ListStreams`) dan pola ARN aliran (diperlukan oleh dan). `GetStream` `DeleteStream`

Untuk daftar lengkap izin yang diperlukan untuk setiap operasi, lihat,, CreateStreamGetStreamDeleteStream, dan ListStreamsdi Referensi API Amazon Aurora DSQL.

Peran layanan

Peran layanan adalah peran IAM yang Aurora DSQL asumsikan untuk menulis catatan CDC ke target Anda. Anda membuat peran ini dan meneruskan ARN di `targetDefinition.kinesis.roleArn`

bidang saat Anda menelepon. `CreateStream` Peran tersebut memerlukan kebijakan kepercayaan dan kebijakan izin.

Kebijakan kepercayaan peran layanan

Kebijakan kepercayaan harus memungkinkan prinsipal layanan Aurora DSQL untuk mengambil peran tersebut. Untuk melindungi dari serangan wakil yang membingungkan, gunakan tombol `aws:SourceAccount` dan `aws:SourceArn` kondisi.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DSQLAssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": "dsql.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "your-account-id"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:dsql:region:your-account-
id:cluster/cluster-id/stream/*"
                }
            }
        }
    ]
}
```

`aws:SourceArn`Kondisi membatasi peran ke aliran di bawah cluster tertentu. Anda harus menggunakan wildcard (`stream/*`) saat membuat aliran karena Aurora DSQL belum menetapkan pengenal aliran. Setelah Anda membuat aliran, Anda dapat mengencangkan kondisi ke aliran ARN (`arn:aws:dsql:region:your-account-id:cluster/cluster-id/stream/stream-id`) yang tepat jika peran tersebut melayani satu aliran.

Untuk menggunakan peran dengan stream di klaster mana pun di akun Anda, gunakan wildcard yang lebih luas:. `arn:aws:dsql:region:your-account-id:cluster/*/stream/*`

Untuk mempelajari lebih lanjut tentang pencegahan wakil yang membingungkan, lihat Cross-service pencegahan wakil bingung di panduan ini.

Kebijakan izin peran layanan

Kebijakan izin memberikan akses peran layanan untuk menulis catatan ke aliran data Kinesis Anda. Kebijakan berikut mencakup izin dan izin penulisan Kinesis. AWS KMS `KMSAccess`Pernyataan ini hanya diperlukan jika aliran data Kinesis Anda menggunakan kunci yang dikelola AWS KMS pelanggan, tetapi Anda dapat memasukkannya terlebih dahulu sehingga menambahkan kunci yang dikelola pelanggan nanti tidak merusak aliran CDC Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:your-account-id:stream/kinesis-stream-
name"
        },
        {
            "Sid": "KMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "kinesis.region.amazonaws.com",
                    "kms:EncryptionContext:aws:kinesis:arn":
 "arn:aws:kinesis:region:your-account-id:stream/kinesis-stream-name",
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        }
```

```
    ]
}
```

Kondisi dalam AWS KMS pernyataan memberikan perlindungan berikut:

- `kms:ViaService`— Membatasi penggunaan kunci untuk permintaan yang datang melalui layanan Kinesis di Wilayah tertentu.
- `kms:EncryptionContext:aws:kinesis:arn`— Membatasi penggunaan kunci untuk operasi enkripsi untuk aliran data Kinesis yang ditentukan.
- `aws:ResourceAccount`— Kunci harus milik AWS akun yang sama dengan prinsipal panggilan, yang mencegah penggunaan kunci lintas akun.

> **Note**
>
> AWS KMS Kunci yang direferensikan di sini adalah kunci enkripsi pada aliran data Kinesis, bukan kunci cluster AWS KMS . Kunci enkripsi cluster melindungi data CDC di dalam batas Aurora DSQL. Kunci enkripsi Kinesis melindungi data CDC setelah Aurora DSQL menuliskannya ke aliran data Kinesis.

Perlindungan data

Aurora DSQL menggunakan Transport Layer Security (TLS) untuk mengenkripsi data CDC dalam perjalanan antara Aurora DSQL dan target Anda. Dalam batas Aurora DSQL, Aurora DSQL mengenkripsi data CDC saat istirahat dengan menggunakan kunci enkripsi cluster.

Jika klaster menggunakan kunci yang dikelola AWS KMS pelanggan dan kunci itu menjadi tidak dapat diakses, transisi `ACTIVE` atau `IMPAIRED` aliran ke `IMPAIRED` kode kesalahan. `CLUSTER_CMK_INACCESSIBLE` Jika kunci menjadi tidak dapat diakses sebelum aliran selesai dibuat, aliran beralih langsung ke. `FAILED`

Untuk penjelasan rinci tentang enkripsi di Aurora DSQL, lihat Enkripsi data untuk Amazon Aurora DSQL dalam panduan ini.

Memahami catatan CDC

> **Important**
>
> Fitur ini disediakan sebagai AWS Pratinjau dan dapat berubah sewaktu-waktu. Untuk informasi selengkapnya, lihat bagian 2, Beta dan Pratinjau, di Ketentuan AWS Layanan. Untuk mempelajari lebih lanjut tentang harga untuk aliran CDC, lihat halaman harga Aurora DSQL.
> Sebelum ketersediaan umum, kami akan menambahkan jenis operasi baru (`"op":"u"`untuk pembaruan) ke muatan aliran Anda. Untuk memastikan aplikasi Anda menangani perubahan ini tanpa modifikasi, perlakukan op nilai yang tidak dikenal sebagai peningkatan dengan menerapkan payload. `after` Lihat Memahami catatan CDC untuk detail.

Aurora DSQL CDC memberikan setiap perubahan sebagai catatan JSON. Rekaman menggunakan struktur amplop dengan tipe operasi, sebelum dan sesudah gambar baris, dan metadata sumber.

Bagaimana catatan memetakan ke Amazon Kinesis

Aurora DSQL menulis setiap catatan CDC sebagai catatan Kinesis tunggal. `Data`Bidang catatan Kinesis berisi muatan JSON. Aurora DSQL menggunakan kunci partisi Kinesis acak untuk mendistribusikan catatan CDC secara merata di seluruh pecahan. Untuk membaca semua perubahan, gunakan semua pecahan pada aliran data Kinesis. Jika catatan melebihi batas ukuran rekaman Kinesis, Aurora DSQL membaginya di beberapa catatan Kinesis. Lihat perinciannya di Menangani catatan besar.

> **Note**
>
> Catatan Kinesis memiliki satu `Data` gumpalan. Nilai kunci primer muncul di bidang payload JSON untuk dihapus, atau `before` `after` bidang untuk sisipan dan pembaruan. Untuk mengekstrak kunci utama untuk pemrosesan hilir, bacalah dari bidang yang sesuai di payload.

Kunci utama dalam muatan

Untuk tabel dengan kunci primer, nilai kolom kunci primer muncul di payload:

- Untuk sisipan dan pembaruan, payload mencakup kolom kunci utama bersama dengan semua kolom lain di `after` bidang.
- Untuk menghapus, kolom kunci utama muncul di `before` bidang.

Misalnya, pertimbangkan tabel dengan kunci primer komposit:

```
CREATE TABLE order_items (
    order_id INT,
    item_id INT,
    quantity INT,
    price NUMERIC,
    PRIMARY KEY (order_id, item_id)
);
```

Penghapusan pada tabel ini menghasilkan muatan di mana`"before": {"order_id": 1001, "item_id": 42}`.

Rekam muatan

Payload menggunakan format amplop JSON berikut.

Contoh INSERT

Contoh berikut menunjukkan catatan CDC untuk operasi insert:

```
{
    "type": "full",
    "op": "c",
    "before": null,
    "after": {"order_id": 1001, "item_id": 42, "quantity": 5, "price": "29.99"},
    "source": {
        "version": "1.0",
        "ts_ms": 1705318200000,
        "ts_ns": 1705318200000000000,
        "txId": "ffthunp5stx6ffs2vyfqoatmfu",
        "schema": "public",
        "table": "order_items",
        "db": "postgres",
        "cluster": "kmabugltfmjdaj2siqr2qbxgju"
    },
    "ts_ms": 1705318200125,
```

```
    "ts_ns": 1705318200125483291
}
```

Contoh UPDATE

Contoh berikut menunjukkan seperti apa catatan CDC yang dihasilkan oleh UPDATE pernyataan setelah Aurora DSQL mulai memancarkan: `op: "u"`

> **Important**
>
> Saat ini Aurora DSQL memancarkan `op: "c"` untuk sisipan dan pembaruan. Rilis berikutnya akan memancarkan pembaruan, dan `op: "u"` `op: "c"` untuk sisipan. Rancang aplikasi Anda untuk ditangani `cu`, `d` sehingga konsumen Anda terus bekerja di seluruh transisi.

```
{
    "type": "full",
    "op": "u",
    "before": null,
    "after": {"order_id": 1001, "item_id": 42, "quantity": 10, "price": "29.99"},
    "source": {
        "version": "1.0",
        "ts_ms": 1705318300000,
        "ts_ns": 1705318300000000000,
        "txId": "qvtiesgmd55cvlfukm3dfuotji",
        "schema": "public",
        "table": "order_items",
        "db": "postgres",
        "cluster": "kmabugltfmjdaj2siqr2qbxgju"
    },
    "ts_ms": 1705318300125,
    "ts_ns": 1705318300125483291
}
```

DELETE contoh

Untuk menghapus pada tabel dengan kunci primer, `before` bidang berisi nilai kunci primer dari baris yang dihapus:

```
{
    "type": "full",
```

```
    "op": "d",
    "before": {"order_id": 1001, "item_id": 42},
    "after": null,
    "source": {
        "version": "1.0",
        "ts_ms": 1705318400000,
        "ts_ns": 1705318400000000000,
        "txId": "xyzabc123def456ghi789jklmno",
        "schema": "public",
        "table": "order_items",
        "db": "postgres",
        "cluster": "kmabugltfmjdaj2siqr2qbxgju"
    },
    "ts_ms": 1705318400125,
    "ts_ns": 1705318400125483291
}
```

Bidang payload

Bidang	Deskripsi
`type`	Jenis rekaman. `full`untuk catatan lengkap yang mencakup inline `before` dan `after` nilai. `chunked`untuk catatan utama yang mereferen sikan catatan fragmen untuk satu atau kedua gambar. `fragment`untuk bagian individu dari gambar yang dipotong. Lihat perinciannya di Menangani catatan besar.
op	Jenis operasi. c= buat (sisipkan), u = pembaruan, d = hapus. Saat ini Aurora DSQL memancarkan c untuk sisipan dan pembaruan. Rilis berikutnya akan memancarkan pembaruan , dan u c untuk sisipan. Rancang aplikasi Anda untuk menangani ketiga nilai tersebut.
`before`	Untuk menghapus pada tabel dengan kunci primer, berisi nilai kunci primer dari baris yang dihapus. Aurora DSQL menetapkan bidang ini

Bidang	Deskripsi
	`null` untuk menyisipkan, memperbarui, dan menghapus pada tabel tanpa kunci utama.
`after`	Status baris penuh setelah perubahan , termasuk semua kolom. Aurora DSQL menetapkan bidang ini untuk dihapus. `null`
`chunked`	Hadir hanya ketika `type` ada`chunked`. Berisi metadata pemasangan ulang untuk `before` gambar, gambar, atau keduanya`after`. Aurora DSQL menghilangkan gambar yang terpotong dari tingkat atas atau bidang dan menempatk annya di bawah`before`. `after` `chunked` Lihat perinciannya di Menangani catatan besar.
`source.version`	Versi format metadata sumber CDC. Versi saat ini adalah `1.0`.
`source.ts_ms`	Transaksi melakukan stempel waktu dalam milidetik sejak zaman Unix, Coordinated Universal Time (UTC).
`source.ts_ns`	Stempel waktu komit transaksi dalam nanodetik , UTC. Stempel waktu presisi tertinggi yang tersedia. Gunakan bidang ini untuk menetapkan urutan total transaksi.
`source.txId`	Pengidentifikasi transaksi unik, dikodekan sebagai base32. Semua catatan dari transaksi yang sama memiliki `txId` nilai yang sama. Gunakan bidang ini untuk mengelompokkan catatan yang termasuk dalam transaksi yang sama.
`source.schema`	Nama skema PostgreSQL (misalnya,). `public`
`source.table`	Nama tabel.

Bidang	Deskripsi
`source.db`	Nama database. Selalu `postgres` untuk Aurora DSQL.
`source.cluster`	Pengidentifikasi cluster Aurora DSQL.
`ts_ms`	Waktu di mana sistem CDC memproses catatan, dalam milidetik, UTC. Perbedaan antara `ts_ms` dan `source.ts_ms` merupakan ukuran lag replikasi.
`ts_ns`	Waktu di mana sistem CDC memproses catatan, dalam nanodetik, UTC.

Rincian format

Rincian berikut menjelaskan bagaimana format Aurora DSQL CDC mencatat. Rancang aplikasi Anda untuk menangani perilaku ini.

- Gambar setelah penuh untuk sisipan dan pembaruan. Aurora DSQL menyertakan status baris lengkap di `after` bidang untuk semua penulisan. `before`Bidang ini `null` untuk sisipan dan pembaruan. Saat ini sisipan dan pembaruan digunakan`op: "c"`, tetapi rilis berikutnya akan memancarkan `op: "u"` pembaruan. Rancang aplikasi Anda untuk digunakan `source.ts_ns` per kunci utama untuk pemesanan daripada mengandalkan op bidang untuk membedakan antara sisipan dan pembaruan.
- Post-change status baris saja. Catatan CDC mencakup status baris penuh setelah setiap perubahan. Status baris sebelum pembaruan tidak disertakan. Untuk menghapus pada tabel dengan kunci primer, `before` bidang berisi nilai kunci primer.
- Jenis numerik diserialkan sebagai string. Aurora DSQL membuat serial `numeric` dan `decimal` nilai sebagai string JSON untuk mempertahankan presisi yang tepat.
- Data biner dikodekan sebagai Base64. Aurora DSQL `bytea` mengkodekan nilai sebagai string Base64.
- Nilai floating-point dan numerik khusus. Aurora DSQL membuat serial NaN dan ±Infinity sebagai string,, dan. `"NaN"` `"Infinity"` `"-Infinity"` Ini berlaku untuk`real`,`double precision`, dan `numeric` jenis.

- Kolom JSON diserialkan sebagai string JSON. Aurora DSQL membuat serial nilai `json` kolom sebagai string JSON yang berisi teks JSON mentah yang disimpan di kolom. Parse nilai string di aplikasi Anda (misalnya, dengan `JSON.parse` in JavaScript atau `json.loads` di Python) untuk mengakses nilai JSON yang mendasarinya.
- Nilai overflow dipancarkan sebagai null. Jika nilai tidak dapat direpresentasikan dalam tipe JSON target selama serialisasi, Aurora DSQL memancarkan JSON untuk kolom itu. `null` Ini berlaku untuk `interval` nilai yang total mikrodetik melebihi rentang bilangan bulat bertanda 64-bit (± 9.223.372.036.854.775.807 mikrodetik, sekitar ±292.271 tahun). Rancang aplikasi Anda untuk menangani `null` nilai tak terduga di kolom yang tidak dapat dibatalkan dalam skema database.
- Catatan besar dibagi menjadi beberapa bagian. Jika rekaman melebihi batas ukuran rekaman Amazon Kinesis, Aurora DSQL membagi `after` gambar `before` atau yang terpengaruh menjadi fragmen dan mengirimkannya sebagai catatan Kinesis terpisah sehingga Anda masih menerima perubahan. Rancang aplikasi Anda untuk memasang kembali gambar. Lihat perinciannya di Menangani catatan besar.

Menangani catatan besar

Ketika JSON serial rekaman CDC melebihi 9 MiB, Aurora DSQL membagi gambar, memberikan beberapa catatan Kinesis. `before` and/or `after` Setiap rekaman berisi `type` bidang tingkat atas yang menunjukkan strukturnya: `full` untuk catatan lengkap, `chunked` untuk catatan utama yang mereferensikan fragmen, dan `fragment` untuk potongan individu dari gambar yang terpotong. Bidang `opsource`,`ts_ms`, dan `ts_ns` bidang pada catatan utama yang terpotong berperilaku sama seperti pada catatan lengkap. Rekaman yang sesuai dengan satu catatan Kinesis telah `type` ditetapkan `full` dan tidak memerlukan penanganan tambahan.

A `chunk_id` stabil di seluruh percobaan ulang. Jika Aurora DSQL mengirimkan ulang sebuah fragmen, itu akan `chunk_id` sama dengan pengiriman asli, sehingga aplikasi Anda dapat melanjutkan buffering di bawah pengenal yang sama tanpa menangani set sebagian dari upaya sebelumnya.

Rekor utama

Rekaman utama yang terpotong menggantikan level atas `before` atau `after` bidang untuk gambar terpisah dengan `chunked` objek yang menjelaskan cara memasangnya kembali. Setiap entri di bawah `chunked` memiliki `chunk_id` (pengenal yang menghubungkan fragmen ke catatan ini), `total_fragments` (jumlah fragmen yang membentuk gambar itu), dan `crc32c` (checksum CRC32C, sebagai string desimal, di atas teks gambar yang dipasang kembali). Jika satu gambar

sebaris dan yang lainnya terpotong, gambar sebaris masih muncul di tingkat atas sebagai nilai atau. `null`

```
{
    "type": "chunked",
    "op": "c",
    "before": null,
    "after": null,
    "source": {
        "version": "1.0",
        "ts_ms": 1705318200000,
        "ts_ns": 1705318200000000000,
        "txId": "ffthunp5stx6ffs2vyfqoatmfu",
        "schema": "public",
        "table": "order_items",
        "db": "postgres",
        "cluster": "cluster-id"
    },
    "chunked": {
        "after": {
            "chunk_id": "chunk-id",
            "total_fragments": 3,
            "crc32c": "2073618257"
        }
    },
    "ts_ms": 1705318200125,
    "ts_ns": 1705318200125483291
}
```

Catatan fragmen

Setiap fragmen adalah catatan Kinesis sendiri `type` dengan set `fragment` ke dan tiga bidang`chunk_id`: cocok dengan nilai dalam catatan `chunked.before.chunk_id` yang sesuai `chunked.after.chunk_id` atau pada catatan utama`index`, adalah posisi berbasis nol dari fragmen dalam gambar, `data` dan merupakan segmen dari teks JSON gambar yang dibagi UTF-8 pada batas karakter (nilai setiap fragmen `data` adalah string yang valid sendiri). UTF-8 Karena Aurora DSQL CDC menggunakan `UNORDERED` mode dan kunci partisi acak, fragmen dan catatan utama dapat tiba di pecahan yang berbeda dan dalam urutan apa pun. Untuk membaca semua fragmen, konsumsi semua pecahan pada aliran data Kinesis. Untuk informasi lebih lanjut tentang pemesanan pengiriman, lihatMemesan.

```
{
    "type": "fragment",
    "chunk_id": "chunk-id",
    "index": 0,
    "data": "partial-JSON-text"
}
```

Untuk memasang kembali gambar yang terlalu besar, buffer setiap rekaman dengan itu. `type` `fragment` `chunk_id` Ketika Anda menerima catatan utama dengan `typechunked`, tunggu sampai Anda memiliki `total_fragments` fragmen untuk setiap `chunk_id` direferensikan di bawah `chunked.before` atau`chunked.after`, urutkan fragmen dengan `index` naik, dan gabungkan string. `data` Hasil gabungan adalah asli `before` atau `after` objek sebagai teks JSON — menguraikannya untuk mengakses nilai kolom. Untuk memverifikasi integritas pengiriman, hitung CRC32C melalui string gabungan dan bandingkan hasilnya dengan atau. `chunked.before.crc32c` `chunked.after.crc32c`

Serialisasi tipe data

Tabel berikut menjelaskan bagaimana Aurora DSQL serialisasi setiap tipe data PostgreSQL dalam catatan CDC.

Jenis bilangan bulat

Jenis PostgreSQL	Representasi JSON	Contoh
`smallint` (`int2`)	Nomor JSON	`42`
`integer` (`int4`)	Nomor JSON	`1001`
`bigint` (`int8`)	Nomor JSON	`9223372036854775807`
`oid`	Nomor JSON (tidak ditandatangani)	`16384`

Nilai `bigint` di luar ±2^53 mungkin kehilangan presisi di lingkungan. JavaScript Gunakan BigInt atau pustaka presisi sewenang-wenang dalam kasus tersebut.

Floating-point jenis

Jenis PostgreSQL	Representasi JSON	Contoh	Catatan
`real` (`float4`)	Nomor JSON	`3.14159`	NaN dan ±Infinity diserialisasikan sebagai string`"NaN"`,`"Infinity"`, `"-Infinity"`
`double precision` (`float8`)	Nomor JSON	`3.141592653589793`	Penanganan nilai khusus yang sama seperti`real`.
`numeric` / `decimal`	String JSON	`"123.45"`	Selalu string untuk mempertahankan presisi yang tepat. NaN dan ±Infinity diserialisasikan sebagai string`"NaN"`,`"Infinity"`, `"-Infinity"`

Boolean

Jenis PostgreSQL	Representasi JSON	Contoh
`boolean`	Boolean JSON	`true` atau `false`

Jenis karakter

Jenis PostgreSQL	Representasi JSON	Contoh
`varchar` / `text`	String JSON	`"Hello, world!"`
`bpchar` (`char(n)`)	String JSON	`"ABC"`(spasi belakang dilucuti)
`name`	String JSON	`"pg_class"`
`"char"`(byte tunggal)	String JSON	`"A"`

Biner

Jenis PostgreSQL	Representasi JSON	Contoh
`bytea`	String JSON (Base64)	`"SGVsbG8gV29ybGQh"`

Jenis data tanggal dan waktu

Jenis PostgreSQL	Representasi JSON	Contoh	Catatan
`date`	Nomor JSON (hari sejak zaman Unix)	`19797`	`+infinity` dan `-infinity` direpresentasikan sebagai hitungan hari penjaga yang berasal dari aritmatika epoch-offset. Nilai-nilai ini tidak sesuai dengan tanggal kalender yang bermakna.
`time`	Nomor JSON (mikrodetik sejak tengah malam)	`52200123456`	

Jenis PostgreSQL	Representasi JSON	Contoh	Catatan
`timetz`	Nomor JSON (mikrodetik sejak tengah malam, UTC)	`52200123456`	Waktu lokal disesuaikan dengan UTC dengan menerapkan offset zona waktu yang disimpan (detik di sebelah barat UTC). Hasilnya dibungkus ke kisaran [0, 86400000000) mikrodetik.
`timestamp`	Nomor JSON (mikrodetik sejak zaman Unix)	`1710510600123456`	±Infinity memetakan nilai sentinel: `9223372036825200000` untuk dan untuk. `+infinity` `-9223372036832400000` `-infinity`
`timestamptz`	Nomor JSON (mikrodetik sejak zaman Unix)	`1710510600123456`	Disimpan dan dipancarkan dalam UTC. Nilai sentinel ±infinity yang sama dengan. `timestamp`

Jenis PostgreSQL	Representasi JSON	Contoh	Catatan
`interval`	Nomor JSON (perkiraan total mikrodetik)	`2802603000000`	Bulan diperkirakan 30,4375 hari (2.629.800 detik). Total dihitung sebagai`(months × 2,629,800 + days × 86,400) × 1,000,000 + microseconds` . Jika hasilnya melebihi rentang bilangan bulat bertanda 64-bit (± 9.223.372.036.854.775.807 mikrodetik, kira-kira ± 292.271 tahun), Aurora DSQL memancarkan JSON untuk kolom. `null`

Jenis lainnya

Jenis PostgreSQL	Representasi JSON	Contoh
`uuid`	String JSON (format hex standar 8-4-4-4-12)	`"550e8400-e29b-41d4-a716-446655440000"`
`oidvector`	Array kosong JSON	`[]`
`json`	String JSON yang berisi teks JSON mentah	`"{\"key\": \"value\"}"`

Nilai NULL

Untuk tipe data apa pun, nilai `NULL` kolom direpresentasikan sebagai JSON`null`.

Evolusi skema dalam catatan CDC

Saat Anda memodifikasi skema tabel—misalnya, dengan menambahkan, menjatuhkan, atau mengganti nama kolom—catatan CDC mencerminkan perubahan yang dimulai dari transaksi yang melakukan perubahan DDL. Catatan dari transaksi yang dilakukan sebelum perubahan DDL menggunakan skema sebelumnya. Contoh:

- Jika Anda menambahkan kolom, catatan dari transaksi sebelumnya tidak menyertakan kolom baru. Catatan dari transaksi penambahan dan seterusnya termasuk kolom baru.
- Jika Anda menjatuhkan kolom, catatan dari transaksi yang jatuh dan seterusnya tidak lagi menyertakan kolom itu.
- Jika Anda mengganti nama kolom, catatan dari transaksi penggantian nama selanjutnya menggunakan nama kolom baru.

Lacak perubahan skema di konsumen hilir Anda dengan memeriksa nama kolom yang ada di setiap rekaman dan bidang. `after` `before` `source.version`Bidang di setiap catatan mengidentifikasi format amplop CDC.

Pemantauan aliran

> **Important**
>
> Fitur ini disediakan sebagai AWS Pratinjau dan dapat berubah sewaktu-waktu. Untuk informasi selengkapnya, lihat bagian 2, Beta dan Pratinjau, di Ketentuan AWS Layanan. Untuk mempelajari lebih lanjut tentang harga untuk aliran CDC, lihat halaman harga Aurora DSQL.
> Sebelum ketersediaan umum, kami akan menambahkan jenis operasi baru (`"op": "u"`untuk pembaruan) ke muatan aliran Anda. Untuk memastikan aplikasi Anda menangani perubahan ini tanpa modifikasi, perlakukan op nilai yang tidak dikenal sebagai peningkatan dengan menerapkan payload. `after` Lihat Memahami catatan CDC untuk detail.

Ketika Aurora DSQL menemukan kesalahan saat mengirimkan catatan CDC, aliran beralih ke status. `IMPAIRED` Aliran yang terganggu terus memproses dan mengirimkan catatan lainnya—Aurora DSQL hanya mencoba kembali catatan yang gagal. Aurora DSQL mengukur lag replikasi dari catatan

tertua yang tidak terkirim, dan lag bertambah hingga Anda menyelesaikan masalah. Aurora DSQL mempertahankan perubahan yang tidak terkirim secara internal selama satu minggu.

Jika Anda menyelesaikan masalah mendasar dalam jendela ini, percobaan ulang berikutnya berhasil, status kesalahan dihapus, dan aliran beralih kembali ke. `ACTIVE` Perbaiki masalah eksternal (kebijakan IAM, AWS KMS kunci, kapasitas Amazon Kinesis, dan sebagainya) dan Aurora DSQL mencoba ulang secara otomatis.

Jika kelambatan replikasi melebihi ambang kegagalan, aliran bertransisi ke. `FAILED`

> **Important**
>
> Aliran yang gagal tidak dapat dipulihkan. Anda harus menghapus aliran yang gagal dan membuat yang baru.

Siklus hidup streaming

Aliran bertransisi melalui status berikut selama siklus hidupnya:

- **`CREATING`**— Aurora DSQL sedang menyiapkan aliran. Aurora DSQL belum mengirimkan catatan CDC.
- **`ACTIVE`**— Aliran beroperasi dan mengirimkan catatan CDC ke target.
- **`IMPAIRED`**— Aliran telah mengalami masalah yang membutuhkan tindakan Anda. Aurora DSQL mencoba kembali catatan yang gagal dengan backoff eksponensial, meskipun catatan lain dapat terus memberikan. Aurora DSQL mengukur lag replikasi dari catatan tertua yang tidak terkirim, dan lag bertambah hingga Anda menyelesaikan masalah. Aurora DSQL buffer perubahan yang tidak terkirim secara internal selama satu minggu. Lihat Referensi kode kesalahan.
- **`FAILED`**— Aliran telah mengalami kesalahan persisten dan tidak lagi mengirimkan catatan CDC. Aliran yang gagal tidak dapat dipulihkan dan Anda harus menghapusnya. Lihat Referensi kode kesalahan kondisi yang menyebabkan aliran memasuki status ini.
- **`DELETING`**— Aurora DSQL menghapus sumber daya aliran.
- **`DELETED`**— Aurora DSQL telah menghapus aliran. Setelah penghapusan selesai, mengembalikan a. `GetStream` `ResourceNotFoundException`

Hubungi `GetStream` untuk melihat status streaming kapan saja. Ketika aliran `IMPAIRED` atau`FAILED`, responsnya menyertakan `statusReason` objek dengan kode kesalahan dan stempel

waktu. Untuk detail selengkapnya tentang bidang `GetStream` respons, lihat GetStreamdi Referensi API Amazon Aurora DSQL.

Memecahkan masalah aliran yang terganggu atau gagal

Ikuti langkah-langkah ini ketika aliran CDC menjadi terganggu atau gagal. Jika alirannya`FAILED`, Anda tidak dapat memulihkannya—hapus aliran, selesaikan masalah mendasar, dan buat yang baru.

1. Dapatkan status streaming. Panggil `GetStream` dan verifikasi `status` bidang. Jika statusnya`ACTIVE`, alirannya sehat.

```
aws dsql get-stream \
  --cluster-identifier cluster-id \
  --stream-identifier stream-id \
  --region region
```

2. Baca kode kesalahan. Jika statusnya `IMPAIRED` atau`FAILED`, responsnya termasuk `statusReason` objek. `error`Bidang berisi kode kesalahan.

```
{
    "status": "IMPAIRED",
    "statusReason": {
        "error": "KINESIS_THROUGHPUT_EXCEEDED",
        "updatedAt": "2025-01-15T14:30:00Z"
    }
}
```

3. Ikuti remediasi. Jika alirannya`IMPAIRED`, cari kode kesalahan di tabel berikut dan terapkan perbaikan yang disarankan. Aurora DSQL mencoba ulang secara otomatis setelah Anda menyelesaikan masalah mendasar. Jika alirannya`FAILED`, hapus, selesaikan masalah, dan buat aliran baru.

Referensi kode kesalahan

Tabel berikut menjelaskan setiap kode kesalahan, penyebabnya, apakah aliran dapat pulih, dan langkah-langkah untuk mengatasinya.

Kode kesalahan	Penyebab	Dapat dipulihkan?	Cara mengatasinya
KINESIS_THROUGHPUT_EXCEEDED	Aliran data Kinesis Anda melebihi batas throughputnya, atau operasi AWS KMS enkripsi yang dibatasi pada aliran data Kinesis, dan kelambatan replikasi telah bertambah.	Ya	Tingkatkan jumlah pecahan pada aliran data Kinesis Anda, atau beralih ke mode kapasitas sesuai permintaan. Jika aliran data Kinesis menggunakan kunci yang dikelola AWS KMS pelanggan, verifikasi bahwa kuota permintaan kunci cukup besar. Setelah Anda meningkatkan kapasitas, Aurora DSQL mencoba ulang secara otomatis.
KINESIS_STREAM_NOT_FOUND	Aliran data Kinesis target tidak ada lagi.	Tidak	Aliran transisi langsung keFAILED. Hapus aliran CDC dan buat yang baru menunjuk ke aliran data Kinesis yang valid.
ROLE_ACCESS_DENIED	Aurora DSQL tidak dapat mengasumsikan peran IAM yang ditentukan dalam definisi target. AWS STS AssumeRole Panggilan itu kembaliAccessDenied .	Ya	Verifikasi kebijakan kepercayaan peran memungkinkan prinsipal layanan Aurora DSQL (dsql.amazonaws.com) untuk mengasumsikannya. Verifikasi aws:Sourc

Kode kesalahan	Penyebab	Dapat dipulihkan?	Cara mengatasinya
			`eAccount` dan `aws:SourceArn` kondisi cocok dengan cluster Anda. Lihat perincian nya di Kebijakan kepercayaan peran layanan. Setelah Anda memperbaiki kebijakan kepercaya an, Aurora DSQL mencoba ulang secara otomatis.
`KINESIS_A CCESS_DENIED`	Peran yang diasumsik an tidak memiliki izin untuk menulis ke aliran data Kinesis. Kinesis kembali. `AccessDen iedException`	Ya	Tambahkan `kinesis:P utRecord` dan `kinesis:P utRecords` izin ke kebijakan peran untuk aliran data Kinesis target Nama Sumber Daya Amazon (ARN). Setelah Anda memperbaiki kebijakan, Aurora DSQL mencoba ulang secara otomatis.

Kode kesalahan	Penyebab	Dapat dipulihkan?	Cara mengatasinya
`KINESIS_KMS_ACCESS_DENIED`	Peran yang diasumsikan tidak memiliki izin untuk menggunakan AWS KMS kunci yang mengenkripsi aliran data Kinesis. Kesalahan ini mencakup penolakan AWS KMS akses dan status kunci yang tidak valid.	Ya	Verifikasi peran memiliki `kms:GenerateDataKey` izin pada AWS KMS kunci yang digunakan aliran data Kinesis. Juga verifikasi bahwa AWS KMS kunci dalam status diaktifkan dan valid. Kunci ini adalah kunci enkripsi pada aliran data Kinesis, bukan kunci cluster. AWS KMS Lihat perinciannya di Kebijakan izin peran layanan. Setelah Anda memperbaiki izin atau status kunci, Aurora DSQL mencoba ulang secara otomatis.

Kode kesalahan	Penyebab	Dapat dipulihkan?	Cara mengatasinya
`KINESIS_OVERSIZE_RECORD`	Catatan CDC melebihi ukuran rekaman maksimum yang dikonfigurasi pada aliran data Kinesis.	Ya	Tingkatkan `MaxRecordSizeInKiB` aliran data Kinesis ke `10240` (10 MiB). Anda dapat memperbarui pengaturan ini pada aliran data Kinesis yang ada tanpa menghapusnya. Setelah Anda meningkatkan batas, Aurora DSQL mencoba ulang rekaman besar secara otomatis dan transisi aliran kembali ke. `ACTIVE`
`CLUSTER_CMK_INACCESSIBLE`	Kunci yang dikelola AWS KMS pelanggan yang mengenkripsi cluster Aurora DSQL tidak dapat diakses.	Ya	Verifikasi kebijakan AWS KMS kunci dan status kunci. Re-enable atau mengembalikan akses ke kunci. Setelah kunci dapat diakses lagi, aliran beralih kembali ke`ACTIVE`.

Tabel sebelumnya mencantumkan setiap `StreamFailureErrorCode` nilai. Untuk detail tentang bidang `statusReason` respons, lihat GetStreamdi Referensi API Amazon Aurora DSQL.

Memulihkan aliran yang terganggu

Sebagian besar kesalahan pertama-tama mentransisikan aliran ke`IMPAIRED`. Aliran yang terganggu terus memproses catatan lain dan mencoba ulang catatan yang gagal secara otomatis. `FAILED`Streaming tidak dapat dipulihkan — Anda harus menghapusnya dan membuat yang baru.

- Untuk kesalahan yang dapat dipulihkan: perbaiki masalah eksternal (kebijakan IAM, kunci, kapasitas Kinesis AWS KMS , atau batas ukuran catatan Kinesis). Percobaan ulang berikutnya yang berhasil menghapus status kesalahan dan mentransisikan aliran kembali ke. `ACTIVE`
- Untuk**`KINESIS_STREAM_NOT_FOUND`**: transisi aliran langsung ke`FAILED`. Hapus aliran yang gagal dan buat yang baru menunjuk ke aliran data Kinesis yang valid.

Untuk semua kode kesalahan lainnya, jika kelambatan replikasi melebihi ambang kegagalan sebelum Anda menyelesaikan masalah, aliran akan beralih dari `IMPAIRED` ke. `FAILED` Aliran yang gagal tidak dapat bertransisi kembali ke`ACTIVE`. Hapus aliran yang gagal, selesaikan masalah mendasar, dan buat yang baru.

Pemantauan kesehatan aliran

Gunakan CloudWatch metrik dan `GetStream` API untuk memantau kesehatan streaming. CloudWatchmetrik memberikan visibilitas berkelanjutan ke kinerja pipa CDC, dan `GetStream` menyediakan kode kesalahan spesifik ketika aliran terganggu atau gagal.

Untuk daftar lengkap metrik CDC, termasuk,, `IsImpairedBehindSourceLag`, dan `PublishedBytesPublishedRecords`, lihat. CloudWatch metrik untuk aliran CDC Untuk detail selengkapnya tentang bidang `GetStream` respons, lihat GetStreamdi Referensi API Amazon Aurora DSQL.

CloudWatch metrik untuk aliran CDC

Gunakan CloudWatch metrik berikut untuk memantau kesehatan dan throughput setiap aliran CDC. Aurora DSQL menerbitkan metrik ini di namespace dengan dimensi dan`AWS/AuroraDSQL`. `ClusterId` `StreamId` Metrik terakhir adalah metrik Amazon Kinesis standar di `AWS/Kinesis` namespace yang mengukur lag pembacaan hilir.

> **Note**
>
> Aurora DSQL juga menerbitkan `StreamDPU` metrik dan di `AWS/AuroraDSQL` namespace untuk `BytesStreamed` penggunaan dan pelacakan penagihan. Untuk deskripsi, lihatMetrik aliran CDC.

Nama metrik	Statistik yang berguna	Deskripsi
`IsImpaired`	Maksimum	Menunjukkan apakah aliran terganggu. Nilainya adalah `1` ketika aliran dalam `IMPAIRED` keadaan, dan `0` ketika aliran sehat. Aurora DSQL memancarkan metrik ini secara terus menerus untuk setiap aliran aktif atau terganggu. Gunakan metrik ini untuk membuat CloudWatch alarm yang memberi tahu Anda saat aliran menjadi terganggu.
`BehindSourceLag`	Rata-rata	Penundaan, dalam milidetik, antara saat transaksi dilakukan di Aurora DSQL dan ketika sistem CDC memproses catatan yang dihasilkan. Nilai yang meningkat menunjukkan bahwa pipa CDC berada di belakang beban kerja tulis.
`PublishedBytes`	Jumlah	Total byte catatan CDC yang ditulis Aurora DSQL ke target selama periode tersebut. Gunakan metrik ini bersama dengan jumlah

Nama metrik	Statistik yang berguna	Deskripsi
		pecahan Kinesis Anda untuk menentukan apakah Anda telah menyediakan kapasitas tulis yang cukup.
`PublishedRecords`	Jumlah	Jumlah total catatan CDC yang ditulis Aurora DSQL ke target selama periode tersebut. Setiap perubahan baris yang berkomitmen menghasilkan satu catatan.
`GetRecords.IteratorAgeMilliseconds` (`AWS/Kinesis`)	Rata-rata	Metrik Kinesis standar yang melaporkan usia rekaman terakhir yang dibaca dari aliran data Kinesis oleh aplikasi hilir Anda, dalam milidetik. Gunakan `StreamName` dimensi. Nilai yang meningkat menunjukkan bahwa aplikasi hilir Anda tidak dapat mengikuti laju di mana Aurora DSQL menulis catatan CDC ke Kinesis.

Tab Pemantauan konsol Aurora DSQL menunjukkan nilai latensi end-to-end Rata-rata yang menggabungkan **`BehindSourceLag`** (latensi sumber CDC) dan (lag pembaca Kinesis). `GetRecords.IteratorAgeMilliseconds` Nilai gabungan ini mewakili total penundaan dari komit database ke pembacaan hilir.

Memantau praktik terbaik

Gunakan praktik berikut untuk mendeteksi dan menyelesaikan masalah pipeline CDC sebelum memengaruhi sistem hilir Anda.

Atur alarm **`BehindSourceLag`**

Buat CloudWatch alarm yang menyala saat `BehindSourceLag` melebihi ambang batas yang penting bagi beban kerja Anda. Misalnya, atur 60 detik untuk target latensi satu menit. Peningkatan berkelanjutan dalam metrik ini berarti pipa CDC tertinggal. Jika lag mencapai ambang kegagalan, aliran beralih ke FAILED. Menangkap tren memberi Anda waktu untuk meningkatkan kapasitas Kinesis atau menyelidiki kemacetan throughput sebelum aliran menurun.

Pantau **`GetRecords.IteratorAgeMilliseconds`**di sisi Kinesis

Bahkan ketika Aurora DSQL mengirimkan catatan tepat waktu, aplikasi hilir Anda dapat tertinggal. Buat CloudWatch alarm aktif `GetRecords.IteratorAgeMilliseconds` (di `AWS/Kinesis` namespace, dimensi`StreamName`) untuk mendeteksi lag hilir secara independen. Jika metrik ini naik dan `BehindSourceLag` tetap datar, kemacetannya ada di aplikasi hilir Anda, bukan di Aurora DSQL.

Lacak **`PublishedBytes`**terhadap kapasitas pecahan Kinesis

Setiap pecahan Kinesis mendukung hingga 1 MiB per detik untuk penulisan. Bandingkan `PublishedBytes` Jumlah per menit dengan total kapasitas tulis pecahan Anda (jumlah pecahan × 60 MiB per menit). Jika penggunaan mendekati 80 persen, tambahkan pecahan atau beralih ke mode kapasitas sesuai permintaan sebelum pemicu pelambatan. `KINESIS_THROUGHPUT_EXCEEDED`

Alarm menyala **`IsImpaired`**untuk deteksi kerusakan instan

Buat CloudWatch alarm yang menyala ketika `IsImpaired` Maksimum lebih besar dari atau sama dengan 1 untuk satu periode evaluasi. Ini memberi Anda sinyal langsung ketika aliran memasuki `IMPAIRED` status, tanpa melakukan polling API. Setelah alarm menyala, panggil `GetStream` untuk membaca `statusReason.error` lapangan dan ikuti langkah-langkah remediasi. Memecahkan masalah aliran yang terganggu atau gagal

Polling **`GetStream`**untuk status terperinci

`IsImpaired`Metrik memberi tahu Anda bahwa aliran terganggu, tetapi `GetStream` API menyediakan kode kesalahan dan stempel waktu tertentu. Polling `GetStream` pada jadwal (misalnya, setiap lima menit) atau sebagai respons terhadap `IsImpaired` alarm. `statusReason.error`Lapangan memberi tahu Anda apa yang salah. Pasangkan ini dengan langkah-langkah pemecahan masalah Memecahkan masalah aliran yang terganggu atau gagal untuk resolusi cepat.

Gunakan dasbor untuk mengkorelasikan metrik

Buat CloudWatch dasbor yang menunjukkan`IsImpaired`,`BehindSourceLag`,`PublishedRecords`,`PublishedBytes`, dan

`GetRecords.IteratorAgeMilliseconds` berdampingan. Mengkorelasikan metrik ini membantu Anda membedakan antara masalah pipa CDC (naik`BehindSourceLag`) dan masalah pembacaan hilir (meningkat `IteratorAge` dengan stabil). `BehindSourceLag`

Pemantauan dan pencatatan untuk Aurora DSQL

Pemantauan dan pencatatan adalah bagian penting untuk menjaga keandalan, ketersediaan, dan kinerja sumber daya Amazon Aurora DSQL Anda. Anda harus memantau dan mengumpulkan data logging dari semua bagian sumber daya Aurora DSQL Anda sehingga Anda dapat dengan mudah men-debug kegagalan multi-titik.

- Amazon CloudWatch memantau AWS sumber daya Anda dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat mengumpulkan dan melacak metrik, membuat dasbor yang disesuaikan, dan mengatur alarm yang memberi tahu Anda atau mengambil tindakan saat metrik tertentu mencapai ambang batas yang ditentukan. Misalnya, Anda dapat CloudWatch melacak penggunaan CPU atau metrik lain dari instans Amazon EC2 Anda dan secara otomatis meluncurkan instans baru bila diperlukan. Untuk informasi selengkapnya, lihat Panduan CloudWatch Pengguna Amazon.
- AWS CloudTrailmenangkap panggilan API dan peristiwa terkait yang dibuat oleh atau atas nama Anda Akun AWS dan 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.

Memantau Aurora DSQL dengan Amazon CloudWatch

Pantau penggunaan Aurora DSQL CloudWatch, yang mengumpulkan data mentah dan memprosesnya menjadi metrik yang dapat dibaca, mendekati waktu nyata. CloudWatch menyimpan statistik ini selama 15 bulan, membantu Anda mendapatkan perspektif yang lebih baik tentang aplikasi web atau kinerja layanan Anda. Atur alarm untuk mengawasi ambang batas tertentu dan mengirim pemberitahuan atau mengambil tindakan saat terpenuhi. Tinjau metrik Penggunaan dan Pengamatan berikut yang tersedia untuk Aurora DSQL.

Untuk informasi selengkapnya, lihat Panduan CloudWatch Pengguna Amazon.

Observabilitas dan kinerja

Tabel ini menguraikan metrik observabilitas untuk Aurora DSQL. Ini mencakup metrik untuk melacak transaksi hanya-baca dan total untuk memberikan karakterisasi beban kerja secara keseluruhan. Metrik yang dapat ditindaklanjuti seperti batas waktu kueri dan tingkat konflik OCC disertakan untuk

membantu mengidentifikasi masalah kinerja dan konflik konkurensi. Session-related metrik, baik aktif maupun total, menawarkan wawasan tentang beban saat ini pada sistem.

CloudWatch Nama Metrik	Metrik	Unit	Deskripsi
ReadOnlyTransactions	Read-only transaksi	none	Jumlah transaksi read-only
TotalTransactions	Total transaksi	none	Jumlah total transaksi yang dijalankan pada sistem, termasuk transaksi read-only.
QueryTimeouts	Waktu kueri	none	Jumlah kueri yang telah habis waktunya karena mencapai waktu transaksi maksimum
OccConflicts	Konflik OCC	none	Jumlah transaksi dibatalkan karena tingkat kunci OCC
CommitLatency	Komit Latensi	milidetik	Waktu yang dihabiskan oleh fase komit eksekusi kueri (P50)
BytesWritten	Bytes Ditulis	byte	Byte ditulis ke penyimpanan
BytesRead	Bytes Baca	byte	Byte dibaca dari penyimpanan
ComputeTime	Waktu komputasi QP	milidetik	Waktu jam dinding QP
ClusterStorageSize	Ukuran Penyimpanan Cluster	byte	Ukuran cluster

Aurora DSQL juga menerbitkan metrik berikut di bawah namespace. AWS/Usage Gunakan metrik ini untuk memantau upaya koneksi dan jumlah koneksi aktif untuk cluster Anda.

CloudWatch Nama Metrik	Tipe	Sumber daya	ResourceId	Layanan	Unit	Deskripsi
ResourceCount	Sumber daya	ClusterConnectionCount	kelompok/<cluster-id>	AuroraDSql	Hitungan	Jumlah koneksi aktif untuk sebuah cluster.
CallCount	API	DbConnectAdmin atau DbConnect	kelompok/<cluster-id>	AuroraDSql	Hitungan	Jumlah panggilan API untuk operasi koneksi database (DbConnectAdmin atau DbConnect).

Metrik penggunaan

Aurora DSQL mengukur semua aktivitas berbasis permintaan, seperti pemrosesan kueri, pembacaan, dan penulisan, menggunakan satu unit penagihan dinormalisasi yang disebut Distributed Processing Unit (DPU).

CloudWatch Nama Metrik	Metrik	Dimensi: ResourceId	Unit	Deskripsi
DitulisPU	Tulis Unit	<cluster-id>	DPU	Perkiraan komponen penggunaan

CloudWatch Nama Metrik	Metrik	Dimensi: ResourceId	Unit	Deskripsi
				aktif tulis dari penggunaan DPU cluster Aurora DSQL Anda.
MultiRegionWriteDPU	Multi-Region Tulis Unit	<cluster-id>	DPU	Berlaku untuk Multi-Region cluster: Memperkirakan komponen penggunaan aktif penulisan Multi-wilayah dari penggunaan DPU cluster Aurora DSQL Anda.
bacaDPU	Baca Unit	<cluster-id>	DPU	Perkiraan komponen penggunaan aktif baca dari penggunaan DPU cluster Aurora DSQL Anda.

CloudWatch Nama Metrik	Metrik	Dimensi: ResourceId	Unit	Deskripsi
ComputedPU	Unit Komputasi	<cluster-id>	DPU	Perkiraan komponen penggunaan aktif komputasi dari penggunaan DPU cluster Aurora DSQL Anda.
TotalDPU	Jumlah Unit	<cluster-id>	DPU	Perkiraan total komponen penggunaan aktif dari penggunaan DPU cluster Aurora DSQL Anda.

Metrik aliran CDC

Aurora DSQL menerbitkan metrik berikut untuk aliran pengambilan data perubahan (CDC). Metrik ini menggunakan `StreamId` dimensi `ClusterId` dan, sehingga Anda dapat memantau setiap aliran CDC secara independen. Untuk informasi selengkapnya tentang aliran CDC, lihat Mengubah aliran pengambilan data (CDC).

CloudWatch Nama Metrik	Metrik	Unit	Deskripsi
IsImpaired	Terganggu	none	Menunjukkan apakah aliran terganggu. Nilainya adalah 1 ketika aliran dalam `IMPAIRED` keadaan, dan `0` ketika aliran

CloudWatch Nama Metrik	Metrik	Unit	Deskripsi
			sehat. Gunakan metrik ini untuk membuat CloudWatch alarm yang memberi tahu Anda saat aliran menjadi terganggu.
PublishedBytes	Byte yang diterbitkan	byte	Jumlah total byte yang ditulis Aurora DSQL ke aliran data Kinesis target.
PublishedRecords	Catatan yang diterbitkan	none	Jumlah catatan CDC yang Aurora DSQL tulis ke target Kinesis data stream.
BehindSourceLag	Di belakang lag sumber	milidetik	Penundaan, dalam milidetik, antara saat transaksi dilakukan di Aurora DSQL dan ketika sistem CDC memproses catatan yang dihasilkan. Nilai yang meningkat menunjukkan bahwa pipa CDC berada di belakang beban kerja tulis. Jika lag tumbuh di luar ambang kegagalan, aliran bertransisi ke`FAILED`.

CloudWatch Nama Metrik	Metrik	Unit	Deskripsi
BytesStreamed	Bytes dialirkan	byte	Total byte dialirkan melalui pipa CDC untuk tujuan penagihan. Metrik ini mencerminkan volume data yang digunakan untuk menghitung biaya streaming.
StreamDPU	Aliran DPU	DPU	Unit Pemrosesan Terdistribusi (DPU) yang dikonsumsi oleh aliran CDC. Metrik ini mencerminkan biaya pemrosesan data perubahan streaming.

Pencatatan operasi Aurora DSQL menggunakan AWS CloudTrail

Amazon Aurora DSQL terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau. Layanan AWS Ada dua jenis peristiwa di CloudTrail: peristiwa manajemen dan peristiwa data. Peristiwa manajemen dipancarkan untuk mengaudit perubahan konfigurasi AWS sumber daya. Peristiwa data menangkap penggunaan AWS sumber daya biasanya di bidang data layanan.

CloudTrail menangkap semua panggilan API untuk Aurora DSQL sebagai peristiwa. Aurora DSQL merekam aktivitas konsol sebagai peristiwa manajemen. Ini juga menangkap upaya koneksi yang diautentikasi ke cluster sebagai peristiwa data.

Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Aurora DSQL, alamat IP dari mana permintaan dibuat, kapan dibuat, identitas pengguna yang membuat permintaan, dan detail tambahan.

CloudTrail diaktifkan secara default di Akun AWS saat Anda membuat akun dan Anda memiliki akses ke riwayat CloudTrail Acara. Riwayat CloudTrail Acara menyediakan catatan yang dapat dilihat, dapat dicari, dapat diunduh, dan tidak dapat diubah dari 90 hari terakhir dari peristiwa manajemen yang direkam dalam file. Wilayah AWS Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna. Tidak ada CloudTrail biaya untuk merekam riwayat Acara.

Untuk membuat catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk peristiwa untuk Aurora DSQL, buat jejak atau penyimpanan data acara AWS CloudTrail Lake (solusi penyimpanan dan analisis terpusat untuk acara). AWS CloudTrail Untuk informasi selengkapnya tentang membuat jejak, lihat Bekerja dengan CloudTrail jalur. Untuk mempelajari cara menyiapkan dan mengelola penyimpanan data acara, lihat Penyimpanan data acara CloudTrail Danau.

Acara manajemen Aurora DSQL di CloudTrail

CloudTrail Acara manajemen memberikan informasi tentang operasi manajemen yang dilakukan pada sumber daya di AWS akun Anda. Ini juga dikenal sebagai operasi bidang kontrol. Secara default, CloudTrail menangkap peristiwa manajemen dalam riwayat Acara.

Amazon Aurora DSQL mencatat semua operasi pesawat kontrol Aurora DSQL sebagai peristiwa manajemen. Untuk daftar operasi bidang kontrol Amazon Aurora DSQL yang dicatat oleh Aurora DSQL CloudTrail, lihat referensi Aurora DSQL API.

Kontrol log pesawat

Amazon Aurora DSQL mencatat operasi bidang kontrol Aurora DSQL berikut sebagai peristiwa manajemen. CloudTrail

- CreateCluster
- DeleteCluster
- GetCluster
- GetVpcEndpointServiceName
- ListClusters
- ListTagsForResource
- TagResource
- UntagResource

- UpdateCluster

Log aliran CDC

Amazon Aurora DSQL mencatat operasi aliran CDC berikut sebagai peristiwa manajemen. CloudTrail Untuk informasi selengkapnya tentang aliran CDC, lihat Mengubah aliran pengambilan data (CDC).

- CreateStream
- DeleteStream
- GetStream
- ListStreams

Cadangkan dan pulihkan log

Amazon Aurora DSQL mencatat operasi pencadangan dan pemulihan Aurora DSQL berikut sebagai peristiwa manajemen. CloudTrail

- `StartBackupJob`
- `StopBackupJob`
- `GetBackupJob`
- `StartRestoreJob`
- `StopRestoreJob`
- `GetRestoreJob`

Untuk informasi lebih lanjut tentang melindungi cluster Aurora DSQL Anda menggunakan, lihat. AWS BackupCadangkan dan pulihkan untuk Amazon Aurora DSQL

Log AWS KMS

Amazon Aurora DSQL mencatat AWS KMS operasi berikut sebagai peristiwa manajemen. CloudTrail

- `GenerateDataKey`
- `Decrypt`

Untuk mempelajari lebih lanjut tentang cara CloudTrail log melacak permintaan yang dikirim Aurora DSQL AWS KMS atas nama Anda, lihat. Memantau interaksi Aurora DSQL dengan AWS KMS

Peristiwa data Aurora DSQL di CloudTrail

CloudTrail Peristiwa data biasanya memberikan informasi tentang operasi sumber daya yang dilakukan pada atau di sumber daya. Ini juga digunakan untuk menangkap operasi pesawat data layanan. Peristiwa data sering kali merupakan aktivitas bervolume tinggi. Secara default, CloudTrail tidak mencatat peristiwa data. Riwayat CloudTrail peristiwa tidak merekam peristiwa data.

Untuk informasi selengkapnya tentang cara mencatat peristiwa data, lihat Mencatat peristiwa data dengan Konsol Manajemen AWS dan Logging peristiwa data dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

Biaya tambahan berlaku untuk peristiwa data. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga.

Untuk Aurora DSQL, CloudTrail menangkap setiap upaya koneksi yang dilakukan ke cluster Aurora DSQL sebagai peristiwa data. Tabel berikut mencantumkan jenis sumber daya Aurora DSQL yang dapat Anda log peristiwa data. Kolom Jenis sumber daya (konsol) menunjukkan nilai yang akan dipilih dari daftar Jenis sumber daya di CloudTrail konsol. Kolom nilai resources.type menunjukkan **`resources.type`** nilai, yang akan Anda tentukan saat mengonfigurasi penyeleksi acara lanjutan menggunakan API atau. AWS CLI CloudTrail CloudTrailKolom API Data yang dicatat ke menampilkan panggilan API yang dicatat CloudTrail untuk jenis sumber daya.

Jenis sumber daya (konsol)	nilai resources.type	API data masuk CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Anda dapat mengonfigurasi pemilih acara lanjutan untuk memfilter pada `eventName` dan `resources.`ARN bidang untuk mencatat peristiwa yang hanya difilter. Untuk informasi selengkapnya tentang bidang ini, lihat AdvancedFieldSelectordi Referensi AWS CloudTrail API.

Contoh berikut menunjukkan bagaimana menggunakan untuk mengkonfigurasi AWS CLI untuk menerima peristiwa data `dsql-data-events-trail` untuk Aurora DSQL.

```
aws cloudtrail put-event-selectors \
--region us-east-1 \
--trail-name dsql-data-events-trail \
--advanced-event-selectors '[{
```

```
"Name": "Log DSQL Data Events",
    "FieldSelectors": [
       { "Field": "eventCategory", "Equals": ["Data"] },
       { "Field": "resources.type", "Equals": ["AWS::DSQL::Cluster"] } ]}]'
```

Keamanan di Amazon Aurora DSQL

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 menjelaskan hal ini sebagai keamanan dari cloud dan keamanan dalam 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. Third-party auditor secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program Kepatuhan Program AWS Kepatuhan Program AWS . Untuk mempelajari tentang program kepatuhan yang berlaku untuk Amazon Aurora DSQL, lihat AWS Layanan dalam Lingkup oleh Program Kepatuhan dalam Lingkup oleh Program Kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, yang mencakup sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Aurora DSQL. Topik berikut menunjukkan cara mengkonfigurasi Aurora DSQL untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan AWS layanan lain yang membantu Anda memantau dan mengamankan sumber daya Aurora DSQL Anda.

Topik

- AWS kebijakan terkelola untuk Amazon Aurora DSQL
- Perlindungan data di Amazon Aurora DSQL
- Enkripsi data untuk Amazon Aurora DSQL
- Manajemen identitas dan akses untuk Aurora DSQL
- Resource-based kebijakan untuk Aurora DSQL
- Menggunakan peran terkait layanan di Aurora DSQL
- Menggunakan kunci kondisi IAM dengan Amazon Aurora DSQL
- Respon insiden di Amazon Aurora DSQL
- Validasi kepatuhan untuk Amazon Aurora DSQL

- Ketahanan di Amazon Aurora DSQL
- Keamanan Infrastruktur di Amazon Aurora DSQL
- Analisis konfigurasi dan kerentanan di Amazon Aurora DSQL
- Cross-service pencegahan wakil bingung
- Praktik terbaik keamanan untuk Aurora DSQL

AWS kebijakan terkelola untuk Amazon Aurora DSQL

Kebijakan AWS terkelola adalah kebijakan mandiri yang dibuat dan dikelola oleh AWS. AWS Kebijakan terkelola dirancang untuk memberikan izin bagi banyak kasus penggunaan umum sehingga Anda dapat mulai menetapkan izin kepada pengguna, grup, dan peran.

Perlu diingat bahwa kebijakan AWS terkelola mungkin tidak memberikan izin hak istimewa paling sedikit untuk kasus penggunaan spesifik Anda karena tersedia untuk digunakan semua pelanggan. AWS Kami menyarankan Anda untuk mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola pelanggan yang khusus untuk kasus penggunaan Anda.

Anda tidak dapat mengubah izin yang ditentukan dalam kebijakan AWS terkelola. Jika AWS memperbarui izin yang ditentukan dalam kebijakan AWS terkelola, pembaruan akan memengaruhi semua identitas utama (pengguna, grup, dan peran) yang dilampirkan kebijakan tersebut. AWS kemungkinan besar akan memperbarui kebijakan AWS terkelola saat baru Layanan AWS diluncurkan atau operasi API baru tersedia untuk layanan yang ada.

Untuk informasi selengkapnya, lihat Kebijakan terkelola AWS dalam Panduan Pengguna IAM.

AWS kebijakan terkelola: AmazonAuroraDSQLFullAccess

Anda dapat melampirkan `AmazonAuroraDSQLFullAccess` ke pengguna, grup, dan peran Anda.

Kebijakan ini memberikan izin yang memungkinkan akses administratif penuh ke Aurora DSQL. Prinsipal dengan izin ini dapat:

- Buat, hapus, dan perbarui cluster Aurora DSQL, termasuk klaster Multi-wilayah
- Mengelola kebijakan inline cluster (membuat, melihat, memperbarui, dan menghapus kebijakan)
- Membuat, memperbarui, menghapus, dan mengelola aliran CDC untuk cluster

- Menambahkan dan menghapus tag dari cluster dan aliran CDC
- Buat daftar cluster dan lihat informasi tentang cluster individu
- Lihat tag yang dilampirkan pada cluster Aurora DSQL
- Connect ke database sebagai pengguna manapun, termasuk admin
- Lulus peran IAM ke aliran CDC untuk pengiriman data ke tujuan target
- Lakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan pencadangan dan pemulihan
- Gunakan AWS KMS kunci yang dikelola pelanggan untuk enkripsi klaster
- Lihat metrik apa pun dari CloudWatch akun mereka
- Gunakan AWS Fault Injection Service (AWS FIS) untuk menyuntikkan kegagalan ke cluster Aurora DSQL untuk pengujian toleransi kesalahan
- Buat peran terkait layanan untuk `dsql.amazonaws.com` layanan, yang diperlukan untuk membuat cluster

Detail izin

Kebijakan ini mencakup izin berikut.

- `dsql`— memberikan kepala sekolah akses penuh ke Aurora DSQL.
- `cloudwatch`— memberikan izin untuk mempublikasikan titik data metrik ke Amazon CloudWatch.
- `iam`— memberikan izin untuk membuat peran terkait layanan dan meneruskan peran ke aliran CDC untuk pengiriman data.
- `backup and restore`— memberikan izin untuk memulai, menghentikan, dan memantau pencadangan dan pemulihan pekerjaan untuk cluster Aurora DSQL.
- `kms`— memberikan izin yang diperlukan untuk memvalidasi akses ke kunci yang dikelola pelanggan yang digunakan untuk enkripsi cluster Aurora DSQL saat membuat, memperbarui, atau menghubungkan ke cluster.
- `fis`— memberikan izin untuk menggunakan (AWS Fault Injection Service AWS FIS) untuk menyuntikkan kegagalan ke cluster Aurora DSQL untuk pengujian toleransi kesalahan.

Anda dapat menemukan `AmazonAuroraDSQLFullAccess` kebijakan di konsol IAM dan di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonAuroraDSQLReadOnlyAccess

Anda dapat melampirkan `AmazonAuroraDSQLReadOnlyAccess` ke pengguna, grup, dan peran Anda.

Memungkinkan akses baca ke Aurora DSQL. Prinsipal dengan izin ini dapat mencantumkan kluster dan melihat informasi tentang kluster individu. Mereka dapat melihat tag yang dilampirkan ke cluster Aurora DSQL dan aliran CDC, melihat informasi tentang aliran CDC individu, dan melihat kebijakan inline cluster. Mereka dapat mengambil dan melihat metrik apa pun dari akun CloudWatch Anda.

Detail izin

Kebijakan ini mencakup izin berikut.

- `dsql`— memberikan izin baca saja untuk semua sumber daya di Aurora DSQL.
- `cloudwatch`— memberikan izin untuk mengambil jumlah batch data CloudWatch metrik dan melakukan matematika metrik pada data yang diambil
- `iam`— memberikan izin untuk meneruskan peran ke aliran CDC untuk konfigurasi pengiriman data.

Anda dapat menemukan `AmazonAuroraDSQLReadOnlyAccess` kebijakan di konsol IAM dan Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonAuroraDSQLConsoleFullAccess

Anda dapat melampirkan `AmazonAuroraDSQLConsoleFullAccess` ke pengguna, grup, dan peran Anda.

Memungkinkan akses administratif penuh ke Amazon Aurora DSQL melalui file. Konsol Manajemen AWS Prinsipal dengan izin ini dapat:

- Buat, hapus, dan perbarui cluster Aurora DSQL, termasuk cluster Multi-region, dengan konsol
- Mengelola kebijakan inline cluster melalui konsol (membuat, melihat, memperbarui, dan menghapus kebijakan)
- Buat daftar cluster dan lihat informasi tentang cluster individu
- Lihat tag pada sumber daya apa pun di akun Anda

- Connect ke database sebagai pengguna manapun, termasuk admin
- Lakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan pencadangan dan pemulihan
- Gunakan AWS KMS kunci yang dikelola pelanggan untuk enkripsi klaster
- Luncurkan AWS CloudShell dari Konsol Manajemen AWS
- Melihat metrik apa pun dari CloudWatch akun Anda
- Gunakan AWS Fault Injection Service (AWS FIS) untuk menyuntikkan kegagalan ke cluster Aurora DSQL untuk pengujian toleransi kesalahan
- Buat peran terkait layanan untuk `dsql.amazonaws.com` layanan, yang diperlukan untuk membuat cluster

Anda dapat menemukan `AmazonAuroraDSQLConsoleFullAccess` kebijakan di konsol IAM dan AmazonAuroraDSQLConsoleFullAccessdi Panduan Referensi Kebijakan AWS Terkelola.

Detail izin

Kebijakan ini mencakup izin berikut.

- `dsql`— memberikan izin administratif penuh untuk semua sumber daya di Aurora DSQL melalui. Konsol Manajemen AWS
- `cloudwatch`— memberikan izin untuk mengambil jumlah batch data CloudWatch metrik dan melakukan matematika metrik pada data yang diambil.
- `tag`— memberikan izin untuk mengembalikan kunci tag dan nilai yang saat ini digunakan dalam yang ditentukan Wilayah AWS untuk akun panggilan.
- `backup and restore`— memberikan izin untuk memulai, menghentikan, dan memantau pencadangan dan pemulihan pekerjaan untuk cluster Aurora DSQL.
- `kms`— memberikan izin yang diperlukan untuk memvalidasi akses ke kunci yang dikelola pelanggan yang digunakan untuk enkripsi cluster Aurora DSQL saat membuat, memperbarui, atau menghubungkan ke cluster.
- `cloudshell`— memberikan izin untuk meluncurkan AWS CloudShell untuk berinteraksi dengan Aurora DSQL.
- `ec2`— memberikan izin untuk melihat informasi titik akhir VPC Amazon yang diperlukan untuk koneksi Aurora DSQL.

- `fis`— memberikan izin untuk digunakan untuk menyuntikkan AWS FIS kegagalan ke cluster Aurora DSQL untuk pengujian toleransi kesalahan.
- `access-analyzer:ValidatePolicy`memberikan izin untuk linter di editor kebijakan, yang memberikan umpan balik real-time tentang kesalahan, peringatan, dan masalah keamanan dalam kebijakan saat ini.
- `fis`— memberikan izin untuk menggunakan (AWS Fault Injection Service AWS FIS) untuk menyuntikkan kegagalan ke cluster Aurora DSQL untuk pengujian toleransi kesalahan.

Anda dapat menemukan `AmazonAuroraDSQLConsoleFullAccess` kebijakan di konsol IAM dan Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AuroraDSQLServiceRolePolicy

Anda tidak dapat melampirkan AuroraDSQLServiceRolePolicy ke entitas IAM Anda. Kebijakan ini dilampirkan ke peran terkait layanan yang memungkinkan Aurora DSQL mengakses sumber daya akun.

Anda dapat menemukan `AuroraDSQLServiceRolePolicy` kebijakan di konsol IAM dan AuroraDSQLServiceRolePolicydi Panduan Referensi Kebijakan AWS Terkelola.

Aurora DSQL memperbarui ke AWS kebijakan terkelola

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk Aurora DSQL sejak layanan ini mulai melacak perubahan ini. Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman riwayat Dokumen Aurora DSQL.

Ubah	Deskripsi	Date
AmazonAuroraDSQLFullAccess dan AmazonAuroraDSQLConsoleFullAccess perbarui	Menambahkan dukungan untuk integrasi AWS Fault Injection Service (AWS FIS) dengan Aurora DSQL. Ini memungkinkan Anda untuk menyuntikkan kegagalan	Agustus 19, 2025

Ubah	Deskripsi	Date
	ke dalam kluster DSQL Aurora wilayah tunggal dan Multi-wilayah untuk menguji toleransi kesalahan aplikasi Anda. Anda dapat membuat template eksperimen di AWS FIS konsol untuk menentukan skenario kegagalan dan menargetkan cluster Aurora DSQL tertentu untuk pengujian. Untuk informasi lebih lanjut tentang kebijakan ini, lihat AmazonAuroraDSQLFullAccessdan AmazonAuroraDSQLConsoleFullAccess.	

Ubah	Deskripsi	Date
AmazonAuroraDSQLFullAccess, AmazonAuroraDSQLReadOnlyAccess, dan AmazonAuroraDSQLConsoleFullAccess perbarui	Menambahkan dukungan kebijakan berbasis sumber daya (RBP) dengan izin baru:,, dan. `PutClusterPolicy` `GetClusterPolicy` `DeleteClusterPolicy` Izin ini memungkinkan pengelolaan kebijakan sebaris yang dilampirkan ke cluster Aurora DSQL untuk kontrol akses berbutir halus. Lihat informasi selengkapnya di AmazonAuroraDSQLFullAccess, AmazonAuroraDSQLReadOnlyAccess, dan AmazonAuroraDSQLConsoleFullAccess.	Oktober 15, 2025

Ubah	Deskripsi	Date
AmazonAuroraDSQLFullAccess memperbarui	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga menambahkan kemampuan untuk menggunakan kunci KMS yang dikelola pelanggan untuk enkripsi cluster. Untuk informasi selengkapnya, lihat AmazonAuroraDSQLFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	21 Mei 2025

Ubah	Deskripsi	Date
AmazonAuroraDSQLConsoleFullAccess memperbarui	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL melalui. AWS Console Home Ini termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga mendukung penggunaan kunci KMS yang dikelola pelanggan untuk enkripsi dan peluncuran cluster. AWS CloudShell Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	21 Mei 2025

Ubah	Deskripsi	Date
AmazonAuroraDSQLFullAccess memperbarui	Kebijakan ini menambahkan empat izin baru untuk membuat dan mengelola kluster database di beberapa Wilayah AWS:`PutMultiRegionProperties` ,, `PutWitnessRegion` `AddPeerCluster` , dan. `RemovePeerCluster` Izin ini mencakup kontrol tingkat sumber daya dan kunci kondisi sehingga Anda dapat mengontrol pengguna kluster mana yang dapat Anda modifikasi. Kebijakan ini juga menambahkan `GetVpcEndpointServiceName` izin untuk membantu Anda terhubung ke klaster Aurora DSQL Anda. AWS PrivateLink Untuk informasi selengkapnya, lihat Untuk informasi selengkapnya, lihat AmazonAuroraDSQLFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

Ubah	Deskripsi	Date
AmazonAuroraDSQLReadOnlyAccess memperbarui	Termasuk kemampuan untuk menentukan nama layanan titik akhir VPC yang benar saat menghubungkan ke cluster Aurora DSQL Anda melalui Aurora DSQL menciptakan titik akhir unik per sel, sehingga API ini membantu memastikan Anda dapat mengidentifikasi titik akhir yang benar untuk cluster Anda dan menghindari kesalahan koneksi. AWS PrivateLink Untuk informasi selengkapnya, lihat AmazonAuroraDSQLReadOnlyAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

Ubah	Deskripsi	Date
AmazonAuroraDSQLConsoleFullAccess memperbarui	Menambahkan izin baru ke Aurora DSQL untuk mendukung manajemen klaster Multi-wilayah dan koneksi titik akhir VPC. Izin baru meliputi: `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

Ubah	Deskripsi	Date
AuroraDsqlServiceLinkedRolePolicy memperbarui	Menambahkan kemampuan untuk memublikasikan metrik ke `AWS/Usage` CloudWatch ruang nama `AWS/AuroraDSQL` dan ruang nama ke kebijakan. Hal ini memungkinkan layanan atau peran terkait untuk memancarkan data penggunaan dan kinerja yang lebih komprehensif ke CloudWatch lingkungan Anda. Untuk informasi selengkapnya, lihat AuroraDsqlServiceLinkedRolePolicydan Menggunakan peran terkait layanan di Aurora DSQL.	8 Mei 2025
Halaman dibuat	Mulai melacak kebijakan AWS terkelola yang terkait dengan Amazon Aurora DSQL	Desember 3, 2024

Perlindungan data di Amazon Aurora DSQL

Model tanggung jawab bersama model berlaku untuk perlindungan data di. Seperti yang dijelaskan dalam model 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 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 di Blog Keamanan .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi kredensil dan mengatur pengguna individu dengan AWS IAM Identity Center atau. AWS Identity and Access Management

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. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan jejak untuk menangkap aktivitas, lihat Bekerja dengan jejak di Panduan Pengguna.
- Gunakan solusi 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.

Kami sangat menyarankan agar Anda tidak pernah memasukkan informasi rahasia atau sensitif, seperti alamat email pelanggan Anda, ke dalam tag atau bidang teks bentuk bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan atau lainnya menggunakan konsol, API AWS CLI, atau AWS SDK. 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.

Enkripsi data

Amazon Aurora DSQL menyediakan infrastruktur penyimpanan yang sangat tahan lama yang dirancang untuk penyimpanan data utama dan kritis misi. Data disimpan secara berlebihan di beberapa perangkat di beberapa fasilitas di Wilayah Aurora DSQL.

Enkripsi saat bergerak

Secara default, enkripsi dalam perjalanan dikonfigurasi untuk Anda. Aurora DSQL menggunakan TLS untuk mengenkripsi semua lalu lintas antara klien SQL Anda dan Aurora DSQL.

Enkripsi dan penandatanganan data dalam transit antara AWS CLI, SDK, atau klien API dan titik akhir Aurora DSQL:

- Aurora DSQL menyediakan titik akhir HTTPS untuk mengenkripsi data dalam perjalanan.

- Untuk melindungi integritas permintaan API ke Aurora DSQL, panggilan API harus ditandatangani oleh pemanggil. Panggilan ditandatangani oleh X.509 sertifikat atau kunci akses AWS rahasia pelanggan sesuai dengan Proses Penandatanganan Versi Tanda Tangan 4 (Sigv4). Untuk informasi selengkapnya, lihat Proses Penandatanganan Versi Tanda Tangan 4 di Referensi Umum AWS.
- Gunakan AWS CLI atau salah satu AWS SDK untuk membuat permintaan. AWS Alat-alat ini secara otomatis menandatangani permintaan untuk Anda dengan kunci akses yang Anda tentukan saat Anda mengonfigurasi alat.

Kepatuhan FIPS

Titik akhir jalur data Aurora DSQL (titik akhir cluster yang digunakan untuk koneksi database) menggunakan modul kriptografi tervalidasi FIPS 140-2 secara default. Tidak diperlukan endpoint FIPS terpisah untuk koneksi cluster.

Untuk operasi pesawat kontrol, Aurora DSQL menyediakan titik akhir FIPS khusus di wilayah yang didukung. Untuk informasi selengkapnya tentang titik akhir FIPS bidang kontrol, lihat titik akhir dan kuota Aurora DSQL di. Referensi Umum AWS

Untuk enkripsi saat istirahat, lihatEnkripsi saat istirahat di Aurora DSQL.

Inter-network privasi lalu lintas

Koneksi dilindungi baik antara Aurora DSQL dan aplikasi lokal dan antara Aurora DSQL dan sumber daya lain dalam hal yang sama. AWS Wilayah AWS

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?
- Direct Connect Koneksi. Untuk informasi lebih lanjut, lihat Apa itu Direct Connect?

Anda mendapatkan akses ke Aurora DSQL melalui jaringan dengan menggunakan AWS operasi API yang diterbitkan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Cipher suite dengan perfect forward secrecy (PFS) seperti DHE (Ephemeral) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Diffie-Hellman Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Perlindungan Data di Daerah Saksi

Saat Anda membuat klaster Multi-wilayah, Wilayah saksi membantu mengaktifkan pemulihan kegagalan otomatis dengan berpartisipasi dalam replikasi sinkron transaksi terenkripsi. Jika klaster peered menjadi tidak tersedia, Wilayah saksi tetap tersedia untuk memvalidasi dan memproses penulisan basis data, memastikan tidak ada kehilangan ketersediaan.

Wilayah Saksi melindungi dan mengamankan data Anda melalui fitur desain ini:

- Wilayah saksi hanya menerima dan menyimpan log transaksi terenkripsi. Itu tidak pernah meng-host, menyimpan atau mentransmisikan kunci enkripsi Anda.
- Wilayah saksi hanya berfokus pada penulisan transaksi logging dan fungsi kuorum. Itu tidak dapat membaca data Anda dengan desain.
- Wilayah saksi beroperasi tanpa titik akhir koneksi cluster atau pemroses kueri. Ini mencegah akses basis data pengguna.

Untuk informasi lebih lanjut tentang Wilayah saksi, lihatMengkonfigurasi klaster Multi-wilayah.

Mengkonfigurasi SSL/TLS sertifikat untuk koneksi Aurora DSQL

Aurora DSQL membutuhkan semua koneksi untuk menggunakan enkripsi Transport Layer Security (TLS). Untuk membuat koneksi aman, sistem klien Anda harus mempercayai Amazon Root Certificate Authority (Amazon Root CA 1). Sertifikat ini sudah diinstal sebelumnya pada banyak sistem operasi. Bagian ini memberikan instruksi untuk memverifikasi sertifikat Amazon Root CA 1 yang sudah diinstal sebelumnya pada berbagai sistem operasi, dan memandu Anda melalui proses menginstal sertifikat secara manual jika belum ada.

Kami merekomendasikan menggunakan PostgreSQL versi 17.

> **Important**
>
> Untuk lingkungan produksi, sebaiknya gunakan mode `verify-full` SSL untuk memastikan tingkat keamanan koneksi tertinggi. Mode ini memverifikasi bahwa sertifikat server ditandatangani oleh otoritas sertifikat tepercaya dan bahwa nama host server cocok dengan sertifikat.

Memverifikasi sertifikat pra-instal

Di sebagian besar sistem operasi, Amazon Root CA 1 sudah diinstal sebelumnya. Untuk memvalidasi ini, Anda dapat mengikuti langkah-langkah di bawah ini.

Linux (RedHat/CentOS/Fedora)

Jalankan perintah berikut di terminal Anda:

```
trust list | grep "Amazon Root CA 1"
```

Jika sertifikat diinstal, Anda melihat output berikut:

```
label: Amazon Root CA 1
```

macOS

1. Buka Pencarian Sorotan (Perintah+Spasi)
2. Cari Akses Gantungan Kunci
3. Pilih Akar Sistem di bawah Gantungan Kunci Sistem
4. Cari Amazon Root CA 1 dalam daftar sertifikat

Windows

> **Note**
>
> Karena masalah yang diketahui dengan klien Windows psql, menggunakan sertifikat root sistem (`sslrootcert=system`) dapat mengembalikan kesalahan berikut:. `SSL error: unregistered scheme` Anda dapat mengikuti cara alternatif untuk terhubung ke cluster Anda menggunakan SSL. Menghubungkan dari Windows

Jika Amazon Root CA 1 tidak diinstal di sistem operasi Anda, ikuti langkah-langkah di bawah ini.

Menginstal sertifikat

Jika `Amazon Root CA 1` sertifikat tidak diinstal sebelumnya pada sistem operasi Anda, Anda harus menginstalnya secara manual untuk membuat koneksi aman ke cluster Aurora DSQL Anda.

Instalasi sertifikat Linux

Ikuti langkah-langkah ini untuk menginstal sertifikat Amazon Root CA pada sistem Linux.

1. Unduh Sertifikat Root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Salin sertifikat ke toko kepercayaan:

```
sudo cp ./AmazonRootCA1.pem /etc/pki/ca-trust/source/anchors/
```

3. Perbarui toko kepercayaan CA:

```
sudo update-ca-trust
```

4. Verifikasi instalasi:

```
trust list | grep "Amazon Root CA 1"
```

Instalasi sertifikat macOS

Langkah-langkah pemasangan sertifikat ini bersifat opsional. Ini Instalasi sertifikat Linux juga berfungsi untuk macOS.

1. Unduh Sertifikat Root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Tambahkan sertifikat ke gantungan kunci Sistem:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain
 AmazonRootCA1.pem
```

3. Verifikasi instalasi:

```
security find-certificate -a -c "Amazon Root CA 1" -p /Library/Keychains/
System.keychain
```

Menghubungkan dengan SSL/TLS verifikasi

Sebelum mengonfigurasi SSL/TLS sertifikat untuk koneksi aman ke cluster Aurora DSQL Anda, pastikan Anda memiliki prasyarat berikut.

- PostgreSQL versi 17 diinstal
- AWS CLI dikonfigurasi dengan kredensil yang sesuai
- Informasi titik akhir cluster Aurora DSQL

Menghubungkan dari Linux

1. Hasilkan dan atur token otentikasi:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --hostname your-cluster-endpoint)
```

2. Connect menggunakan sertifikat sistem (jika sudah diinstal sebelumnya):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Atau, sambungkan menggunakan sertifikat yang diunduh:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

> **Note**
>
> Untuk selengkapnya tentang pengaturan PGSSLMODE, lihat sslmode di dokumentasi PostgreSQL 17 Database Connection Control Functions.

Menghubungkan dari macOS

1. Hasilkan dan atur token otentikasi:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --hostname your-cluster-endpoint)
```

2. Connect menggunakan sertifikat sistem (jika sudah diinstal sebelumnya):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Atau, unduh sertifikat root dan simpan sebagai `root.pem` (jika sertifikat tidak diinstal sebelumnya)

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

4. Connect menggunakan psql:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

Menghubungkan dari Windows

Menggunakan Command Prompt

1. Hasilkan token otentikasi:

```
aws dsql generate-db-connect-admin-auth-token ^
--region=your-cluster-region ^
--expires-in=3600 ^
--hostname=your-cluster-endpoint
```

2. Mengatur variabel lingkungan kata sandi:

```
set "PGPASSWORD=token-from-above"
```

3. Atur konfigurasi SSL:

```
set PGSSLROOTCERT=C:\full\path\to\root.pem
set PGSSLMODE=verify-full
```

4. Connect ke database:

```
"C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres ^
--username admin ^
--host your-cluster-endpoint
```

Menggunakan PowerShell

1. Hasilkan dan atur token otentikasi:

```
$env:PGPASSWORD = (aws dsql generate-db-connect-admin-auth-token --region=your-
cluster-region --expires-in=3600 --hostname=your-cluster-endpoint)
```

2. Atur konfigurasi SSL:

```
$env:PGSSLROOTCERT='C:\full\path\to\root.pem'
$env:PGSSLMODE='verify-full'
```

3. Connect ke database:

```
 "C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres `
--username admin `
--host your-cluster-endpoint
```

Sumber daya tambahan

- Dokumentasi PostgreSQL SSL
- Layanan Amazon Trust

Enkripsi data untuk Amazon Aurora DSQL

Amazon Aurora DSQL mengenkripsi semua data pengguna saat istirahat. Untuk keamanan yang ditingkatkan, enkripsi ini menggunakan AWS Key Management Service (AWS KMS). Fungsi ini membantu mengurangi beban operasional dan kompleksitas yang terlibat dalam melindungi data sensitif. Enkripsi saat istirahat membantu Anda:

- Mengurangi beban operasional untuk melindungi data sensitif
- Membangun aplikasi sensitif keamanan yang memenuhi kepatuhan enkripsi yang ketat dan persyaratan peraturan
- Tambahkan lapisan perlindungan data tambahan dengan selalu mengamankan data Anda dalam klaster terenkripsi
- Mematuhi kebijakan organisasi, peraturan industri atau pemerintah, dan persyaratan kepatuhan

Dengan Aurora DSQL, Anda dapat membangun aplikasi yang sensitif terhadap keamanan yang memenuhi kepatuhan enkripsi dan persyaratan peraturan yang ketat. Bagian berikut menjelaskan cara mengkonfigurasi enkripsi untuk database Aurora DSQL baru dan yang sudah ada dan mengelola kunci enkripsi Anda.

Topik

- Jenis kunci KMS untuk Aurora DSQL
- Enkripsi saat istirahat di Aurora DSQL
- Penggunaan AWS KMS dan kunci data dengan Aurora DSQL
- Otorisasi penggunaan Anda AWS KMS key para Aurora DSQL
- Konteks enkripsi Aurora DSQL
- Memantau interaksi Aurora DSQL dengan AWS KMS
- Membuat cluster Aurora DSQL terenkripsi
- Menghapus atau memperbarui kunci untuk cluster Aurora DSQL Anda
- Pertimbangan untuk enkripsi dengan Aurora DSQL

Jenis kunci KMS untuk Aurora DSQL

Aurora DSQL terintegrasi dengan AWS KMS untuk mengelola kunci enkripsi untuk cluster Anda. Untuk mempelajari lebih lanjut tentang jenis dan status kunci, lihat AWS Key Management Service

konsep di Panduan AWS Key Management Service Pengembang. Saat membuat klaster baru, Anda dapat memilih dari jenis kunci KMS berikut untuk mengenkripsi klaster Anda:

Kunci milik AWS

Jenis enkripsi default. Aurora DSQL memiliki kunci tanpa biaya tambahan kepada Anda. Amazon Aurora DSQL secara transparan mendekripsi data klaster saat Anda mengakses kluster terenkripsi. Anda tidak perlu mengubah kode atau aplikasi untuk menggunakan atau mengelola cluster terenkripsi, dan semua kueri Aurora DSQL berfungsi dengan data terenkripsi Anda.

Kunci yang dikelola pelanggan

Anda membuat, memiliki, dan mengelola kunci di Anda Akun AWS. Anda memiliki kontrol penuh atas tombol KMS. AWS KMS dikenakan biaya.

Enkripsi saat istirahat menggunakan Kunci milik AWS tersedia tanpa biaya tambahan. Namun, AWS KMS biaya berlaku untuk kunci yang dikelola pelanggan. Untuk informasi lebih lanjut, lihat halaman AWS KMS Harga.

Anda dapat beralih di antara jenis-jenis kunci ini kapan saja. Untuk informasi selengkapnya tentang jenis kunci, lihat Kunci yang dikelola pelanggan dan Kunci milik AWSdi Panduan AWS Key Management Service Pengembang.

> **Note**
>
> Enkripsi Aurora DSQL saat istirahat tersedia di semua Wilayah di AWS mana Aurora DSQL tersedia.

Enkripsi saat istirahat di Aurora DSQL

Amazon Aurora DSQL menggunakan Advanced Encryption Standard (AES-256) 256-bit untuk mengenkripsi data Anda saat istirahat. Enkripsi ini membantu melindungi data Anda dari akses tidak sah ke penyimpanan yang mendasarinya. AWS KMS mengelola kunci enkripsi untuk cluster Anda. Anda dapat menggunakan defaultKunci milik AWS, atau memilih untuk menggunakan sendiri AWS KMS Kunci yang dikelola pelanggan. Untuk mempelajari lebih lanjut tentang menentukan dan mengelola kunci untuk klaster Aurora DSQL Anda, lihat dan. Membuat cluster Aurora DSQL terenkripsi Menghapus atau memperbarui kunci untuk cluster Aurora DSQL Anda

Topik

- Kunci milik AWS
- Kunci yang dikelola pelanggan

Kunci milik AWS

Aurora DSQL mengenkripsi semua cluster secara default dengan. Kunci milik AWS Kunci ini gratis untuk digunakan dan diputar setiap tahun untuk melindungi sumber daya akun Anda. Anda tidak perlu melihat, mengelola, menggunakan, atau mengaudit kunci ini, sehingga tidak ada tindakan yang diperlukan untuk perlindungan data. Untuk informasi selengkapnya Kunci milik AWS, lihat Kunci milik AWSdi Panduan AWS Key Management Service Pengembang.

Kunci yang dikelola pelanggan

Anda membuat, memiliki, dan mengelola kunci yang dikelola pelanggan di Anda Akun AWS. Anda memiliki kendali penuh atas kunci KMS ini, termasuk kebijakan, materi enkripsi, tag, dan aliasnya. Untuk informasi selengkapnya tentang mengelola izin, lihat Kunci yang dikelola pelanggan di Panduan AWS Key Management Service Pengembang.

Saat Anda menentukan kunci terkelola pelanggan untuk enkripsi tingkat kluster, Aurora DSQL mengenkripsi cluster dan semua data regionalnya dengan kunci itu. Untuk mencegah kehilangan data dan mempertahankan akses cluster, Aurora DSQL memerlukan akses ke kunci enkripsi Anda. Jika Anda menonaktifkan kunci terkelola pelanggan, menjadwalkan kunci Anda untuk dihapus, atau memiliki kebijakan yang membatasi akses layanan Anda, status enkripsi untuk klaster Anda berubah menjadi. `KMS_KEY_INACCESSIBLE` Ketika Aurora DSQL tidak dapat mengakses kunci, pengguna tidak dapat terhubung ke cluster, status enkripsi untuk klaster berubah`KMS_KEY_INACCESSIBLE`, dan layanan kehilangan akses ke data cluster.

Untuk klaster Multi-region, pelanggan dapat mengonfigurasi kunci AWS KMS enkripsi setiap wilayah secara terpisah, dan setiap cluster regional menggunakan kunci enkripsi tingkat klaster sendiri. Jika Aurora DSQL tidak dapat mengakses kunci enkripsi untuk peer di klaster Multi-wilayah, status untuk rekan tersebut menjadi `KMS_KEY_INACCESSIBLE` dan menjadi tidak tersedia untuk operasi baca dan tulis. Rekan-rekan lainnya melanjutkan operasi normal.

> **Note**
>
> Jika Aurora DSQL tidak dapat mengakses kunci terkelola pelanggan Anda, status enkripsi klaster Anda akan berubah menjadi. `KMS_KEY_INACCESSIBLE` Setelah Anda

mengembalikan akses kunci, layanan akan secara otomatis mendeteksi pemulihan dalam waktu 15 menit. Untuk informasi selengkapnya, lihat Cluster idling.
Untuk klaster Multi-wilayah, jika akses kunci hilang untuk waktu yang lama, waktu pemulihan klaster tergantung pada berapa banyak data yang ditulis saat kunci tidak dapat diakses.

Penggunaan AWS KMS dan kunci data dengan Aurora DSQL

Fitur enkripsi Aurora DSQL saat istirahat menggunakan AWS KMS key dan hierarki kunci data untuk melindungi data cluster Anda.

Kami menyarankan Anda merencanakan strategi enkripsi Anda sebelum menerapkan cluster Anda di Aurora DSQL. Jika Anda menyimpan data sensitif atau rahasia di Aurora DSQL, pertimbangkan untuk menyertakan enkripsi sisi klien dalam paket Anda. Dengan cara ini, Anda dapat mengenkripsi data sedekat mungkin dengan asalnya, dan memastikan perlindungannya sepanjang siklus hidupnya.

Topik

- Penggunaan AWS KMS key s dengan Aurora DSQL
- Menggunakan tombol cluster dengan Aurora DSQL
- Caching kunci cluster

Penggunaan AWS KMS key s dengan Aurora DSQL

Enkripsi saat istirahat melindungi cluster Aurora DSQL Anda di bawah file. AWS KMS key Secara default, Aurora DSQL menggunakan Kunci milik AWS, kunci enkripsi multi-tenant yang dibuat dan dikelola di akun layanan Aurora DSQL. Tetapi Anda dapat mengenkripsi cluster Aurora DSQL Anda di bawah kunci yang dikelola pelanggan di Anda. Akun AWS Anda dapat memilih kunci KMS yang berbeda untuk setiap cluster, bahkan jika itu berpartisipasi dalam pengaturan Multi-wilayah.

Anda memilih kunci KMS untuk klaster saat Anda membuat atau memperbarui cluster. Anda dapat mengubah kunci KMS untuk cluster kapan saja, baik di konsol Aurora DSQL atau dengan menggunakan operasi. `UpdateCluster` Proses peralihan kunci tidak memerlukan downtime atau menurunkan layanan.

> **Important**
>
> Aurora DSQL hanya mendukung tombol KMS simetris. Anda tidak dapat menggunakan kunci KMS asimetris untuk mengenkripsi cluster Aurora DSQL Anda.

Kunci yang dikelola pelanggan memberikan manfaat berikut.

- Anda membuat dan mengelola kunci KMS, termasuk menyetel kebijakan utama dan kebijakan IAM untuk mengontrol akses ke kunci KMS. Anda dapat mengaktifkan dan menonaktifkan kunci KMS, mengaktifkan dan menonaktifkan rotasi kunci otomatis, dan menghapus kunci KMS ketika sudah tidak digunakan.
- Anda dapat menggunakan kunci yang dikelola pelanggan dengan material kunci yang diimpor atau kunci yang dikelola pelanggan di penyimpanan kunci kustom yang Anda miliki dan kelola.
- Anda dapat mengaudit enkripsi dan dekripsi cluster Aurora DSQL Anda dengan memeriksa panggilan Aurora DSQL API ke dalam log. AWS KMS AWS CloudTrail

Namun, Kunci milik AWS ini gratis dan penggunaannya tidak dihitung terhadap AWS KMS sumber daya atau kuota permintaan. Kunci yang dikelola pelanggan dikenakan biaya untuk setiap panggilan API dan AWS KMS kuota berlaku untuk kunci ini.

Menggunakan tombol cluster dengan Aurora DSQL

Aurora DSQL menggunakan AWS KMS key for the cluster untuk menghasilkan dan mengenkripsi kunci data unik untuk cluster, yang dikenal sebagai kunci cluster.

Kunci cluster digunakan sebagai kunci enkripsi kunci. Aurora DSQL menggunakan kunci cluster ini untuk melindungi kunci enkripsi data yang digunakan untuk mengenkripsi data cluster. Aurora DSQL menghasilkan kunci enkripsi data unik untuk setiap struktur yang mendasarinya dalam sebuah cluster, tetapi beberapa item cluster mungkin dilindungi oleh kunci enkripsi data yang sama.

Untuk mendekripsi kunci cluster, Aurora DSQL mengirimkan permintaan AWS KMS ketika Anda pertama kali mengakses cluster terenkripsi. Agar klaster tetap tersedia, Aurora DSQL secara berkala memverifikasi akses dekripsi ke kunci KMS, bahkan ketika Anda tidak aktif mengakses cluster.

Aurora DSQL menyimpan dan menggunakan kunci cluster dan kunci enkripsi data di luar. AWS KMS Layanan ini melindungi semua kunci dengan enkripsi Advanced Encryption Standard (AES) dan

kunci enkripsi 256-bit. Kemudian, ia menyimpan kunci terenkripsi dengan data terenkripsi sehingga tersedia untuk mendekripsi data cluster sesuai permintaan.

Jika Anda mengubah kunci KMS untuk klaster Anda, Aurora DSQL mengenkripsi ulang kunci cluster yang ada dengan kunci KMS baru.

Caching kunci cluster

Untuk menghindari panggilan AWS KMS untuk setiap operasi Aurora DSQL, Aurora DSQL menyimpan kunci cluster plaintext untuk setiap pemanggil dalam memori. Jika Aurora DSQL mendapat permintaan untuk kunci cluster cache setelah 15 menit tidak aktif, ia mengirimkan permintaan baru untuk AWS KMS mendekripsi kunci cluster. Panggilan ini akan menangkap setiap perubahan yang dibuat pada kebijakan akses AWS KMS key dalam AWS KMS atau AWS Identity and Access Management (IAM) setelah permintaan terakhir untuk mendekripsi kunci klaster.

Otorisasi penggunaan Anda AWS KMS key para Aurora DSQL

Jika Anda menggunakan kunci yang dikelola pelanggan di akun Anda untuk melindungi klaster Aurora DSQL Anda, kebijakan pada kunci tersebut harus memberikan izin kepada Aurora DSQL untuk menggunakannya atas nama Anda.

Anda memiliki kendali penuh atas kebijakan pada kunci yang dikelola pelanggan. Aurora DSQL tidak memerlukan otorisasi tambahan untuk menggunakan default untuk Kunci milik AWS melindungi cluster Aurora DSQL di Anda. Akun AWS

Kebijakan kunci untuk kunci yang dikelola pelanggan

Ketika Anda memilih kunci yang dikelola pelanggan untuk melindungi cluster Aurora DSQL, Aurora DSQL memerlukan izin untuk menggunakan AWS KMS key atas nama kepala sekolah yang membuat pilihan. Prinsipal itu, pengguna atau peran, harus memiliki izin pada Aurora DSQL AWS KMS key yang dibutuhkan. Anda dapat memberikan izin ini dalam kebijakan utama, atau kebijakan IAM.

Minimal, Aurora DSQL memerlukan izin berikut pada kunci yang dikelola pelanggan:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`(untuk kms: ReEncryptFrom dan kms:) ReEncryptTo
- `kms:GenerateDataKey`

- `kms:DescribeKey`

Sebagai contoh, kebijakan kunci berikut hanya menyediakan izin yang diperlukan. Kebijakan ini memiliki efek sebagai berikut:

- Memungkinkan Aurora DSQL untuk menggunakan AWS KMS key dalam operasi kriptografi, tetapi hanya ketika bertindak atas nama kepala sekolah di akun yang memiliki izin untuk menggunakan Aurora DSQL. Jika prinsipal yang ditentukan dalam pernyataan kebijakan tidak memiliki izin untuk menggunakan Aurora DSQL, panggilan gagal, bahkan ketika itu berasal dari layanan Aurora DSQL.
- Kunci `kms:ViaService` kondisi mengizinkan izin hanya jika permintaan berasal dari Aurora DSQL atas nama prinsipal yang tercantum dalam pernyataan kebijakan. Pengguna utama ini tidak dapat memanggil operasi ini secara langsung.

Sebelum menggunakan kebijakan kunci contoh, ganti prinsip contoh dengan prinsip aktual dari Anda. Akun AWS

```
{
  "Sid": "Enable dsql IAM User Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:Encrypt",
    "kms:ReEncryptFrom",
    "kms:ReEncryptTo"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:EncryptionContext:aws:dsql:ClusterId": "w4abucpbwuxx",
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
},
{
  "Sid": "Enable dsql IAM User Describe Permissions",
```

```
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
}
```

Konteks enkripsi Aurora DSQL

Konteks enkripsi adalah seperangkat pasangan kunci-nilai yang berisi data non-rahasia yang arbitrer. Ketika Anda menyertakan konteks enkripsi dalam permintaan untuk mengenkripsi data, secara AWS KMS kriptografis mengikat konteks enkripsi ke data terenkripsi. Untuk mendekripsi data, Anda harus meneruskan konteks enkripsi yang sama.

Aurora DSQL menggunakan konteks enkripsi yang sama di semua operasi kriptografi. AWS KMS Jika Anda menggunakan kunci yang dikelola pelanggan untuk melindungi klaster Aurora DSQL Anda, Anda dapat menggunakan konteks enkripsi untuk mengidentifikasi penggunaan catatan dan log audit AWS KMS key dalam. Itu juga muncul dalam teks biasa di log seperti yang ada di. AWS CloudTrail

Konteks enkripsi juga dapat digunakan sebagai syarat untuk otorisasi dalam kebijakan.

Dalam permintaannya AWS KMS, Aurora DSQL menggunakan konteks enkripsi dengan pasangan kunci-nilai:

```
"encryptionContext": {
  "aws:dsql:ClusterId": "w4abucpbwuxx"
},
```

Pasangan kunci-nilai mengidentifikasi cluster yang dienkripsi Aurora DSQL. Kuncinya adalah `aws:dsql:ClusterId`. Nilainya adalah pengidentifikasi cluster.

Memantau interaksi Aurora DSQL dengan AWS KMS

Jika Anda menggunakan kunci yang dikelola pelanggan untuk melindungi klaster Aurora DSQL Anda, Anda dapat menggunakan AWS CloudTrail log untuk melacak permintaan yang dikirim Aurora DSQL atas nama Anda. AWS KMS

Perluas bagian berikut untuk mempelajari bagaimana Aurora DSQL menggunakan operasi dan. AWS KMS `GenerateDataKey` `Decrypt`

`GenerateDataKey`

Saat Anda mengaktifkan enkripsi saat istirahat di cluster, Aurora DSQL membuat kunci cluster yang unik. Ini mengirimkan `GenerateDataKey` permintaan untuk AWS KMS yang menentukan AWS KMS key untuk cluster.

Peristiwa yang mencatat operasi `GenerateDataKey` serupa dengan peristiwa contoh berikut. Pengguna adalah akun layanan Aurora DSQL. Parameter termasuk Amazon Resource Name (ARN) dari AWS KMS key, penentu kunci yang memerlukan kunci 256-bit, dan konteks enkripsi yang mengidentifikasi cluster.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dsql.amazonaws.com"
    },
    "eventTime": "2025-05-16T18:41:24Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "dsql.amazonaws.com",
    "userAgent": "dsql.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dsql:ClusterId": "w4abucpbwuxx"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
    },
    "responseElements": null,
    "requestID": "2da2dc32-d3f4-4d6c-8a41-aff27cd9a733",
```

```
    "eventID": "426df0a6-ba56-3244-9337-438411f826f4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "f88e0dd8-6057-4ce0-b77d-800448426d4e",
    "vpcEndpointId": "AWS Internal",
    "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
    "eventCategory": "Management"
}
```

Dekripsi

Saat Anda mengakses cluster Aurora DSQL terenkripsi, Aurora DSQL perlu mendekripsi kunci cluster sehingga dapat mendekripsi kunci di bawahnya dalam hierarki. Kemudian mendekripsi data di cluster. Untuk mendekripsi kunci cluster, Aurora DSQL mengirimkan `Decrypt` permintaan yang menentukan untuk AWS KMS cluster. AWS KMS key

Peristiwa yang mencatat operasi `Decrypt` serupa dengan peristiwa contoh berikut. Pengguna adalah kepala sekolah Anda Akun AWS yang mengakses cluster. Parameter termasuk kunci cluster terenkripsi (sebagai gumpalan ciphertext) dan konteks enkripsi yang mengidentifikasi cluster. AWS KMS berasal dari ID dari AWS KMS key ciphertext.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "dsql.amazonaws.com"
  },
  "eventTime": "2018-02-14T16:42:39Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-east-1",
```

```
  "sourceIPAddress": "dsql.amazonaws.com",
  "userAgent": "dsql.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "encryptionContext": {
      "aws:dsql:ClusterId": "w4abucpbwuxx"
    },
    "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
  },
  "responseElements": null,
  "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
  "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "AWS Internal",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "sharedEventID": "d99f2dc5-b576-45b6-aa1d-3a3822edbeeb",
  "vpcEndpointId": "AWS Internal",
  "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
  "eventCategory": "Management"
}
```


Membuat cluster Aurora DSQL terenkripsi

Semua cluster Aurora DSQL dienkripsi saat istirahat. Secara default, cluster menggunakan tanpa Kunci milik AWS biaya, atau Anda dapat menentukan AWS KMS kunci khusus. Ikuti langkah-langkah ini untuk membuat cluster terenkripsi Anda baik dari Konsol Manajemen AWS atau. AWS CLI

Console

Untuk membuat cluster terenkripsi di Konsol Manajemen AWS

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Di panel navigasi di sisi kiri konsol, pilih Klaster.
3. Pilih Create Cluster di kanan atas dan pilih Single-Region.
4. Dalam pengaturan enkripsi Cluster, pilih salah satu opsi berikut.
   - Terima pengaturan default untuk mengenkripsi dengan tanpa Kunci milik AWS biaya tambahan.
   - Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk menentukan kunci KMS kustom. Kemudian, cari atau masukkan ID atau alias kunci KMS Anda. Atau, pilih Buat AWS KMS kunci untuk membuat kunci baru di AWS KMS Konsol.
5. Pilih Buat klaster.

Untuk mengonfirmasi jenis enkripsi klaster Anda, navigasikan ke halaman Clusters dan pilih ID klaster untuk melihat detail klaster. Tinjau tab pengaturan Cluster Pengaturan kunci KMS Cluster menunjukkan kunci default Aurora DSQL untuk cluster yang AWS menggunakan kunci yang dimiliki atau ID kunci untuk jenis enkripsi lainnya.

> **Note**
>
> Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Anda menetapkan kebijakan kunci KMS dengan tepat. Untuk contoh dan informasi lebih lanjut, lihatthe section called “Kebijakan kunci untuk kunci yang dikelola pelanggan”.

CLI

Untuk membuat cluster yang dienkripsi dengan default Kunci milik AWS

- Gunakan perintah berikut untuk membuat cluster Aurora DSQL.

```
aws dsql create-cluster
```

Seperti yang ditunjukkan dalam detail enkripsi berikut, status enkripsi untuk cluster diaktifkan secara default, dan jenis enkripsi default adalah kunci yang dimiliki AWS. Cluster sekarang dienkripsi dengan kunci AWS default yang dimiliki di akun layanan Aurora DSQL.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Untuk membuat klaster yang dienkripsi dengan kunci terkelola pelanggan Anda

- Gunakan perintah berikut untuk membuat cluster Aurora DSQL, mengganti ID kunci dalam teks merah dengan ID kunci yang dikelola pelanggan Anda.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Seperti yang ditunjukkan dalam detail enkripsi berikut, status enkripsi untuk cluster diaktifkan secara default, dan jenis enkripsi adalah kunci KMS yang dikelola pelanggan. Cluster sekarang dienkripsi dengan kunci Anda.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Menghapus atau memperbarui kunci untuk cluster Aurora DSQL Anda

Anda dapat menggunakan atau AWS CLI untuk memperbarui Konsol Manajemen AWS atau menghapus kunci enkripsi pada cluster yang ada di Amazon Aurora DSQL. Jika Anda menghapus kunci tanpa menggantinya, Aurora DSQL menggunakan default. Kunci milik AWS Ikuti langkah-langkah ini untuk memperbarui kunci enkripsi cluster yang ada dari konsol Aurora DSQL atau. AWS CLI

Console

Untuk memperbarui atau menghapus kunci enkripsi di Konsol Manajemen AWS

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Di panel navigasi di sisi kiri konsol, pilih Klaster.
3. Dari tampilan daftar, temukan dan pilih baris cluster yang ingin Anda perbarui.
4. Pilih menu Tindakan dan kemudian pilih Ubah.
5. Dalam pengaturan enkripsi Cluster, pilih salah satu opsi berikut untuk mengubah pengaturan enkripsi Anda.
   - Jika Anda ingin beralih dari kunci khusus ke kunci Kunci milik AWS, hapus pilihan Sesuaikan pengaturan enkripsi (lanjutan). Pengaturan default akan menerapkan dan mengenkripsi cluster Anda dengan tanpa Kunci milik AWS biaya.
   - Jika Anda ingin beralih dari satu kunci KMS kustom ke yang lain atau dari tombol Kunci milik AWS ke KMS, pilih opsi Sesuaikan pengaturan enkripsi (lanjutan) jika belum dipilih. Kemudian, cari dan pilih ID atau alias kunci yang ingin Anda gunakan. Atau, pilih Buat AWS KMS kunci untuk membuat kunci baru di AWS KMS Konsol.
6. Pilih Simpan.

CLI

Contoh berikut menunjukkan cara menggunakan AWS CLI untuk memperbarui cluster terenkripsi.

Untuk memperbarui cluster terenkripsi dengan default Kunci milik AWS

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

Deskripsi cluster diatur ke `ENABLED` dan `EncryptionType` adalah`AWS_OWNED_KMS_KEY`. `EncryptionStatus`

```
"encryptionDetails": {
```

```
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Cluster ini sekarang dienkripsi menggunakan default Kunci milik AWS di akun layanan Aurora DSQL.

Untuk memperbarui cluster terenkripsi dengan kunci terkelola pelanggan untuk Aurora DSQL

Perbarui cluster terenkripsi, seperti pada contoh berikut:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-ab1234a1b234
```

Transisi deskripsi cluster ke `UPDATING` dan `EncryptionType` is`CUSTOMER_MANAGED_KMS_KEY`. `EncryptionStatus` Setelah Aurora DSQL selesai menyebarkan kunci baru melalui platform, status enkripsi akan dialihkan ke `ENABLED`

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Anda menetapkan kebijakan kunci KMS dengan tepat. Untuk contoh dan informasi lebih lanjut, lihatthe section called “Kebijakan kunci untuk kunci yang dikelola pelanggan”.

Pertimbangan untuk enkripsi dengan Aurora DSQL

- Aurora DSQL mengenkripsi semua data cluster saat istirahat. Anda tidak dapat menonaktifkan enkripsi ini atau mengenkripsi hanya beberapa item dalam sebuah cluster.
- AWS Backup mengenkripsi cadangan Anda dan cluster apa pun yang dipulihkan dari cadangan ini. Anda dapat mengenkripsi data cadangan Anda dengan AWS Backup menggunakan kunci yang AWS dimiliki atau kunci yang dikelola pelanggan.
- Status perlindungan data berikut diaktifkan untuk Aurora DSQL:
  - Data saat istirahat - Aurora DSQL mengenkripsi semua data statis pada media penyimpanan persisten
  - Data dalam perjalanan - Aurora DSQL mengenkripsi semua komunikasi menggunakan Transport Layer Security (TLS) secara default
- Saat Anda beralih ke kunci yang berbeda, kami sarankan agar Anda tetap mengaktifkan kunci asli hingga transisi selesai. AWS membutuhkan kunci asli untuk mendekripsi data sebelum mengenkripsi data Anda dengan kunci baru. Prosesnya selesai ketika cluster `encryptionStatus` berada `ENABLED` dan Anda melihat kunci `kmsKeyArn` yang dikelola pelanggan baru.
- Saat Anda menonaktifkan Kunci Terkelola Pelanggan atau mencabut akses Aurora DSQL untuk menggunakan kunci Anda, klaster Anda akan masuk ke status. `IDLE`
- API DSQL Amazon Aurora Konsol Manajemen AWS dan Amazon menggunakan istilah yang berbeda untuk jenis enkripsi:
  - AWS Konsol — Di konsol, Anda akan melihat `KMS` saat menggunakan kunci yang dikelola Pelanggan dan `DEFAULT` saat menggunakan kunci Kunci milik AWS.
  - API — Amazon Aurora DSQL API digunakan `CUSTOMER_MANAGED_KMS_KEY` untuk kunci yang dikelola pelanggan, dan untuk. `AWS_OWNED_KMS_KEY` Kunci milik AWS
- Jika Anda tidak menentukan kunci enkripsi selama pembuatan klaster, Aurora DSQL secara otomatis mengenkripsi data Anda menggunakan file. Kunci milik AWS
- Anda dapat beralih antara kunci yang dikelola Pelanggan Kunci milik AWS dan kapan saja. Buat perubahan ini menggunakan Konsol Manajemen AWS, AWS CLI, atau Amazon Aurora DSQL API.

Manajemen identitas dan akses untuk Aurora DSQL

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang

dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya Aurora DSQL. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Bagaimana Amazon Aurora DSQL bekerja dengan IAM
- Identity-based contoh kebijakan untuk Amazon Aurora DSQL
- Memecahkan masalah identitas dan akses Amazon Aurora DSQL

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda berdasarkan peran Anda:

- Pengguna layanan - minta izin dari administrator Anda jika Anda tidak dapat mengakses fitur (lihat Memecahkan masalah identitas dan akses Amazon Aurora DSQL)
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat Bagaimana Amazon Aurora DSQL bekerja dengan IAM)
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat Identity-based contoh kebijakan untuk Amazon Aurora DSQL)

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensyal identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredensil dari sumber identitas seperti AWS IAM Identity Center (Pusat Identitas IAM), autentikasi masuk tunggal, atau kredensional. Google/Facebook Untuk informasi selengkapnya tentang cara masuk, lihat Cara masuk ke Akun AWS Anda dalam Panduan Pengguna AWS Sign-In .

Untuk akses terprogram, AWS sediakan SDK dan CLI untuk menandatangani permintaan secara kriptografis. Untuk informasi selengkapnya, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang disebut pengguna Akun AWS root yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Untuk tugas yang memerlukan kredensial pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori perusahaan Anda, penyedia identitas web, atau Directory Service yang mengakses Layanan AWS menggunakan kredensil dari sumber identitas. Identitas terfederasi mengambil peran yang memberikan kredensial sementara.

Untuk manajemen akses terpusat, kami menyarankan AWS IAM Identity Center. Untuk informasi selengkapnya, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat Mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredensi sementara di Panduan Pengguna IAM.

Grup IAM menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan beralih dari pengguna ke peran IAM (konsol) atau dengan

memanggil operasi AWS CLI atau AWS API. Untuk informasi selengkapnya, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna terfederasi, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon EC2. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukan operasinya.

Identity-based kebijakan

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan dalam Panduan Pengguna IAM.

Identity-based kebijakan dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada beberapa identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat Pilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Resource-based kebijakan

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contohnya termasuk kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3.

Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Anda harus menentukan principal dalam kebijakan berbasis sumber daya.

Resource-based kebijakan adalah kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat menetapkan izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCP) – Menentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam Panduan Pengguna AWS Organizations .
- Kebijakan kontrol sumber daya (RCP) – Menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Kebijakan kontrol sumber daya (RCP) dalam Panduan Pengguna AWS Organizations .
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

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 Pengguna IAM.

Bagaimana Amazon Aurora DSQL bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Aurora DSQL, pelajari fitur IAM apa yang tersedia untuk digunakan dengan Aurora DSQL.

Fitur IAM yang dapat Anda gunakan dengan Amazon Aurora DSQL

Fitur IAM	Dukungan Aurora DSQL
Identity-based kebijakan	Ya
Resource-based kebijakan	Ya
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Ya
ACL	Tidak
ABAC (tanda dalam kebijakan)	Ya
Kredensial sementara	Ya
Izin principal	Ya
Peran layanan	Ya
Service-linked peran	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang bagaimana Aurora DSQL dan layanan AWS lainnya bekerja dengan sebagian besar fitur IAM, AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Identity-based kebijakan untuk Aurora DSQL

Mendukung kebijakan berbasis identitas: Ya

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke identitas, seperti pengguna IAM, grup 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 Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Identity-based contoh kebijakan untuk Aurora DSQL

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Identity-based contoh kebijakan untuk Amazon Aurora DSQL

Resource-based kebijakan dalam Aurora DSQL

Mendukung kebijakan berbasis sumber daya: Ya

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM 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 principal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan principal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau layanan AWS. Resource-based kebijakan adalah kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan yang dikelola AWS dari IAM dalam kebijakan berbasis sumber daya.

Untuk mempelajari cara membuat dan mengelola kebijakan berbasis sumber daya untuk klaster Aurora DSQL, lihat kebijakan untuk Aurora DSQL. Resource-based

Tindakan kebijakan untuk Aurora DSQL

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan Aurora DSQL, lihat Tindakan yang Ditentukan oleh Amazon Aurora DSQL di Referensi Otorisasi Layanan.

Tindakan kebijakan di Aurora DSQL menggunakan awalan berikut sebelum tindakan:

```
dsql
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "dsql:action1",
      "dsql:action2"
]
```

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Identity-based contoh kebijakan untuk Amazon Aurora DSQL

Sumber daya kebijakan untuk Aurora DSQL

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Praktik terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis sumber daya Aurora DSQL dan ARNnya, lihat Sumber Daya yang Ditentukan oleh Amazon Aurora DSQL di Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan yang Ditentukan oleh Amazon Aurora DSQL.

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Identity-based contoh kebijakan untuk Amazon Aurora DSQL

Kunci kondisi kebijakan untuk Aurora DSQL

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Condition` menentukan ketika pernyataan dieksekusi berdasarkan kriteria yang ditetapkan. 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. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Untuk melihat daftar kunci kondisi Aurora DSQL, lihat Kunci kondisi untuk Amazon Aurora DSQL di Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang ditentukan oleh Amazon Aurora DSQL.

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Identity-based contoh kebijakan untuk Amazon Aurora DSQL

ACL di Aurora DSQL

Mendukung ACL: Tidak

Daftar kontrol akses (ACL) mengendalikan principal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun kebijakan tersebut tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan Aurora DSQL

Mendukung ABAC (tanda dalam kebijakan): Ya

Attribute-based Access Control (ABAC) adalah strategi otorisasi yang mendefinisikan izin berdasarkan atribut yang disebut tag. Anda dapat melampirkan tag ke entitas dan AWS sumber daya

IAM, lalu merancang kebijakan ABAC untuk mengizinkan operasi saat tag prinsipal cocok dengan tag pada sumber daya.

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 selengkapnya tentang ABAC, lihat Tentukan izin dengan otorisasi ABAC dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) dalam Panduan Pengguna IAM.

Menggunakan kredensil sementara dengan Aurora DSQL

Mendukung kredensial sementara: Ya

Kredensi sementara menyediakan akses jangka pendek ke AWS sumber daya dan secara otomatis dibuat saat Anda menggunakan federasi atau beralih peran. AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM dan Layanan AWS yang berfungsi dengan IAM dalam Panduan Pengguna IAM.

Cross-service izin utama untuk Aurora DSQL

Mendukung sesi akses terusan (FAS): Ya

Sesi akses teruskan (FAS) menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses terusan.

Peran layanan untuk Aurora DSQL

Mendukung peran layanan: Ya

Peran layanan adalah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat merusak fungsionalitas Aurora DSQL. Edit peran layanan hanya ketika Aurora DSQL memberikan panduan untuk melakukannya.

Service-linked peran untuk Aurora DSQL

Mendukung peran terkait layanan: Ya

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat mengambil peran untuk melakukan tindakan atas nama Anda. Service-linked peran muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang membuat atau mengelola peran terkait layanan untuk Aurora DSQL, lihat. Menggunakan peran terkait layanan di Aurora DSQL

Identity-based contoh kebijakan untuk Amazon Aurora DSQL

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya Aurora DSQL. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM dengan menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM (konsol) di Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh Aurora DSQL, termasuk format ARN untuk setiap jenis sumber daya, lihat Tindakan, Sumber Daya, dan Kunci Kondisi untuk Amazon Aurora DSQL di Referensi Otorisasi Layanan.

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol Aurora DSQL
- Mengizinkan pengguna melihat izin mereka sendiri
- Izinkan manajemen cluster dan koneksi database
- Akses sumber daya Aurora DSQL berdasarkan tag

Praktik terbaik kebijakan

Identity-based kebijakan menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Aurora DSQL di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. 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 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 DSQL

Untuk mengakses konsol Amazon Aurora DSQL, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya Aurora DSQL 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 gantinya, izinkan akses hanya ke tindakan yang sesuai dengan operasi API yang coba mereka lakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol Aurora DSQL, lampirkan juga Aurora `AmazonAuroraDSQLConsoleFullAccess` DSQL atau kebijakan terkelola ke entitas. `AmazonAuroraDSQLReadOnlyAccess` AWS Untuk informasi selengkapnya, lihat Menambah izin untuk pengguna dalam Panduan Pengguna IAM.

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": "*"
        }
    ]
}
```

Izinkan manajemen cluster dan koneksi database

Kebijakan berikut memberikan izin pengguna IAM untuk mengelola dan menyambung ke cluster Aurora DSQL tertentu. Kebijakan ini mencakup pengelolaan klaster dan tindakan koneksi ke satu klaster Amazon Resource Name (ARN), sekaligus `dsql:ListClusters` mengizinkan semua resource karena tindakan ini tidak mendukung izin tingkat sumber daya.

Contoh ini digunakan `dsql:DbConnectAdmin` untuk terhubung dengan `admin` peran. Untuk terhubung dengan peran database kustom sebagai gantinya, ganti `dsql:DbConnectAdmin` dengan`dsql:DbConnect`. Untuk informasi selengkapnya, lihat Otentikasi dan otorisasi untuk Aurora DSQL.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowClusterManagement",
            "Effect": "Allow",
            "Action": [
                "dsql:GetCluster",
                "dsql:UpdateCluster",
                "dsql:DeleteCluster",
                "dsql:DbConnectAdmin",
                "dsql:TagResource",
                "dsql:ListTagsForResource",
                "dsql:UntagResource"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/my-cluster-id"
        },
        {
            "Sid": "AllowListClusters",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
        }
    ]
}
```

Akses sumber daya Aurora DSQL berdasarkan tag

Anda dapat menggunakan kondisi dalam kebijakan berbasis identitas untuk mengontrol akses ke sumber daya Aurora DSQL berdasarkan tag. Contoh berikut menunjukkan cara membuat kebijakan yang memungkinkan melihat klaster. Namun, kebijakan hanya memberikan izin jika tag klaster `0wner` memiliki nilai nama pengguna tersebut. Kebijakan ini juga memberi izin yang diperlukan untuk menyelesaikan tindakan ini pada konso tersebutl.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListClustersInConsole",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
```

```
        },
        {
            "Sid": "ViewClusterIfOwner",
            "Effect": "Allow",
            "Action": "dsql:GetCluster",
            "Resource": "arn:aws:dsql:*:*:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

Anda dapat melampirkan kebijakan ini ke pengguna IAM di akun Anda. Jika pengguna bernama `richard-roe` mencoba untuk melihat cluster Aurora DSQL, cluster harus diberi tag atau. `0wner=richard-roe` `owner=richard-roe` Jika tidak, IAM menolak akses. Kunci tanda syarat `0wner` sama dengan kedua `0wner` dan `owner` karena nama kunci syarat tidak terpengaruh huruf besar/kecil. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM JSON: Syarat dalam Panduan Pengguna IAM.

Kebijakan berikut memungkinkan pengguna untuk membuat cluster hanya jika mereka menandai klaster dengan nama pengguna mereka sendiri sebagai. `0wner` Ini juga memungkinkan penandaan hanya pada cluster yang sudah dimiliki pengguna.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateTaggedCluster",
            "Effect": "Allow",
            "Action": "dsql:CreateCluster",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
```

```
            }
        },
        {
            "Sid": "AllowTagOwnedClusters",
            "Effect": "Allow",
            "Action": "dsql:TagResource",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

Memecahkan masalah identitas dan akses Amazon Aurora DSQL

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Aurora DSQL dan IAM.

Topik

- Saya tidak berwenang untuk melakukan tindakan di Aurora DSQL
- Saya tidak berwenang untuk melakukan iam: PassRole
- Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya Aurora DSQL saya

Saya tidak berwenang untuk melakukan tindakan di Aurora DSQL

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang *`my-dsql-cluster`* sumber daya tetapi tidak memiliki *`GetCluster`* izin.

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

Dalam hal ini, kebijakan untuk pengguna `mateojackson` harus diperbarui untuk mengizinkan akses ke sumber daya *`my-dsql-cluster`* dengan menggunakan tindakan *`GetCluster`*.

Jika Anda membutuhkan bantuan, hubungi administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan bahwa Anda tidak berwenang untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Aurora DSQL.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Aurora DSQL. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya Aurora DSQL saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang 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 orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah Aurora DSQL mendukung fitur-fitur ini, lihat. Bagaimana Amazon Aurora DSQL bekerja dengan IAM
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Resource-based kebijakan untuk Aurora DSQL

Gunakan kebijakan berbasis sumber daya untuk Aurora DSQL untuk membatasi atau memberikan akses ke klaster Anda melalui dokumen kebijakan JSON yang dilampirkan langsung ke sumber daya klaster Anda. Kebijakan ini memberikan kontrol halus atas siapa yang dapat mengakses klaster Anda dan dalam kondisi apa.

Cluster Aurora DSQL dapat diakses dari internet publik secara default, dengan otentikasi IAM sebagai kontrol keamanan utama. Resource-based Kebijakan memungkinkan Anda untuk menambahkan pembatasan akses, terutama untuk memblokir akses dari internet publik.

Resource-based kebijakan bekerja bersama kebijakan berbasis identitas IAM. AWS mengevaluasi kedua jenis kebijakan untuk menentukan izin akhir untuk setiap permintaan akses ke klaster Anda. Secara default, cluster Aurora DSQL dapat diakses dalam akun. Jika pengguna atau peran IAM memiliki izin Aurora DSQL, mereka dapat mengakses kluster tanpa kebijakan berbasis sumber daya yang dilampirkan.

> **Note**
>
> Perubahan kebijakan berbasis sumber daya pada akhirnya konsisten dan biasanya berlaku dalam satu menit.

Untuk informasi selengkapnya tentang perbedaan antara kebijakan berbasis identitas dan berbasis sumber daya, lihat kebijakan dan kebijakan berbasis sumber daya di Panduan Identity-based Pengguna IAM.

> **Warning**
>
> Pengguna Akun AWS root akun yang memiliki cluster selalu mempertahankan akses penuh ke sumber dayanya sendiri, terlepas dari kondisi kebijakan berbasis sumber daya, termasuk pembatasan sumber VPC. Resource-based kebijakan tidak membatasi akses untuk pengguna root. Untuk membatasi akses pengguna root ke cluster Anda, gunakan kebijakan berbasis identitas IAM atau hindari penggunaan kredenal root untuk koneksi database. Untuk informasi selengkapnya, lihat Praktik terbaik pengguna Root untuk Anda Akun AWS.

Kapan menggunakan kebijakan berbasis sumber daya

Resource-based kebijakan sangat berguna dalam skenario ini:

- Network-based kontrol akses — Batasi akses berdasarkan VPC atau alamat IP yang meminta berasal dari, atau memblokir akses internet publik sepenuhnya. Gunakan tombol kondisi seperti `aws:SourceVpc` dan `aws:SourceIp` untuk mengontrol akses jaringan.
- Beberapa tim atau aplikasi — Berikan akses ke klaster yang sama untuk beberapa tim atau aplikasi. Daripada mengelola kebijakan IAM individual untuk setiap prinsipal, Anda menentukan aturan akses sekali di klaster.
- Akses bersyarat kompleks — Kontrol akses berdasarkan beberapa faktor seperti atribut jaringan, konteks permintaan, dan atribut pengguna. Anda dapat menggabungkan beberapa kondisi dalam satu kebijakan.
- Tata kelola keamanan terpusat — Aktifkan pemilik klaster untuk mengontrol akses menggunakan sintaks AWS kebijakan yang sudah dikenal yang terintegrasi dengan praktik keamanan Anda yang ada.

> **Note**
>
> Cross-account akses belum didukung untuk kebijakan berbasis sumber daya Aurora DSQL tetapi akan tersedia di rilis mendatang.

Ketika seseorang mencoba terhubung ke klaster Aurora DSQL Anda, AWS evaluasi kebijakan berbasis sumber daya Anda sebagai bagian dari konteks otorisasi, bersama dengan kebijakan IAM yang relevan, untuk menentukan apakah permintaan tersebut harus diizinkan atau ditolak.

Resource-based kebijakan dapat memberikan akses ke prinsipal dalam AWS akun yang sama dengan klaster. Untuk klaster Multi-region, setiap kluster regional memiliki kebijakan berbasis sumber dayanya sendiri, yang memungkinkan kontrol akses bila diperlukan. Region-specific

> **Note**
>
> Kunci konteks kondisi dapat bervariasi antar Wilayah (seperti ID VPC).

Topik

- Membuat cluster dengan kebijakan berbasis sumber daya
- Menambahkan dan mengedit kebijakan berbasis sumber daya untuk klaster
- Melihat kebijakan berbasis sumber daya
- Menghapus kebijakan berbasis sumber daya
- Contoh kebijakan berbasis sumber daya umum
- Memblokir akses publik dengan kebijakan berbasis sumber daya di Aurora DSQL
- Operasi dan Kebijakan Aurora DSQL API Resource-Based

Membuat cluster dengan kebijakan berbasis sumber daya

Anda dapat melampirkan kebijakan berbasis sumber daya saat membuat klaster baru untuk memastikan kontrol akses sudah ada sejak awal. Setiap cluster dapat memiliki kebijakan inline tunggal yang dilampirkan langsung ke cluster.

AWS Konsol Manajemen

Untuk menambahkan kebijakan berbasis sumber daya selama pembuatan klaster

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Pilih Buat klaster.

3. Konfigurasikan nama klaster, tag, dan pengaturan multi-wilayah sesuai kebutuhan.
4. Di bagian Pengaturan cluster, cari opsi Resource-based kebijakan.
5. Aktifkan Tambahkan kebijakan berbasis sumber daya.
6. Masukkan dokumen kebijakan Anda di editor JSON. Misalnya, untuk memblokir akses internet publik:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

7. Anda dapat menggunakan pernyataan Edit atau Tambahkan pernyataan baru untuk membuat kebijakan Anda.
8. Selesaikan konfigurasi cluster yang tersisa dan pilih Create cluster.

AWS CLI

Gunakan `--policy` parameter saat membuat klaster untuk melampirkan kebijakan inline:

```
aws dsql create-cluster --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
```

```
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}'
```

AWS SDK

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}

response = client.create_cluster(
    policy=json.dumps(policy)
)

print(f"Cluster created: {response['identifier']}")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
```

```
DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
    }
  }]
}
""";

CreateClusterRequest request = CreateClusterRequest.builder()
    .policy(policy)
    .build();

CreateClusterResponse response = client.createCluster(request);
System.out.println("Cluster created: " + response.identifier());
```

Menambahkan dan mengedit kebijakan berbasis sumber daya untuk klaster

AWS Konsol Manajemen

Untuk menambahkan kebijakan berbasis sumber daya ke klaster yang ada

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Pilih klaster Anda dari daftar klaster untuk membuka halaman detail klaster.
3. Pilih tab Izin.
4. Di bagian Resource-based kebijakan, pilih Tambahkan kebijakan.
5. Masukkan dokumen kebijakan Anda di editor JSON. Anda dapat menggunakan pernyataan Edit atau Tambahkan pernyataan baru untuk membuat kebijakan Anda.
6. Pilih Tambahkan kebijakan.

Untuk mengedit kebijakan berbasis sumber daya yang ada

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Pilih klaster Anda dari daftar klaster untuk membuka halaman detail klaster.
3. Pilih tab Izin.
4. Di bagian Resource-based kebijakan, pilih Edit.
5. Ubah dokumen kebijakan di editor JSON. Anda dapat menggunakan pernyataan Edit atau Tambahkan pernyataan baru untuk memperbarui kebijakan Anda.
6. Pilih Simpan perubahan.

AWS CLI

Gunakan `put-cluster-policy` perintah untuk melampirkan kebijakan baru atau memperbarui kebijakan yang ada di klaster:

```
aws dsql put-cluster-policy --identifier your_cluster_id --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": { "aws:SourceVpc": "true" }
        }
    }]
}'
```

AWS SDK

Python

```
import boto3
import json

client = boto3.client('dsql')
```

```
policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": {"aws:SourceVpc": "true"}
        }
    }]
}

response = client.put_cluster_policy(
    identifier='your_cluster_id',
    policy=json.dumps(policy)
)
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.PutClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "Null": {"aws:SourceVpc": "true"}
    }
  }]
}
""";

PutClusterPolicyRequest request = PutClusterPolicyRequest.builder()
```

```
    .identifier("your_cluster_id")
    .policy(policy)
    .build();

client.putClusterPolicy(request);
```

Melihat kebijakan berbasis sumber daya

Anda dapat melihat kebijakan berbasis sumber daya yang dilampirkan ke kluster Anda untuk memahami kontrol akses saat ini.

AWS Konsol Manajemen

Untuk melihat kebijakan berbasis sumber daya

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Pilih klaster Anda dari daftar klaster untuk membuka halaman detail klaster.
3. Pilih tab Izin.
4. Lihat kebijakan terlampir di bagian Resource-based kebijakan.

AWS CLI

Gunakan `get-cluster-policy` perintah untuk melihat kebijakan berbasis sumber daya kluster:

```
aws dsql get-cluster-policy --identifier your_cluster_id
```

AWS SDK

Python

```
import boto3
import json

client = boto3.client('dsql')

response = client.get_cluster_policy(
```

```
    identifier='your_cluster_id'
)

# Parse and pretty-print the policy
policy = json.loads(response['policy'])
print(json.dumps(policy, indent=2))
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyRequest;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyResponse;

DsqlClient client = DsqlClient.create();

GetClusterPolicyRequest request = GetClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

GetClusterPolicyResponse response = client.getClusterPolicy(request);
System.out.println("Policy: " + response.policy());
```

Menghapus kebijakan berbasis sumber daya

Anda dapat menghapus kebijakan berbasis sumber daya dari kluster untuk mengubah kontrol akses.

> **Important**
>
> Saat Anda menghapus semua kebijakan berbasis sumber daya dari klaster, akses akan dikontrol sepenuhnya oleh kebijakan berbasis identitas IAM.

AWS Konsol Manajemen

Untuk menghapus kebijakan berbasis sumber daya

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/

2. Pilih klaster Anda dari daftar klaster untuk membuka halaman detail klaster.
3. Pilih tab Izin.
4. Di bagian Resource-based kebijakan, pilih Hapus.
5. Dalam dialog konfirmasi, ketik **`confirm`** untuk mengonfirmasi penghapusan.
6. Pilih Hapus.

AWS CLI

Gunakan `delete-cluster-policy` perintah untuk menghapus kebijakan dari klaster:

```
aws dsql delete-cluster-policy --identifier your_cluster_id
```

AWS SDK

Python

```
import boto3

client = boto3.client('dsql')

response = client.delete_cluster_policy(
    identifier='your_cluster_id'
)

print("Policy deleted successfully")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

DeleteClusterPolicyRequest request = DeleteClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

client.deleteClusterPolicy(request);
System.out.println("Policy deleted successfully");
```

Contoh kebijakan berbasis sumber daya umum

Contoh-contoh ini menunjukkan pola umum untuk mengontrol akses ke cluster Aurora DSQL Anda. Anda dapat menggabungkan dan memodifikasi pola-pola ini untuk memenuhi persyaratan akses spesifik Anda.

Blokir akses internet publik

Kebijakan ini memblokir koneksi ke klaster Aurora DSQL Anda dari internet publik (non-VPC). Kebijakan ini tidak menentukan dari mana pelanggan VPC dapat terhubung—hanya saja mereka harus terhubung dari VPC. Untuk membatasi akses ke VPC tertentu, gunakan `aws:SourceVpc` dengan operator `StringEquals` kondisi.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

> **Note**
>
> Contoh ini hanya digunakan `aws:SourceVpc` untuk memeriksa koneksi VPC. Kunci `aws:VpcSourceIp` dan `aws:SourceVpce` kondisi memberikan perincian tambahan tetapi tidak diperlukan untuk kontrol VPC-only akses dasar.

Untuk memberikan pengecualian untuk peran tertentu, gunakan kebijakan ini sebagai gantinya:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromOutsideVPC",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        },
        "StringNotEquals": {
          "aws:PrincipalArn": [
            "arn:aws:iam::123456789012:role/ExceptionRole",
            "arn:aws:iam::123456789012:role/AnotherExceptionRole"
          ]
        }
      }
    }
  ]
}
```

Batasi akses ke AWS Organisasi

Kebijakan ini membatasi akses ke prinsipal dalam Organisasi: AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
```

```
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "o-exampleorgid"
        }
      }
    }
  ]
}
```

Membatasi akses ke Unit Organisasi tertentu

Kebijakan ini membatasi akses ke kepala sekolah dalam Unit Organisasi (OU) tertentu dalam suatu Organisasi, memberikan kontrol yang lebih terperinci daripada akses di seluruh AWS organisasi:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotLike": {
          "aws:PrincipalOrgPaths": "o-exampleorgid/r-examplerootid/ou-exampleouid/*"
        }
      }
    }
  ]
}
```

Multi-Region kebijakan klaster

Untuk klaster Multi-region, setiap kluster regional mempertahankan kebijakan sumber dayanya sendiri, memungkinkan kontrol. Region-specific Berikut adalah contoh dengan kebijakan berbeda per wilayah:

kebijakan us-east-1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-east1-id"
        },
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

kebijakan us-east-2:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
```

```
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-east2-id"
        }
      }
    }
  ]
}
```

> **Note**
>
> Kunci konteks kondisi dapat bervariasi antara Wilayah AWS (seperti ID VPC).

Memblokir akses publik dengan kebijakan berbasis sumber daya di Aurora DSQL

Block Public Access (BPA) adalah fitur yang mengidentifikasi dan mencegah melampirkan kebijakan berbasis sumber daya yang memberikan akses publik ke cluster Aurora DSQL Anda di seluruh akun Anda. AWS Dengan BPA, Anda dapat mencegah akses publik ke sumber daya Aurora DSQL Anda. BPA melakukan pemeriksaan selama pembuatan atau modifikasi kebijakan berbasis sumber daya dan membantu meningkatkan postur keamanan Anda dengan Aurora DSQL.

BPA menggunakan penalaran otomatis untuk menganalisis akses yang diberikan oleh kebijakan berbasis sumber daya Anda dan memberi tahu Anda jika izin tersebut ditemukan pada saat mengelola kebijakan berbasis sumber daya. Analisis memverifikasi akses di semua pernyataan kebijakan berbasis sumber daya, tindakan, dan kumpulan kunci kondisi yang digunakan dalam kebijakan Anda.

> **Important**
>
> BPA membantu melindungi sumber daya Anda dengan mencegah akses publik diberikan melalui kebijakan berbasis sumber daya yang secara langsung melekat pada sumber daya Aurora DSQL Anda, seperti cluster. Selain menggunakan BPA, hati-hati memeriksa kebijakan berikut untuk mengonfirmasi bahwa mereka tidak memberikan akses publik:

- Identity-based kebijakan yang dilampirkan pada AWS prinsipal terkait (misalnya, peran IAM)
- Resource-based kebijakan yang dilampirkan pada AWS sumber daya terkait (misalnya, AWS kunci Layanan Manajemen Kunci (KMS))

Anda harus memastikan bahwa prinsipal tidak menyertakan * entri atau bahwa salah satu kunci kondisi yang ditentukan membatasi akses dari prinsipal ke sumber daya. Jika kebijakan berbasis sumber daya memberikan akses publik ke klaster Anda di seluruh akun AWS , Aurora DSQL akan memblokir Anda dari membuat atau memodifikasi kebijakan hingga spesifikasi dalam kebijakan diperbaiki dan dianggap non-publik.

Anda dapat membuat kebijakan non-publik dengan menentukan satu atau beberapa prinsip di dalam blok. `Principal` Contoh kebijakan berbasis sumber daya berikut memblokir akses publik dengan menentukan dua prinsip.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id"
}
```

Kebijakan yang membatasi akses dengan menentukan kunci kondisi tertentu juga tidak dianggap publik. Seiring dengan evaluasi prinsipal yang ditentukan dalam kebijakan berbasis sumber daya, kunci kondisi tepercaya berikut digunakan untuk menyelesaikan evaluasi kebijakan berbasis sumber daya untuk akses non-publik:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`

- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`
- `aws:Ec2InstanceSourceVpc`
- `aws:SourceOrgID`
- `aws:SourceOrgPaths`

Selain itu, agar kebijakan berbasis sumber daya bersifat non-publik, nilai untuk Amazon Resource Name (ARN) dan kunci string tidak boleh berisi wildcard atau variabel. Jika kebijakan berbasis sumber daya menggunakan `aws:PrincipalIsAWSService` kunci, Anda harus memastikan bahwa Anda telah menetapkan nilai kunci ke true.

Kebijakan berikut membatasi akses ke pengguna `Ben` di akun yang ditentukan. Kondisi tersebut membuat `Principal` terkendala dan tidak dianggap publik.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/Ben"
    }
  }
}
```

Contoh berikut dari kebijakan berbasis sumber daya non-publik kendala menggunakan operator. `sourceVPC` `StringEquals`

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dsql:*",
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Operasi dan Kebijakan Aurora DSQL API Resource-Based

Resource-based kebijakan di Aurora DSQL mengontrol akses ke operasi API tertentu. Bagian berikut mencantumkan semua operasi Aurora DSQL API yang diatur berdasarkan kategori, dengan indikasi mana yang mendukung kebijakan berbasis sumber daya.

Kolom Mendukung RBP menunjukkan apakah operasi API tunduk pada evaluasi kebijakan berbasis sumber daya saat kebijakan dilampirkan ke klaster.

Tag API

Operasi API	Deskripsi	Mendukung RBP
ListTagsForResource	Daftar tag untuk sumber daya Aurora DSQL	Ya
TagResource	Menambahkan tag ke sumber daya Aurora DSQL	Ya
UntagResource	Menghapus tag dari sumber daya Aurora DSQL	Ya

API manajemen cluster

Operasi API	Deskripsi	Mendukung RBP
CreateCluster	Membuat cluster baru	Tidak
DeleteCluster	Menghapus cluster	Ya
GetCluster	Mengambil informasi tentang cluster	Ya
GetVpcEndpointServiceName	Mengambil nama layanan titik akhir VPC untuk sebuah cluster	Ya
ListClusters	Daftar cluster di akun Anda	Tidak
UpdateCluster	Memperbarui konfigurasi cluster	Ya

Multi-Region API properti

Operasi API	Deskripsi	Mendukung RBP
AddPeerCluster	Menambahkan peer cluster ke konfigurasi multi-wilayah	Ya
PutMultiRegionProperties	Menetapkan properti multi-region untuk sebuah cluster	Ya
PutWitnessRegion	Menetapkan wilayah saksi untuk klaster multi-wilayah	Ya

Resource-based API kebijakan

Operasi API	Deskripsi	Mendukung RBP
DeleteClusterPolicy	Menghapus kebijakan berbasis sumber daya dari klaster	Ya

Operasi API	Deskripsi	Mendukung RBP
GetClusterPolicy	Mengambil kebijakan berbasis sumber daya untuk klaster	Ya
PutClusterPolicy	Membuat atau memperbarui kebijakan berbasis sumber daya untuk klaster	Ya

AWS Fault Injection Service API

Operasi API	Deskripsi	Mendukung RBP
InjectError	Menyuntikkan kesalahan untuk pengujian injeksi kesalahan	Tidak

Backup dan restore API

Operasi API	Deskripsi	Mendukung RBP
GetBackupJob	Mengambil informasi tentang pekerjaan cadangan	Tidak
GetRestoreJob	Mengambil informasi tentang pekerjaan pemulihan	Tidak
StartBackupJob	Memulai pekerjaan cadangan untuk klaster	Ya
StartRestoreJob	Memulai pekerjaan pemulihan dari cadangan	Tidak
StopBackupJob	Menghentikan pekerjaan cadangan yang sedang berjalan	Tidak
StopRestoreJob	Menghentikan pekerjaan pemulihan yang sedang berjalan	Tidak

Menggunakan peran terkait layanan di Aurora DSQL

Aurora DSQL menggunakan peran terkait layanan AWS Identity and Access Management (IAM). Peran terkait layanan adalah jenis peran IAM unik yang ditautkan langsung ke Aurora DSQL. Service-linked peran telah ditentukan sebelumnya oleh Aurora DSQL dan mencakup semua izin yang diperlukan layanan untuk memanggil Layanan AWS atas nama cluster Aurora DSQL Anda.

Service-linked peran membuat proses penyiapan lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual untuk menggunakan Aurora DSQL. Saat Anda membuat klaster, Aurora DSQL secara otomatis membuat peran terkait layanan untuk Anda. Anda dapat menghapus peran terkait layanan hanya setelah Anda menghapus semua cluster Anda. Ini melindungi sumber daya Aurora DSQL Anda karena Anda tidak dapat secara tidak sengaja menghapus izin yang diperlukan untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat layanan Layanan AWS yang berfungsi dengan IAM dan cari layanan yang memiliki Ya di kolom PeranService-Linked . Pilih Ya dengan sebuah tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Service-linked peran tersedia di semua Wilayah Aurora DSQL yang didukung.

Service-linked izin peran untuk Aurora DSQL

Aurora DSQL menggunakan peran terkait layanan bernama — Memungkinkan `AWSServiceRoleForAuroraDsql` Amazon Aurora DSQL membuat dan mengelola sumber daya atas nama Anda. AWS Peran terkait layanan ini dilampirkan ke kebijakan terkelola berikut ini: AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan. Anda mungkin menemukan pesan kesalahan berikut:`You don't have the permissions to create an Amazon Aurora DSQL service-linked role`. Jika Anda melihat pesan ini, pastikan bahwa Anda mengaktifkan izin berikut:
>
> JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateDsqlServiceLinkedRole",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "dsql.amazonaws.com"
                }
            }
        }
    ]
}
```

Untuk informasi selengkapnya, lihat izin Service-linked peran.

Buat peran tertaut layanan

Anda tidak perlu membuat peran AuroraDSQLServiceLinkedRolePolicy terkait layanan secara manual. Aurora DSQL menciptakan peran terkait layanan untuk Anda. Jika peran AuroraDSQLServiceLinkedRolePolicy terkait layanan telah dihapus dari akun Anda, Aurora DSQL akan membuat peran tersebut saat Anda membuat klaster Aurora DSQL baru.

Edit peran tertaut layanan

Aurora DSQL tidak memungkinkan Anda untuk mengedit peran terkait layanan. AuroraDSQLServiceLinkedRolePolicy Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit deskripsi peran menggunakan konsol IAM, AWS Command Line Interface (AWS CLI), atau IAM API.

Hapus peran tertaut layanan

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, sebaiknya hapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak terpakai yang tidak dipantau atau dipelihara secara aktif.

Sebelum Anda dapat menghapus peran terkait layanan untuk akun, Anda harus menghapus klaster apa pun di akun tersebut.

Anda dapat menggunakan konsol IAM, API IAM AWS CLI, atau IAM untuk menghapus peran terkait layanan. Untuk informasi selengkapnya, lihat Membuat peran terkait layanan di Panduan Pengguna IAM.

Wilayah yang Didukung untuk peran terkait layanan Aurora DSQL

Aurora DSQL mendukung penggunaan peran terkait layanan di semua Wilayah tempat layanan tersedia. Untuk informasi selengkapnya, lihat AWS Wilayah dan titik akhir.

Menggunakan kunci kondisi IAM dengan Amazon Aurora DSQL

Saat Anda memberikan izin di Aurora DSQL, Anda dapat menentukan kondisi yang menentukan bagaimana kebijakan izin diterapkan. Berikut ini adalah contoh bagaimana Anda dapat menggunakan kunci kondisi dalam kebijakan izin Aurora DSQL.

Contoh 1: Berikan izin untuk membuat klaster di tempat tertentu Wilayah AWS

Kebijakan berikut memberikan izin untuk membuat cluster di Wilayah AS Timur (Virginia N.) dan Timur AS (Ohio). Kebijakan ini menggunakan ARN sumber daya untuk membatasi Wilayah yang diizinkan, sehingga Aurora DSQL hanya dapat membuat klaster hanya jika ARN tersebut ditentukan di bagian kebijakan. `Resource`

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": ["dsql:CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
```

```
        }
    ]
}
```

Contoh 2: Berikan izin untuk membuat klaster Multi-wilayah secara spesifik Wilayah AWS detik

Kebijakan berikut memberikan izin untuk membuat klaster Multi-wilayah di Wilayah AS Timur (Virginia N.) dan Timur AS (Ohio). Kebijakan ini menggunakan ARN sumber daya untuk membatasi Wilayah yang diizinkan, sehingga Aurora DSQL dapat membuat klaster Multi-wilayah hanya jika ARN ini ditentukan di bagian kebijakan. `Resource` Perhatikan bahwa membuat klaster Multi-wilayah juga memerlukan `AddPeerCluster` izin`PutMultiRegionProperties`,`PutWitnessRegion`, dan di setiap Wilayah yang ditentukan.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "dsql:CreateCluster",
          "dsql:PutMultiRegionProperties",
          "dsql:PutWitnessRegion",
          "dsql:AddPeerCluster"
        ],
        "Resource": [
           "arn:aws:dsql:us-east-1:123456789012:cluster/*",
           "arn:aws:dsql:us-east-2:123456789012:cluster/*"
        ]
      }
    ]
}
```

Contoh 3: Berikan izin untuk membuat klaster Multi-wilayah dengan Wilayah saksi tertentu

Kebijakan berikut menggunakan kunci `dsql:WitnessRegion` kondisi Aurora DSQL dan memungkinkan pengguna membuat klaster Multi-wilayah dengan Wilayah saksi di AS Barat (Oregon). Jika Anda tidak menentukan `dsql:WitnessRegion` kondisinya, Anda dapat menggunakan Wilayah mana pun sebagai Wilayah saksi.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateWithSpecificWitness",
            "Effect": "Allow",
            "Action": [
                "dsql:CreateCluster",
                "dsql:PutWitnessRegion"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "dsql:WitnessRegion": "us-west-2"
                }
            }
        },
        {
            "Sid": "AllowMultiRegionSetup",
            "Effect": "Allow",
            "Action": [
                "dsql:PutMultiRegionProperties",
                "dsql:AddPeerCluster"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*"
        }
    ]
}
```

Respon insiden di Amazon Aurora DSQL

Keamanan adalah prioritas tertinggi di AWS. Sebagai bagian dari model tanggung jawab bersama AWS Cloud, AWS mengelola pusat data, jaringan, dan arsitektur perangkat lunak yang memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan. AWS bertanggung jawab atas setiap respons insiden sehubungan dengan layanan Amazon Aurora DSQL itu sendiri. Selain itu, sebagai AWS pelanggan, Anda berbagi tanggung jawab untuk menjaga keamanan di cloud. Ini berarti Anda mengontrol keamanan yang Anda pilih untuk diterapkan dari AWS alat dan fitur yang dapat Anda akses. Selain itu, Anda bertanggung jawab atas respons insiden di pihak Anda dari model tanggung jawab bersama.

Dengan menetapkan garis dasar keamanan yang memenuhi tujuan aplikasi Anda yang berjalan di cloud, Anda dapat mendeteksi penyimpangan yang dapat Anda tanggapi. Untuk membantu Anda memahami dampak respons insiden dan pilihan Anda terhadap tujuan perusahaan Anda, kami mendorong Anda untuk meninjau sumber daya berikut:

- AWS Panduan Respons Insiden Keamanan
- AWS Praktik Terbaik untuk Keamanan, Identitas, dan Kepatuhan
- Perspektif Keamanan dari AWS whitepaper Cloud Adoption Framework (CAF)

Amazon GuardDuty adalah layanan deteksi ancaman terkelola yang terus memantau perilaku berbahaya atau tidak sah untuk membantu pelanggan melindungi Akun AWS dan beban kerja serta mengidentifikasi aktivitas mencurigakan yang berpotensi sebelum meningkat menjadi insiden. Ini memantau aktivitas seperti panggilan API yang tidak biasa atau penerapan yang berpotensi tidak sah yang menunjukkan kemungkinan kompromi akun atau sumber daya atau pengintaian oleh aktor jahat. Misalnya, Amazon GuardDuty dapat mendeteksi aktivitas mencurigakan di Amazon Aurora DSQL API, seperti pengguna yang masuk dari lokasi baru dan membuat cluster baru.

Validasi kepatuhan untuk Amazon Aurora DSQL

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program AWS Kepatuhan Program 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 Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. Untuk informasi selengkapnya tentang tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS, lihat Dokumentasi AWS Keamanan.

Ketahanan di Amazon Aurora DSQL

Infrastruktur AWS global dibangun di sekitar Wilayah AWS dan Availability Zones (AZ). Wilayah AWS menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional. Aurora DSQL dirancang sedemikian rupa sehingga Anda dapat memanfaatkan infrastruktur AWS Regional sambil menyediakan ketersediaan database tertinggi. Secara default, kluster wilayah tunggal di Aurora DSQL memiliki Multi-AZ ketersediaan, memberikan toleransi terhadap kegagalan komponen utama dan gangguan infrastruktur yang dapat memengaruhi akses ke AZ penuh. Multi-Region cluster memberikan semua manfaat dari Multi-AZ ketahanan sambil tetap menyediakan ketersediaan database yang sangat konsisten, bahkan dalam kasus di mana tidak dapat diakses oleh Wilayah AWS klien aplikasi.

Untuk informasi selengkapnya tentang Wilayah AWS dan Availability Zone, lihat Infrastruktur AWS Global.

Selain infrastruktur AWS global, Aurora DSQL menawarkan beberapa fitur untuk membantu mendukung ketahanan data dan kebutuhan cadangan Anda.

Pencadangan dan pemulihan

Aurora DSQL mendukung pencadangan dan pemulihan dengan. Konsol AWS Backup Anda dapat melakukan pencadangan dan pemulihan penuh untuk kluster Single-region dan Multi-region Anda. Untuk informasi selengkapnya, lihat Cadangkan dan pulihkan untuk Amazon Aurora DSQL.

Replikasi

Secara desain, Aurora DSQL melakukan semua transaksi tulis ke log transaksi terdistribusi dan secara sinkron mereplikasi semua data log yang berkomitmen ke replika penyimpanan pengguna

dalam tiga AZ. Multi-Region cluster menyediakan kemampuan replikasi Lintas wilayah lengkap antara Wilayah baca dan tulis.

Wilayah saksi yang ditunjuk mendukung penulisan log-only transaksi dan tidak menggunakan penyimpanan. Wilayah Saksi tidak memiliki titik akhir. Ini berarti bahwa Wilayah saksi hanya menyimpan log transaksi terenkripsi, tidak memerlukan administrasi atau konfigurasi, dan tidak dapat diakses oleh pengguna. Jika Daerah saksi mengalami gangguan, tidak ada dampak terhadap ketersediaan klaster. Transaksi tulis mungkin mengalami sedikit peningkatan latensi sampai Wilayah saksi pulih.

Log transaksi Aurora DSQL dan penyimpanan pengguna didistribusikan dengan semua data yang disajikan ke prosesor kueri Aurora DSQL sebagai volume logis tunggal. Aurora DSQL secara otomatis membagi, menggabungkan, dan mereplikasi data berdasarkan rentang kunci utama basis data dan pola akses. Aurora DSQL secara otomatis menskalakan replika baca, baik naik maupun turun, berdasarkan frekuensi akses baca.

Replika penyimpanan cluster didistribusikan di seluruh armada penyimpanan multi-penyewa. Jika komponen atau AZ menjadi rusak, Aurora DSQL secara otomatis mengalihkan akses ke komponen yang masih ada dan secara asinkron memperbaiki replika yang hilang. Setelah Aurora DSQL memperbaiki replika yang rusak, Aurora DSQL secara otomatis menambahkannya kembali ke kuorum penyimpanan dan membuatnya tersedia untuk cluster Anda.

Ketersediaan tinggi

Secara default, kluster Single-region dan Multi-region di Aurora DSQL adalah aktif-aktif, dan Anda tidak perlu menyediakan, mengonfigurasi, atau mengkonfigurasi ulang cluster apa pun secara manual. Aurora DSQL sepenuhnya mengotomatiskan pemulihan cluster, yang menghilangkan kebutuhan untuk operasi failover primer-sekunder tradisional. Replikasi selalu sinkron dan dilakukan dalam beberapa AZ, sehingga tidak ada risiko kehilangan data karena replikasi lag atau failover ke database sekunder asinkron selama pemulihan kegagalan.

Single-Region cluster menyediakan endpoint Multi-AZ redundan yang secara otomatis memungkinkan akses bersamaan dengan konsistensi data yang kuat di tiga AZ. Ini berarti bahwa replika penyimpanan pengguna pada salah satu dari ketiga AZ ini selalu mengembalikan hasil yang sama ke satu atau lebih pembaca dan selalu tersedia untuk menerima penulisan. Konsistensi dan Multi-AZ ketahanan yang kuat ini tersedia di semua Wilayah untuk klaster Multi-wilayah Aurora DSQL. Ini berarti bahwa klaster Multi-wilayah menyediakan dua titik akhir Regional yang sangat konsisten, sehingga klien dapat membaca atau menulis tanpa pandang bulu ke salah satu Wilayah tanpa jeda replikasi pada komit.

Aurora DSQL menyediakan 99,99% ketersediaan untuk kluster Single-region dan 99,999% untuk cluster Multi-region.

Pengujian injeksi kesalahan

Amazon Aurora DSQL terintegrasi dengan AWS Fault Injection Service (AWS FIS), layanan yang dikelola sepenuhnya untuk menjalankan eksperimen injeksi kesalahan terkontrol guna meningkatkan ketahanan aplikasi. Menggunakan AWS FIS, Anda dapat:

- Buat templat eksperimen yang menentukan skenario kegagalan tertentu
- Kegagalan injeksi (tingkat kesalahan koneksi cluster yang meningkat) untuk memvalidasi penanganan kesalahan aplikasi dan mekanisme pemulihan
- Uji perilaku aplikasi Multi-wilayah untuk memvalidasi pergeseran lalu lintas aplikasi antara Wilayah AWS saat seseorang Wilayah AWS mengalami tingkat kesalahan koneksi yang tinggi

Misalnya, dalam cluster Multi-region yang mencakup US East (Virginia N.) dan US East (Ohio), Anda dapat menjalankan eksperimen di US East (Ohio) untuk menguji kegagalan di sana sementara US East (Virginia N.) melanjutkan operasi normal. Pengujian terkontrol ini membantu Anda mengidentifikasi dan menyelesaikan masalah potensial sebelum memengaruhi beban kerja produksi.

Lihat Target tindakan di panduan AWS FIS pengguna untuk daftar lengkap tindakan yang AWS FIS didukung.

Untuk informasi tentang tindakan DSQL Amazon Aurora yang tersedia di, AWS FIS lihat referensi tindakan Aurora DSQL di Panduan Pengguna.AWS FIS

Untuk mulai menjalankan eksperimen injeksi kesalahan, lihat Merencanakan AWS FIS eksperimen Anda di Panduan AWS FIS Pengguna.

Keamanan Infrastruktur di Amazon Aurora DSQL

Sebagai layanan terkelola, Amazon Aurora DSQL dilindungi oleh prosedur keamanan jaringan AWS global yang dijelaskan dalam Praktik Terbaik untuk Keamanan, Identitas, & Kepatuhan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses Aurora DSQL melalui jaringan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2 atau versi yang lebih baru. Klien juga harus mendukung cipher suite dengan perfect forward secrecy (PFS) seperti DHE (Ephemeral) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). 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 principal IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Mengelola dan menghubungkan ke cluster DSQL Amazon Aurora menggunakan AWS PrivateLink

Dengan AWS PrivateLink Amazon Aurora DSQL, Anda dapat menyediakan antarmuka titik akhir Amazon VPC (titik akhir antarmuka) di Amazon Virtual Private Cloud Anda. Titik akhir ini dapat diakses langsung dari aplikasi yang ada di lokasi melalui Amazon VPC Direct Connect dan, atau berbeda dengan Wilayah AWS peering VPC Amazon. Menggunakan AWS PrivateLink dan antarmuka endpoint, Anda dapat menyederhanakan konektivitas jaringan pribadi dari aplikasi Anda ke Aurora DSQL.

Aplikasi dalam VPC Amazon Anda dapat mengakses Aurora DSQL menggunakan titik akhir antarmuka Amazon VPC tanpa memerlukan alamat IP publik.

Endpoint antarmuka diwakili oleh satu atau lebih antarmuka jaringan elastis (ENI) yang diberi alamat IP pribadi dari subnet di VPC Amazon Anda. Permintaan ke Aurora DSQL melalui titik akhir antarmuka tetap ada di jaringan. AWS Untuk informasi selengkapnya tentang cara menghubungkan VPC Amazon dengan jaringan lokal, lihat Panduan Direct Connect Pengguna dan Panduan Pengguna AWS Site-to-Site VPN VPN.

Untuk informasi umum tentang titik akhir antarmuka, lihat Mengakses AWS layanan menggunakan antarmuka titik akhir Amazon VPC di AWS PrivateLinkPanduan Pengguna.

Jenis titik akhir VPC Amazon untuk Aurora DSQL

Aurora DSQL membutuhkan dua jenis endpoint yang berbeda. AWS PrivateLink

1. Endpoint manajemen — Endpoint ini digunakan untuk operasi administratif, seperti,,`get`, `create` `update``delete`, dan pada cluster `list` Aurora DSQL. Lihat Mengelola cluster Aurora DSQL menggunakan AWS PrivateLink.
2. Connection endpoint — Endpoint ini digunakan untuk menghubungkan ke cluster Aurora DSQL melalui klien PostgreSQL. Lihat Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink.

Pertimbangan saat menggunakan AWS PrivateLink para Aurora DSQL

Pertimbangan Amazon VPC berlaku untuk AWS PrivateLink Aurora DSQL. Untuk informasi selengkapnya, lihat Mengakses AWS layanan menggunakan titik akhir VPC antarmuka dan AWS PrivateLink kuota di Panduan. AWS PrivateLink

Mengelola cluster Aurora DSQL menggunakan AWS PrivateLink

Anda dapat menggunakan AWS Command Line Interface atau AWS Software Development Kit (SDK) untuk mengelola cluster Aurora DSQL melalui titik akhir antarmuka Aurora DSQL.

Membuat titik akhir Amazon VPC

Untuk membuat titik akhir antarmuka VPC Amazon, lihat Membuat titik akhir VPC Amazon di Panduan. AWS PrivateLink

```
aws ec2 create-vpc-endpoint \
--region region \
--service-name com.amazonaws.region.dsql \
--vpc-id your-vpc-id \
--subnet-ids your-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id \
```

Untuk menggunakan nama DNS Regional default untuk permintaan API Aurora DSQL, jangan nonaktifkan DNS pribadi saat Anda membuat titik akhir antarmuka Aurora DSQL. Saat DNS pribadi diaktifkan, permintaan ke layanan Aurora DSQL yang dibuat dari dalam VPC Amazon Anda akan secara otomatis diselesaikan ke alamat IP pribadi titik akhir VPC Amazon, bukan nama DNS publik. Saat DNS pribadi diaktifkan, permintaan Aurora DSQL yang dibuat dalam VPC Amazon Anda akan secara otomatis diselesaikan ke titik akhir VPC Amazon Anda.

Jika DNS pribadi tidak diaktifkan, gunakan `--endpoint-url` parameter `--region` dan dengan AWS CLI perintah untuk mengelola cluster Aurora DSQL melalui titik akhir antarmuka Aurora DSQL.

Daftar cluster menggunakan URL endpoint

Dalam contoh berikut, ganti Wilayah AWS `us-east-1` dan nama DNS `vpce-1a2b3c4d-5e6f.dsql.us-east-1.vpce.amazonaws.com` ID endpoint Amazon VPC dengan informasi Anda sendiri.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-east-1.vpce.amazonaws.com list-clusters
```

Operasi API

Lihat referensi Aurora DSQL API untuk dokumentasi pengelolaan sumber daya di Aurora DSQL.

Mengelola kebijakan endpoint

Dengan menguji dan mengonfigurasi kebijakan titik akhir VPC Amazon secara menyeluruh, Anda dapat membantu memastikan bahwa klaster Aurora DSQL Anda aman, sesuai, dan selaras dengan kontrol akses dan persyaratan tata kelola khusus organisasi Anda.

Contoh: Kebijakan akses DSQL Aurora Penuh

Kebijakan berikut memberikan akses penuh ke semua tindakan dan sumber daya Aurora DSQL melalui titik akhir VPC Amazon yang ditentukan.

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-xxxxxxxxxxxxxxxx \
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "dsql:*",
          "Resource": "*"
        }
      ]
    }'
```

Contoh: Kebijakan Akses Aurora DSQL Terbatas

Kebijakan berikut hanya mengizinkan tindakan Aurora DSQL ini.

- `CreateCluster`
- `GetCluster`
- `ListClusters`

Semua tindakan Aurora DSQL lainnya ditolak.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink

Setelah AWS PrivateLink endpoint Anda diatur dan aktif, Anda dapat terhubung ke cluster Aurora DSQL Anda menggunakan klien PostgreSQL. Instruksi koneksi di bawah ini menguraikan langkah-langkah untuk membangun nama host yang tepat untuk menghubungkan melalui titik akhir. AWS PrivateLink

Menyiapkan sebuah AWS PrivateLink titik akhir koneksi

Langkah 1: Dapatkan nama layanan untuk cluster Anda

Saat membuat AWS PrivateLink endpoint untuk menghubungkan ke cluster Anda, Anda harus terlebih dahulu mengambil nama layanan khusus cluster.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Contoh tanggapan

```
{
    "serviceName": "com.amazonaws.us-east-1.dsql-fnh4"
}
```

Nama layanan termasuk pengenal, seperti `dsql-fnh4` dalam contoh. Pengenal ini juga diperlukan saat membuat nama host untuk menghubungkan ke cluster Anda.

AWS SDK for Python (Boto3)

```
import boto3

dsql_client = boto3.client('dsql', region_name='us-east-1')
response = dsql_client.get_vpc_endpoint_service_name(
    identifier='your-cluster-id'
)
service_name = response['serviceName']
print(f"Service Name: {service_name}")
```

AWS SDK for Java 2.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameRequest;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameResponse;

String region = "us-east-1";
String clusterId = "your-cluster-id";

DsqlClient dsqlClient = DsqlClient.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

GetVpcEndpointServiceNameResponse response = dsqlClient.getVpcEndpointServiceName(
    GetVpcEndpointServiceNameRequest.builder()
        .identifier(clusterId)
        .build()
);
String serviceName = response.serviceName();
System.out.println("Service Name: " + serviceName);
```

Langkah 2: Buat titik akhir Amazon VPC

Menggunakan nama layanan yang diperoleh pada langkah sebelumnya, buat titik akhir VPC Amazon.

> **Important**
>
> Petunjuk koneksi di bawah ini hanya berfungsi untuk menghubungkan ke cluster saat privat diaktifkan DNS. Jangan gunakan `--no-private-dns-enabled` bendera saat membuat titik akhir, karena ini akan mencegah instruksi koneksi di bawah ini berfungsi dengan baik. Jika Anda menonaktifkan DNS pribadi, Anda harus membuat catatan DNS pribadi wildcard Anda sendiri yang menunjuk ke titik akhir yang dibuat.

AWS CLI

```
aws ec2 create-vpc-endpoint \
    --region us-east-1 \
    --service-name service-name-for-your-cluster \
    --vpc-id your-vpc-id \
    --subnet-ids subnet-id-1 subnet-id-2  \
    --vpc-endpoint-type Interface \
    --security-group-ids security-group-id
```

Contoh respon

```
{
    "VpcEndpoint": {
        "VpcEndpointId": "vpce-0123456789abcdef0",
        "VpcEndpointType": "Interface",
        "VpcId": "vpc-0123456789abcdef0",
        "ServiceName": "com.amazonaws.us-east-1.dsql-fnh4",
        "State": "pending",
        "RouteTableIds": [],
        "SubnetIds": [
            "subnet-0123456789abcdef0",
            "subnet-0123456789abcdef1"
        ],
        "Groups": [
            {
                "GroupId": "sg-0123456789abcdef0",
                "GroupName": "default"
```

```
            }
        ],
        "PrivateDnsEnabled": true,
        "RequesterManaged": false,
        "NetworkInterfaceIds": [
            "eni-0123456789abcdef0",
            "eni-0123456789abcdef1"
        ],
        "DnsEntries": [
            {
                "DnsName": "*.dsql-fnh4.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            }
        ],
        "CreationTimestamp": "2025-01-01T00:00:00.000Z"
    }
}
```

SDK for Python

```
import boto3

ec2_client = boto3.client('ec2', region_name='us-east-1')
response = ec2_client.create_vpc_endpoint(
    VpcEndpointType='Interface',
    VpcId='your-vpc-id',
    ServiceName='com.amazonaws.us-east-1.dsql-fnh4',  # Use the service name from
 previous step
    SubnetIds=[
        'subnet-id-1',
        'subnet-id-2'
    ],
    SecurityGroupIds=[
        'security-group-id'
    ]
)

vpc_endpoint_id = response['VpcEndpoint']['VpcEndpointId']
print(f"VPC Endpoint created with ID: {vpc_endpoint_id}")
```

SDK for Java 2.x

Gunakan URL endpoint untuk Aurora DSQL API

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.CreateVpcEndpointRequest;
import software.amazon.awssdk.services.ec2.model.CreateVpcEndpointResponse;
import software.amazon.awssdk.services.ec2.model.VpcEndpointType;

String region = "us-east-1";
String serviceName = "com.amazonaws.us-east-1.dsql-fnh4";  // Use the service name
 from previous step
String vpcId = "your-vpc-id";

Ec2Client ec2Client = Ec2Client.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

CreateVpcEndpointRequest request = CreateVpcEndpointRequest.builder()
    .vpcId(vpcId)
    .serviceName(serviceName)
    .vpcEndpointType(VpcEndpointType.INTERFACE)
    .subnetIds("subnet-id-1", "subnet-id-2")
    .securityGroupIds("security-group-id")
    .build();

CreateVpcEndpointResponse response = ec2Client.createVpcEndpoint(request);
String vpcEndpointId = response.vpcEndpoint().vpcEndpointId();
System.out.println("VPC Endpoint created with ID: " + vpcEndpointId);
```

Pengaturan tambahan saat menghubungkan melalui Direct Connect atau peering VPC Amazon

Beberapa pengaturan tambahan mungkin diperlukan untuk terhubung ke cluster Aurora DSQL menggunakan titik akhir AWS PrivateLink koneksi dari perangkat on-premise melalui Amazon VPC peering atau. Direct Connect Penyiapan ini tidak diperlukan jika aplikasi Anda berjalan di VPC Amazon yang sama dengan titik akhir Anda AWS PrivateLink . Entri DNS pribadi yang dibuat di atas tidak akan diselesaikan dengan benar di luar VPC Amazon titik akhir, tetapi Anda dapat membuat catatan DNS pribadi Anda sendiri yang menyelesaikan titik akhir koneksi Anda. AWS PrivateLink

Buat data DNS CNAME pribadi yang menunjuk ke nama domain yang sepenuhnya memenuhi syarat dari AWS PrivateLink titik akhir. Nama domain dari catatan DNS yang dibuat harus dibangun dari komponen-komponen berikut:

1. Pengidentifikasi layanan dari nama layanan. Misalnya: `dsql-fnh4`
2. The Wilayah AWS

Buat catatan DNS CNAME dengan nama domain dalam format berikut: `*.service-identifier.region.on.aws`

Format nama domain penting karena dua alasan:

1. Nama host yang digunakan untuk terhubung ke Aurora DSQL harus sesuai dengan sertifikat server Aurora DSQL saat menggunakan mode SSL. `verify-full` Ini memastikan tingkat keamanan koneksi tertinggi.
2. Aurora DSQL menggunakan bagian ID cluster dari nama host yang digunakan untuk terhubung ke Aurora DSQL untuk mengidentifikasi cluster penghubung.

Jika membuat catatan DNS pribadi tidak memungkinkan, Anda masih dapat terhubung ke Aurora DSQL. Lihat Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink titik akhir tanpa DNS pribadi.

Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink titik akhir koneksi

Setelah AWS PrivateLink titik akhir Anda diatur dan aktif (periksa apakah ada`available`), Anda dapat terhubung ke cluster Aurora DSQL Anda menggunakan klien PostgreSQL. `State` Untuk petunjuk tentang penggunaan AWS SDK, Anda dapat mengikuti panduan dalam Pemrograman dengan Aurora DSQL. Anda harus mengubah titik akhir cluster agar sesuai dengan format nama host.

Membangun nama host

Nama host untuk menghubungkan AWS PrivateLink berbeda dari nama host DNS publik. Anda perlu membangunnya menggunakan komponen-komponen berikut.

1. `Your-cluster-id`
2. Pengidentifikasi layanan dari nama layanan. Misalnya: `dsql-fnh4`
3. The Wilayah AWS. Misalnya: `us-east-1`

Gunakan format berikut: *cluster-id*.*service-identifier*.*region*.on.aws

Contoh: Koneksi Menggunakan PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export SERVICE_IDENTIFIER=dsql-fnh4  # This should match the identifier in your service
 name

# Construct the hostname
export HOSTNAME="$CLUSTERID.$SERVICE_IDENTIFIER.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $HOSTNAME)

# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink titik akhir tanpa DNS pribadi

Petunjuk koneksi di atas bergantung pada catatan DNS pribadi. Jika aplikasi Anda berjalan di VPC Amazon yang sama dengan AWS PrivateLink titik akhir Anda, catatan DNS dibuat untuk Anda. Atau, jika Anda terhubung dari perangkat lokal melalui peering VPC Amazon atau Direct Connect, Anda dapat membuat catatan DNS pribadi Anda sendiri. Namun, penyiapan catatan DNS tidak selalu dimungkinkan karena pembatasan jaringan yang diberlakukan oleh tim keamanan Anda. Jika aplikasi Anda harus terhubung menggunakan Direct Connect atau dari VPC Amazon peered, dan penyiapan catatan DNS tidak memungkinkan, Anda masih dapat terhubung ke Aurora DSQL.

Aurora DSQL menggunakan bagian ID cluster dari nama host Anda untuk mengidentifikasi cluster penghubung, tetapi jika pengaturan catatan DNS tidak memungkinkan, Aurora DSQL mendukung menentukan cluster target menggunakan opsi koneksi. `amzn-cluster-id` Dengan opsi ini, Anda dapat menggunakan nama domain yang sepenuhnya memenuhi syarat dari AWS PrivateLink titik akhir Anda sebagai nama host Anda saat menghubungkan.

> **Important**
>
> Saat menghubungkan dengan nama domain atau alamat IP AWS PrivateLink titik akhir yang sepenuhnya memenuhi syarat, mode `verify-full` SSL tidak didukung. Untuk alasan ini, pengaturan DNS pribadi lebih disukai.

Contoh: Menentukan opsi koneksi ID cluster menggunakan PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export HOSTNAME=vpce-04037adb76c111221-d849uc2p.dsql-fnh4.us-east-1.vpce.amazonaws.com
 # This should match your endpoint's fully-qualified domain name

# Construct the hostname used to generate the authentication token
export AUTH_HOSTNAME="$CLUSTERID.dsql.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $AUTH_HOSTNAME)

# Specify the amzn-cluster-id connection option
export PGOPTIONS="-c amzn-cluster-id=$CLUSTERID"

# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

Memecahkan masalah dengan AWS PrivateLink

Masalah dan Solusi Umum

Tabel berikut mencantumkan masalah umum dan solusi yang berkaitan AWS PrivateLink dengan Aurora DSQL.

Isu	Kemungkinan penyebab	Solusi
Batas waktu koneksi	Grup keamanan tidak dikonfigurasi dengan benar	Gunakan Amazon VPC Reachability Analyzer untuk memastikan penyiapan jaringan Anda memungkinkan lalu lintas di port 5432.
Kegagalan resolusi DNS	DNS pribadi tidak diaktifkan	Verifikasi bahwa titik akhir VPC Amazon dibuat dengan DNS pribadi diaktifkan.
Kegagalan otentikasi	Kredensi salah atau token kedaluwarsa	Buat token otentikasi baru dan verifikasi nama pengguna.
Nama layanan tidak ditemukan	ID cluster salah	Double-check ID cluster Anda dan Wilayah AWS saat mengambil nama layanan.

Sumber Daya Terkait

Untuk informasi selengkapnya, lihat sumber daya berikut:

- Panduan Pengguna Amazon Aurora DSQL
- Dokumentasi AWS PrivateLink
- Akses AWS layanan melalui AWS PrivateLink

Analisis konfigurasi dan kerentanan di Amazon Aurora DSQL

AWS menangani tugas-tugas keamanan dasar seperti sistem operasi tamu (OS) dan patching database, konfigurasi firewall, dan pemulihan bencana. Prosedur ini telah ditinjau dan disertifikasi oleh pihak ketiga yang sesuai. Untuk detail selengkapnya, lihat sumber daya berikut:

- Model tanggung jawab bersama
- Amazon Web Services: Ikhtisar proses keamanan (whitepaper)

Cross-service pencegahan wakil bingung

Masalah "confused deputy" adalah masalah keamanan di mana entitas yang tidak memiliki izin untuk melakukan tindakan dapat memengaruhi entitas yang memiliki hak akses lebih tinggi untuk melakukan tindakan. Pada tahun AWS, peniruan lintas layanan dapat mengakibatkan masalah wakil yang membingungkan. Cross-service peniruan identitas dapat terjadi ketika satu layanan (layanan panggilan) memanggil layanan lain (layanan yang disebut). 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 principal layanan yang telah diberi akses ke sumber daya di akun Anda.

Sebaiknya gunakan kunci konteks kondisi global `aws:SourceAccount` dan global `aws:SourceArn` dalam kebijakan sumber daya untuk membatasi izin yang diberikan Amazon Aurora DSQL layanan lain ke sumber daya. Gunakan `aws:SourceArn` jika Anda ingin hanya satu sumber daya yang akan dikaitkan dengan akses lintas layanan. Gunakan `aws:SourceAccount` jika Anda ingin mengizinkan sumber daya apa pun di akun tersebut dikaitkan dengan penggunaan lintas layanan.

Cara paling efektif untuk melindungi dari masalah "confused deputy" adalah dengan menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN lengkap sumber daya. Jika Anda tidak mengetahui ARN lengkap sumber daya atau jika Anda menentukan beberapa sumber daya, gunakan kunci kondisi konteks global `aws:SourceArn` dengan karakter wildcard (*) untuk bagian ARN yang tidak diketahui. Misalnya, `arn:aws:dsql:*:123456789012:*`.

Jika nilai `aws:SourceArn` tidak berisi ID akun, seperti ARN bucket Amazon S3, Anda harus menggunakan kedua kunci konteks kondisi global tersebut untuk membatasi izin.

Nilai `aws:SourceArn` harus ARN dari sumber daya Aurora DSQL yang peran layanan bertindak atas nama.

Contoh berikut menunjukkan bagaimana Anda dapat menggunakan `aws:SourceArn` dan kunci konteks kondisi `aws:SourceAccount` global di Aurora DSQL untuk mencegah masalah wakil bingung.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
```

```
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "backup.amazonaws.com"
    },
    "Action": "dsql:GetCluster",
    "Resource": [
      "arn:aws:dsql:*:123456789012:cluster/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:backup:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Peran layanan aliran CDC

Aliran perubahan pengambilan data (CDC) memerlukan peran layanan IAM yang diasumsikan Aurora DSQL untuk menulis catatan CDC ke target Anda. Saat Anda membuat peran ini, gunakan `aws:SourceAccount` dan `aws:SourceArn` ketentuan dalam kebijakan kepercayaan untuk memastikan bahwa hanya aliran CDC di akun Anda yang dapat mengambil peran tersebut.

Setel `aws:SourceArn` ke pola ARN aliran untuk cluster yang menggunakan peran. Karena Aurora DSQL belum menetapkan pengenal aliran saat Anda membuat aliran, gunakan wildcard untuk bagian aliran ARN:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DSQLAssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": "dsql.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "your-account-id"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:dsql:region:your-account-
id:cluster/cluster-id/stream/*"
                }
            }
        }
    ]
}
```

Setelah Anda membuat aliran, Anda dapat mengencangkan `aws:SourceArn` ke ARN streaming yang tepat jika peran tersebut melayani satu aliran. Untuk penjelasan selengkapnya tentang kebijakan kepercayaan dan kebijakan izin untuk peran layanan CDC, lihat. Mengkonfigurasi IAM

Praktik terbaik keamanan untuk Aurora DSQL

Aurora DSQL menyediakan sejumlah fitur keamanan untuk dipertimbangkan saat Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau tidak memadai untuk lingkungan Anda, perlakukan itu sebagai pertimbangan yang bermanfaat, bukan sebagai resep.

Topik

- Praktik terbaik keamanan Detektif untuk Aurora DSQL
- Praktik terbaik keamanan preventif untuk Aurora DSQL

Praktik terbaik keamanan Detektif untuk Aurora DSQL

Selain cara-cara berikut untuk menggunakan Aurora DSQL dengan aman, lihat Keamanan AWS Well-Architected Tool untuk mempelajari tentang bagaimana teknologi cloud meningkatkan keamanan Anda.

CloudWatch Alarm Amazon

> Menggunakan CloudWatch alarm Amazon, Anda menonton satu metrik selama periode waktu yang Anda tentukan. Jika metrik melebihi ambang batas tertentu, pemberitahuan akan 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.

Tandai sumber daya Aurora DSQL Anda untuk identifikasi dan otomatisasi

Anda dapat menetapkan metadata ke AWS sumber daya Anda dalam bentuk tag. Setiap tanda adalah label sederhana yang terdiri dari kunci yang ditetapkan pelanggan dan nilai opsional yang memudahkan untuk mengelola, mencari, dan memfilter sumber daya.

Penandaan memungkinkan implementasi kontrol berkelompok. Meskipun tidak ada jenis tanda yang melekat, tanda memungkinkan Anda untuk mengelompokkan sumber daya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Berikut ini beberapa contohnya:

- Keamanan – Digunakan untuk menentukan persyaratan seperti enkripsi.
- Kerahasiaan – Sebuah pengidentifikasi untuk dukungan sumber daya tingkat kerahasiaan data tertentu.
- Lingkungan – Digunakan untuk membedakan antara pengembangan, pengujian, dan infrastruktur produksi.

Anda dapat menetapkan metadata ke AWS sumber daya Anda dalam bentuk tag. Setiap tanda adalah label sederhana yang terdiri dari kunci yang ditetapkan pelanggan dan nilai opsional yang memudahkan untuk mengelola, mencari, dan memfilter sumber daya.

Penandaan memungkinkan implementasi kontrol berkelompok. Meskipun tidak ada jenis tag yang melekat, tag memungkinkan Anda untuk mengelompokkan sumber daya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Berikut ini adalah beberapa contoh.

- Keamanan — digunakan untuk menentukan persyaratan seperti enkripsi.
- Kerahasiaan — pengenal untuk tingkat kerahasiaan data tertentu yang didukung sumber daya.
- Lingkungan — digunakan untuk membedakan antara pembangunan, pengujian, dan infrastruktur produksi.

Untuk informasi selengkapnya, lihat Praktik Terbaik untuk Menandai AWS Sumber Daya.

Praktik terbaik keamanan preventif untuk Aurora DSQL

Selain cara-cara berikut untuk menggunakan Aurora DSQL dengan aman, lihat Keamanan AWS Well-Architected Tool untuk mempelajari tentang bagaimana teknologi cloud meningkatkan keamanan Anda.

Gunakan peran IAM untuk mengautentikasi akses ke Aurora DSQL.

Pengguna, aplikasi, dan lainnya Layanan AWS yang mengakses Aurora DSQL harus menyertakan AWS kredensi yang valid dalam API dan permintaan. AWS AWS CLI Anda tidak boleh menyimpan AWS kredensyal secara langsung di aplikasi atau instans EC2. Ini adalah kredensyal jangka panjang yang tidak diputar secara otomatis. Ada dampak bisnis yang signifikan jika kredensil ini dikompromikan. Peran IAM memungkinkan Anda memperoleh kunci akses sementara yang dapat Anda gunakan untuk mengakses Layanan AWS dan sumber daya.

Untuk informasi selengkapnya, lihat Otentikasi dan otorisasi untuk Aurora DSQL.

Gunakan kebijakan IAM untuk otorisasi dasar Aurora DSQL.

Saat Anda memberikan izin, Anda memutuskan siapa yang mendapatkannya, operasi Aurora DSQL API mana yang mereka dapatkan izin, dan tindakan spesifik yang ingin Anda izinkan pada sumber daya tersebut. Menerapkan akses hak akses paling rendah adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat disebabkan oleh kesalahan atau niat jahat.

Lampirkan kebijakan izin ke peran IAM dan berikan izin untuk melakukan operasi pada sumber daya Aurora DSQL. Juga tersedia batas izin untuk entitas IAM, yang memungkinkan Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM.

Mirip dengan praktik terbaik pengguna root untuk Anda Akun AWS, jangan gunakan `admin` peran di Aurora DSQL untuk melakukan operasi sehari-hari. Sebagai gantinya, kami menyarankan Anda membuat peran basis data khusus untuk mengelola dan terhubung ke klaster Anda. Untuk informasi selengkapnya, lihat Mengakses Aurora DSQL dan Memahami otentikasi dan otorisasi untuk Aurora DSQL.

Gunakan **`verify-full`** di lingkungan produksi.

Pengaturan ini memverifikasi bahwa sertifikat server ditandatangani oleh otoritas sertifikat tepercaya dan bahwa nama host server cocok dengan sertifikat.

Perbarui klien PostgreSQL Anda

Perbarui klien PostgreSQL Anda secara teratur ke versi terbaru untuk mendapatkan manfaat dari peningkatan keamanan. Kami merekomendasikan menggunakan PostgreSQL versi 17.

Menandai sumber daya di Aurora DSQL

Di AWS, tag adalah pasangan nilai kunci yang ditentukan pengguna yang Anda tentukan dan kaitkan dengan sumber daya Aurora DSQL seperti cluster dan aliran CDC. Tanda adalah opsional. Jika Anda memberikan kunci, nilainya opsional.

Anda dapat menggunakan Konsol Manajemen AWS,, atau AWS SDK untuk menambahkan AWS CLI, membuat daftar, dan menghapus tag pada cluster Aurora DSQL dan aliran CDC. Anda dapat menambahkan tag selama dan setelah pembuatan sumber daya dengan menggunakan AWS konsol. Untuk menandai sumber daya setelah pembuatan dengan AWS CLI, gunakan `TagResource` operasi.

Menandai cluster dengan Nama

Aurora DSQL membuat cluster dengan pengenal unik global yang ditetapkan sebagai Amazon Resource Name (ARN). Jika Anda ingin menetapkan nama yang ramah pengguna ke klaster Anda, kami sarankan Anda menggunakan Tag.

Jika Anda membuat konsol dengan konsol Aurora DSQL, Aurora DSQL secara otomatis membuat tag. Tag ini memiliki kunci Nama dan nilai yang dihasilkan secara otomatis yang mewakili nama cluster. Nilai ini dapat dikonfigurasi, sehingga Anda dapat menetapkan nama yang lebih ramah ke cluster Anda. Jika sebuah cluster memiliki tag Nama dengan nilai terkait, Anda dapat melihat nilai di seluruh konsol Aurora DSQL.

Persyaratan penandaan

Tag memiliki persyaratan sebagai berikut:

- Kunci tidak dapat diawali dengan`aws:`.
- Kunci harus unik per set tag.
- Kunci harus antara 1 dan 128 karakter yang diizinkan.
- Nilai harus antara 0 dan 256 karakter yang diizinkan.
- Nilai tidak harus unik per set tag.
- Karakter yang diizinkan untuk kunci dan nilai adalah huruf, digit, spasi putih, dan salah satu simbol berikut: _.:/= + - @.
- Kunci dan nilai peka huruf besar dan kecil.

Menandai aliran CDC

Aliran CDC adalah sumber daya yang dapat diberi tag secara independen. Anda dapat menambahkan tag saat membuat aliran dengan meneruskan `--tags` parameter ke`CreateStream`, dan Anda dapat membaca, menambah, atau menghapus tag pada aliran yang ada dengan menggunakan`ListTagsForResource`,`TagResource`, dan `UntagResource` dengan aliran ARN. Tag pada aliran CDC terpisah dari tag di cluster induk dan dari tag di aliran data Amazon Kinesis tujuan.

Arn aliran CDC memiliki format. `arn:aws:dsql:`*`region`*`:`*`account-id`*`:cluster/`*`cluster-id`*`/stream/`*`stream-id`* Untuk informasi selengkapnya tentang aliran CDC, lihat Mengubah aliran pengambilan data (CDC).

Menandai catatan penggunaan

Saat menggunakan tag di Aurora DSQL, pertimbangkan hal berikut.

- Saat menggunakan operasi Aurora DSQL API AWS CLI atau Aurora, pastikan untuk memberikan Nama Sumber Daya Amazon (ARN) agar sumber daya Aurora DSQL dapat digunakan. Untuk informasi selengkapnya, lihat format Nama Sumber Daya Amazon (ARN) untuk sumber daya Aurora DSQL.
- Setiap sumber daya memiliki satu set tag, yang merupakan kumpulan dari satu atau beberapa tag yang ditetapkan ke sumber daya.
- Setiap sumber daya dapat memiliki hingga 50 tag per set tag.
- Jika Anda menghapus sumber daya, tag terkait akan dihapus.
- Anda dapat menambahkan tag saat membuat sumber daya, Anda dapat melihat dan memodifikasi tag menggunakan operasi API berikut:`TagResource`,`UntagResource`, dan`ListTagsForResource`.
- Anda dapat menggunakan tag dengan kebijakan IAM. Anda dapat menggunakannya untuk mengelola akses ke cluster Aurora DSQL dan untuk mengontrol tindakan apa yang dapat diterapkan pada sumber daya tersebut. Untuk mempelajari lebih lanjut, lihat Mengontrol akses ke AWS sumber daya menggunakan tag.
- Anda dapat menggunakan tag untuk berbagai aktivitas lainnya AWS. Untuk mempelajari lebih lanjut, lihat Strategi penandaan umum.

Pertimbangan untuk bekerja dengan Amazon Aurora DSQL

Pertimbangkan perilaku berikut saat Anda bekerja dengan Amazon Aurora DSQL. Untuk informasi selengkapnya tentang kompatibilitas dan dukungan PostgreSQL, lihat. Kompatibilitas fitur SQL di Aurora DSQL Untuk kuota dan batasan, lihatKuota cluster dan batas database di Amazon Aurora DSQL.

- Perhitungan batas penyimpanan mungkin memerlukan waktu untuk mencerminkan penyimpanan yang dibebaskan setelah menjalankan `DROP TABLE` perintah. Jika Anda membutuhkan kapasitas penyimpanan tambahan, lihat Kuota cluster untuk meminta pembaruan kuota.
- Untuk tabel besar di Aurora DSQL, gunakan katalog sistem untuk mengambil jumlah baris tabel alih-alih operasi. `COUNT(*)` Untuk informasi selengkapnya, lihat Menggunakan tabel dan perintah sistem di Aurora DSQL.
- Aurora DSQL mengelola izin melalui hibah tingkat skema. Pengguna admin membuat skema menggunakan `CREATE SCHEMA` dan memberikan akses ke peran lain menggunakan`GRANT USAGE ON SCHEMA`. Pengguna admin mengelola objek dalam skema publik, sementara pengguna non-admin membuat objek dalam skema yang dibuat pengguna. Peran admin dapat memberikan dirinya sendiri peran lain untuk mendapatkan izin pada objek yang dibuat pengguna. Untuk informasi selengkapnya, lihat Mengotorisasi peran database untuk menggunakan SQL dalam database Anda.
- Saat driver memanggil`PG_PREPARED_STATEMENTS`, Aurora DSQL memberikan tampilan seluruh cluster dari pernyataan yang disiapkan dalam cache. Anda mungkin melihat pernyataan yang lebih siap per koneksi daripada yang diharapkan untuk peran cluster dan IAM yang sama. Aurora DSQL mengelola nama pernyataan secara dinamis selama persiapan.
- Saat menghubungkan dari IPv4-only instance, pastikan klien Anda dikonfigurasi untuk koneksi IPv4. Beberapa klien PostgreSQL mencoba koneksi IPv4 dan IPv6 dalam mode dualstack. Jika koneksi IPv4 mengalami pelambatan, klien dapat mencoba IPv6 dan mengembalikan kesalahan pada host. `NetworkUnreachable` IPv4-only Konfigurasikan klien Anda untuk menggunakan IPv4 secara eksplisit untuk menghindari perilaku ini.
- Setelah pengguna admin membuat skema baru, `GRANT` dan `REVOKE` perubahan menyebar ke koneksi yang ada dalam masa koneksi (hingga satu jam). Untuk efek langsung, buat koneksi baru setelah perubahan izin.
- Selama skenario pemulihan klaster tertaut Multi-wilayah yang jarang terjadi, operasi pemulihan klaster otomatis mempertahankan ketersediaan tinggi, tetapi Anda mungkin mengalami kontrol konkurensi sementara atau kesalahan koneksi. Dalam kebanyakan kasus, hanya persentase

dari beban kerja Anda yang terpengaruh. Ketika Anda mengalami kesalahan sementara ini, coba kembali transaksi Anda atau hubungkan kembali dengan klien Anda.

- Beberapa klien SQL, seperti Datagrip, meminta metadata sistem yang luas untuk mengisi informasi skema. Aurora DSQL menyediakan metadata inti untuk fungsionalitas kueri SQL. Tampilan skema di klien ini dapat menampilkan informasi terbatas dibandingkan dengan set fitur lengkap mereka.
- Untuk memastikan kueri mengenali skema dan tabel yang baru dibuat, segarkan koneksi Anda setelah membuat atau menjatuhkan objek database. Ini termasuk skenario di mana Anda melihat `Schema Already Exists` kesalahan setelah menjatuhkan skema, atau saat menanyakan objek yang dibuat di koneksi lain. Putuskan sambungan dan sambungkan kembali, atau jalankan `SET search_path` lagi untuk menyegarkan cache katalog.
- Untuk kueri yang kompleks, gunakan `EXPLAIN ANALYZE VERBOSE` untuk mengidentifikasi operasi latensi tinggi dan mengoptimalkan rencana kueri. Mencakup indeks dapat secara signifikan mengurangi biaya DPU dengan mengaktifkan pemindaian hanya indeks alih-alih pemindaian tabel penuh. Untuk informasi selengkapnya, lihat Bekerja dengan rencana Aurora DSQL EXPLY.
- Batas koneksi dikelola di tingkat cluster. Lihat Kuota cluster untuk meminta pembaruan kuota.

Kuota cluster dan batas database di Amazon Aurora DSQL

Bagian berikut menjelaskan kuota cluster dan batas database untuk Aurora DSQL.

Kuota cluster

Anda Akun AWS memiliki kuota cluster berikut di Aurora DSQL. Untuk meminta peningkatan kuota layanan untuk kluster Single-region dan Multi-region dalam suatu kelompok tertentu Wilayah AWS, gunakan halaman konsol Service Quotas. Untuk kenaikan kuota lainnya, hubungi AWS Dukungan.

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL
Maksimum kluster wilayah tunggal per Akun AWS	20 kluster	Ya	Kode kesalahan API `ServiceQuotaExceededExce`
Kluster Multi-wilayah maksimum per Akun AWS	5 kluster	Ya	Kode kesalahan API `ServiceQuotaExceededExce`
Penyimpanan maksimum per cluster	Batas default 10 TiB, hingga 256 TiB dengan peningkatan batas yang disetujui	Ya	`DISK_FULL(53100)`
Koneksi maksimum per cluster	10.000 koneksi	Ya	`TOO_MANY_CONNECTIONS(53300)`

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL
Tingkat koneksi maksimum per cluster	100 koneksi per detik	Tidak	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Kapasitas burst koneksi maksimum per cluster	1.000 koneksi	Tidak	Tidak ada kode kesalahan
Pekerjaan pemulihan bersamaan maksimum	4	Tidak	Tidak ada kode kesalahan
Tingkat Isi Ulang Koneksi	100 koneksi per detik	Tidak	Tidak ada kode kesalahan
Aliran CDC maksimum per cluster	5 aliran	Tidak	Kode kesalahan API `ServiceQuotaExceededExcep`

Batas basis data di Aurora DSQL

Tabel berikut menjelaskan batas database di Aurora DSQL.

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Ukuran gabungan maksimum kolom yang	1 KiB	Tidak	`54000`	`ERROR: key size too large`

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
digunakan dalam kunci primer				
Ukuran gabungan maksimum kolom dalam indeks sekunder	1 KiB	Tidak	54000	ERROR: key size too large
Ukuran maksimum baris dalam tabel	2 MiB	Tidak	54000	ERROR: maximum row size excee
Ukuran maksimum kolom yang bukan bagian dari indeks	1 MiB	Tidak	54000	ERROR: maximum column size ex
Jumlah maksimum kolom dalam kunci primer atau indeks sekunder	8	Tidak	54011	ERROR: more than 8 column key are not supported
Jumlah maksimum kolom dalam tabel	255	Tidak	54011	ERROR: tables can have at mos

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Jumlah maksimum indeks dalam tabel	24	Tidak	`54000`	`ERROR: more than 24 indexes p allowed`
Ukuran maksimum semua data yang dimodifikasi dalam transaksi tulis	10 MiB	Tidak	`54000`	`ERROR: transaction size limit DETAIL: Current transaction s 10mb`
Jumlah maksimum baris tabel yang dapat dimutasi dalam blok transaksi	3.000 baris per transaksi. Lihat Pertimbangan Aurora DSQL untuk kompatibilitas PostgreSQL.	Tidak	`54000`	`ERROR: transaction row limit`
Jumlah dasar maksimum memori yang dapat digunakan oleh operasi kueri	128 MiB per transaksi	Tidak	`53200`	`ERROR: query requires too muc out of memory.`

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Jumlah maksimum skema yang didefinisikan dalam database	10	Tidak	54000	ERROR: more than 10 schemas n
Jumlah maksimum tabel dalam database	1.000 tabel	Tidak	54000	ERROR: creating more than 100 allowed
Jumlah maksimum database dalam sebuah cluster	1	Tidak	Tidak ada kode kesalahan	ERROR: unsupported statement
Waktu transaksi maksimum	5 menit	Tidak	54000	ERROR: transaction age limit exceeded
Durasi koneksi maksimum	60 menit	Tidak	Tidak ada kode kesalahan	Tidak ada pesan kesalahan
Jumlah maksimum tampilan dalam database	5.000	Tidak	54000	ERROR: creating more than 500 allowed
Ukuran definisi tampilan maksimum	2 MiB	Tidak	54000	ERROR: view definition too la

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Jumlah urutan maksimum	5.000	Tidak	`54000`	`ERROR: creating more than 500 not allowed`
Ukuran pesan maksimal	10 MiB	Tidak	`08P01`	`FATAL: invalid message length`

Untuk batas tipe data khusus untuk Aurora DSQL, lihat. Tipe data yang didukung di Aurora DSQL

Referensi Aurora DSQL API

Selain Konsol Manajemen AWS dan AWS Command Line Interface (AWS CLI), Aurora DSQL juga menyediakan antarmuka API. Anda dapat menggunakan operasi API untuk mengelola sumber daya Anda di Aurora DSQL.

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 lebih lanjut tentang AWS CLI, lihat AWS Command Line Interface referensi untuk Aurora DSQL.

Memecahkan masalah di Aurora DSQL

> **Note**
>
> Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan Aurora DSQL. Jika Anda menemukan masalah yang tidak tercantum di sini, hubungi AWS dukungan

Topik

- Memecahkan masalah kesalahan koneksi
- Memecahkan masalah kesalahan otentikasi
- Memecahkan masalah kesalahan otorisasi
- Memecahkan masalah kesalahan SQL
- Memecahkan masalah respons kontrol konkurensi
- Memecahkan masalah koneksi SSL/TLS

Memecahkan masalah kesalahan koneksi

kesalahan: kode kesalahan SSL tidak dikenal: 6 atau tidak dapat menerima koneksi, sni tidak diterima

Anda mungkin menggunakan versi psql lebih awal dari versi 14, yang tidak mendukung Server Name Indication (SNI). SNI diperlukan saat menghubungkan ke Aurora DSQL.

Anda dapat memeriksa versi klien Anda dengan`psql --version`.

kesalahan: NetworkUnreachable

`NetworkUnreachable`Kesalahan selama upaya koneksi mungkin menunjukkan bahwa klien Anda tidak mendukung koneksi IPv6, daripada menandakan masalah jaringan yang sebenarnya. Kesalahan ini biasanya terjadi pada IPv4-only instance karena bagaimana klien PostgreSQL menangani koneksi dual-stack. Ketika server mendukung mode dual-stack, klien ini pertama-tama menyelesaikan nama host ke alamat IPv4 dan IPv6. Mereka mencoba koneksi IPv4 terlebih dahulu, kemudian mencoba IPv6 jika koneksi awal gagal. Jika sistem Anda tidak mendukung IPv6, Anda

akan melihat `NetworkUnreachable` kesalahan umum alih-alih pesan “IPv6 tidak didukung” yang jelas.

Memecahkan masalah kesalahan otentikasi

Autentikasi IAM gagal untuk pengguna “...”

Saat Anda membuat token otentikasi Aurora DSQL IAM, durasi maksimum yang dapat Anda atur adalah 1 minggu. Setelah satu minggu, Anda tidak dapat mengautentikasi dengan token itu.

Selain itu, Aurora DSQL menolak permintaan koneksi Anda jika peran yang Anda anggap telah kedaluwarsa. Misalnya, jika Anda mencoba terhubung dengan peran IAM sementara meskipun token otentikasi Anda belum kedaluwarsa, Aurora DSQL akan menolak permintaan koneksi.

Untuk mempelajari lebih lanjut tentang bagaimana IAM bekerja dengan Aurora DSQL, lihat Memahami otentikasi dan otorisasi untuk Aurora DSQL dan di Aurora DSQL.AWS Identity and Access Management

Terjadi kesalahan (InvalidAccessKeyId) saat memanggil GetObject operasi: ID Kunci AWS Akses yang Anda berikan tidak ada dalam catatan kami

IAM menolak permintaan Anda. Untuk informasi selengkapnya, lihat Mengapa permintaan ditandatangani.

Peran IAM tidak ada <role>

Aurora DSQL tidak dapat menemukan peran IAM Anda. Untuk informasi selengkapnya, lihat peran IAM.

Peran IAM harus terlihat seperti ARN IAM

Lihat Pengidentifikasi IAM - ARN IAM untuk informasi lebih lanjut.

Pengguna salah untuk pemetaan tindakan

Kesalahan ini terjadi ketika jenis token otentikasi tidak cocok dengan peran database. Aurora DSQL menggunakan dua jenis token: `DbConnectAdmin` untuk `admin` peran dan `DbConnect` untuk peran basis data kustom.

- Jika Anda melihat`Wrong user to action mapping. user: admin, action: DbConnect`, gunakan `generate-db-connect-admin-auth-token` sebagai penggantigenerate-db-connect-auth-token.

- Jika Anda melihat`Wrong user to action mapping. user: myusername, action: DbConnectAdmin`, gunakan `generate-db-connect-auth-token` sebagai penggantigenerate-db-connect-admin-auth-token.

Memecahkan masalah kesalahan otorisasi

Peran tidak didukung <role>

Aurora DSQL tidak mendukung operasi. `GRANT` Lihat Subset perintah SQL yang didukung di Aurora DSQL.

Tidak dapat membangun kepercayaan dengan peran <role>

Aurora DSQL tidak mendukung operasi. `GRANT` Lihat Subset perintah SQL yang didukung di Aurora DSQL.

Peran tidak ada <role>

Aurora DSQL tidak dapat menemukan pengguna database tertentu. Lihat Mengotorisasi peran basis data kustom untuk terhubung ke klaster.

KESALAHAN: izin ditolak untuk memberikan kepercayaan IAM dengan peran <role>

Untuk memberikan akses ke peran database, Anda harus terhubung ke klaster Anda dengan peran admin. Untuk mempelajari lebih lanjut, lihat Mengotorisasi peran database untuk menggunakan SQL dalam database.

ERROR: peran harus memiliki atribut LOGIN <role>

Setiap peran database yang Anda buat harus memiliki `LOGIN` izin.

Untuk mengatasi kesalahan ini, pastikan Anda telah membuat Peran PostgreSQL dengan izin. `LOGIN` Untuk informasi selengkapnya, lihat MEMBUAT PERAN dan MENGUBAH PERAN dalam dokumentasi PostgreSQL.

KESALAHAN: peran tidak dapat dijatuhkan karena beberapa objek bergantung padanya <role>

Aurora DSQL mengembalikan kesalahan jika Anda menjatuhkan peran database dengan hubungan IAM sampai Anda mencabut hubungan menggunakan. `AWS IAM REVOKE` Untuk mempelajari lebih lanjut, lihat Mencabut otorisasi.

Memecahkan masalah kesalahan SQL

Kesalahan: Tidak didukung

Aurora DSQL tidak mendukung semua dialek. PostgreSQL-based Untuk mempelajari tentang apa yang didukung, lihat Fitur PostgreSQL yang Didukung di Aurora DSQL.

Kesalahan: gunakan **`CREATE INDEX ASYNC`** sebagai gantinya

Untuk membuat indeks pada tabel dengan baris yang ada, Anda harus menggunakan `CREATE INDEX ASYNC` perintah. Untuk mempelajari lebih lanjut, lihat Membuat indeks secara asinkron di Aurora DSQL.

Memecahkan masalah respons kontrol konkurensi

OC000 “ERROR: ubah konflik dengan transaksi lain (OC000)”

Transaksi ini mencoba memodifikasi tupel yang sama dengan transaksi bersamaan lainnya. Ini menunjukkan pertengkaran pada tupel yang dimodifikasi. Untuk mempelajari lebih lanjut, lihat Kontrol konkurensi di Aurora DSQL.

OC001 “ERROR: skema telah diperbarui oleh transaksi lain (OC001)”

Sesi Anda memiliki salinan cache dari katalog skema di versi V1, dimuat pada waktu T1.

Transaksi terpisah memperbarui katalog ke versi V2 pada waktu T2.

Pada saat T3, ketika sesi Anda menjalankan kueri, ia mendeteksi bahwa itu berada di belakang dan mencoba untuk rebase ke perubahan katalog baru. Dalam beberapa situasi rebase tidak dapat berhasil, dan Aurora DSQL mengembalikan respons OC001. `40001` Waktu antara T2 dan T3 dapat berkisar dari milidetik hingga menit, karena prosesor kueri menemukan perubahan katalog secara reaktif daripada menerima pembaruan proaktif.

Ketika Anda mencoba lagi dari sesi yang sama, Aurora DSQL menyegarkan cache katalog. Transaksi yang dicoba ulang menggunakan katalog V2 dan berhasil selama tidak ada perubahan katalog lebih lanjut yang terjadi sejak T2.

Memecahkan masalah koneksi SSL/TLS

Kesalahan SSL: verifikasi sertifikat gagal

Kesalahan ini menunjukkan bahwa klien tidak dapat memverifikasi sertifikat server. Pastikan bahwa:

1. Sertifikat Amazon Root CA 1 diinstal dengan benar. Lihat Mengkonfigurasi SSL/TLS sertifikat untuk koneksi Aurora DSQL petunjuk tentang cara memvalidasi dan menginstal sertifikat ini.
2. Variabel `PGSSLROOTCERT` lingkungan menunjuk ke file sertifikat yang benar.
3. File sertifikat memiliki izin yang benar.

Kode kesalahan SSL yang tidak dikenal: 6

Kesalahan ini terjadi dengan klien PostgreSQL di bawah versi 14. Tingkatkan klien PostgreSQL Anda ke versi 17 untuk mengatasi masalah ini.

Kesalahan SSL: skema tidak terdaftar (Windows)

Ini adalah masalah yang diketahui dengan klien psql Windows saat menggunakan sertifikat sistem. Gunakan metode file sertifikat yang diunduh yang dijelaskan dalam Menghubungkan dari Windows instruksi.

Memberikan umpan balik tentang Amazon Aurora DSQL

Jika Anda menemukan fitur yang penting untuk migrasi Anda tetapi saat ini tidak didukung di Aurora DSQL, AWS menyediakan beberapa saluran untuk umpan balik:

Saluran umpan balik

Server Aurora DSQL Discord

: Bergabunglah dengan server Aurora DSQL Discord untuk terhubung dengan tim dan komunitas AWS. Bagikan permintaan fitur, diskusikan tantangan migrasi, dan dapatkan umpan balik waktu nyata.

AWS Support

: Jika Anda memiliki paket AWS Support, buat kasus dukungan untuk mendiskusikan persyaratan spesifik dan kebutuhan timeline Anda.

AWS RE: POSTING

: Gunakan AWS re:Post untuk mengajukan pertanyaan dan berbagi umpan balik dengan komunitas dan pakar AWS.

Permintaan fitur yang efektif

Saat meminta fitur, berikan:

- Gunakan deskripsi kasus: Jelaskan apa yang ingin Anda capai dan mengapa
- Solusi saat ini: Jelaskan alternatif apa pun yang Anda coba
- Dampak bisnis: Jelaskan bagaimana fitur yang hilang memengaruhi timeline migrasi atau fungsionalitas aplikasi Anda
- Tingkat prioritas: Tunjukkan apakah ini memblokir migrasi Anda atau akan menjadi nice-to-have peningkatan

Catatan rilis untuk Aurora DSQL

Halaman ini menjelaskan fitur baru, peluncuran layanan, dan pembaruan penting untuk Aurora DSQL. Untuk perubahan khusus dokumentasi, lihat Riwayat dokumen.

2026

Juni 2026

Juni 8, 2026

Kompatibilitas PostgreSQL - Tipe data JSONB dengan kompresi - Aurora DSQL sekarang mendukung tipe data PostgreSQL, dengan kompresi opsional untuk data yang `JSONB` disimpan. Anda dapat menggunakan `JSONB` kolom untuk menyimpan data semi-terstruktur seperti metadata konfigurasi sistem, parameter API, dan log peristiwa bersama data relasional, serta kode dan alat yang ada yang bergantung pada `JSONB` jenis PostgreSQL bekerja dengan Aurora DSQL tanpa modifikasi. Aurora DSQL memungkinkan kompresi PostgreSQL secara default, menyimpan muatan yang lebih besar dengan lebih efisien. `JSONB` Untuk informasi selengkapnya, lihat Tipe data yang didukung di Aurora DSQL.

Mei 2026

14 Mei 2026

Fitur - Ubah pengambilan data (Pratinjau) - Aurora DSQL memperkenalkan dukungan untuk pengambilan data perubahan (CDC) dalam pratinjau, memungkinkan Anda untuk melakukan streaming perubahan database waktu nyata langsung ke Amazon Kinesis Data Streams. Kemampuan yang dikelola sepenuhnya ini menghilangkan kebutuhan untuk membangun atau memelihara saluran streaming khusus, membuatnya lebih mudah untuk membangun aplikasi yang digerakkan oleh peristiwa, menyalakan saluran analitik waktu nyata, dan menyinkronkan data di seluruh sistem. Untuk informasi selengkapnya, lihat Memulai dengan CDC.

11 Mei 2026

Perluasan Wilayah - Aurora DSQL tersedia di lima Wilayah tambahan - Single-Region klaster sekarang tersedia di Asia Pasifik (Hong Kong), Asia Pasifik (Mumbai), Asia Pasifik (Singapura), Eropa (Stockholm), dan Amerika Selatan (São Paulo). Untuk daftar lengkap Wilayah yang didukung, lihat Ketersediaan wilayah untuk Aurora DSQL.

4 Mei 2026

Kompatibilitas PostgreSQL - Tipe data JSON dengan kompresi - Aurora DSQL sekarang mendukung tipe data PostgreSQL dengan kompresi opsional. `JSON` Anda dapat menyimpan data semi-terstruktur seperti muatan API, objek konfigurasi, atau log peristiwa di samping data relasional. Aurora DSQL memungkinkan kompresi PostgreSQL secara default, menyimpan muatan yang lebih besar dengan lebih efisien. `JSON`

April 2026

April 24, 2026

Kompatibilitas PostgreSQL - MENGUBAH DUKUNGAN HAK ISTIMEWA DEFAULT - Aurora DSQL sekarang mendukung **`ALTER DEFAULT PRIVILEGES`** PostgreSQL untuk tabel, urutan, fungsi, dan jenis. Objek yang baru dibuat secara otomatis mewarisi hak istimewa default yang dikonfigurasi, menghapus kebutuhan akan `GRANT` pernyataan manual setelah setiap pembuatan objek.

April 24, 2026

Kompatibilitas PostgreSQL - BUAT SKEMA dengan pernyataan bersarang - Aurora DSQL sekarang `CREATE SEQUENCE` mendukung `GRANT` bersarang dan pernyataan `CREATE SCHEMA` dalam sebuah perintah, memungkinkan pengembang untuk menentukan objek skema dan izinnya dalam satu operasi atom.

April 24, 2026

Kompatibilitas PostgreSQL - DUKUNGAN UBAH NAMA - Aurora DSQL sekarang **`ALTER USER/ROLE/GROUP ... RENAME TO`** mendukung, memungkinkan administrator untuk USER/ROLE mengganti nama peran basis data tanpa menjatuhkan dan membuatnya kembali. Perubahan nama menyebar di semua sesi aktif melalui pemberitahuan versi katalog.

April 24, 2026

Kompatibilitas PostgreSQL - Dukungan ALTER RUTIN - Aurora DSQL sekarang **`ALTER ROUTINE ... SET SCHEMA`** mendukung`SET OWNER`,, dan, bersama dengan,,, dan. `RENAME TO` `GRANT` `REVOKE` `COMMENT ON` `DROP ROUTINE` `ROUTINE`Kata kunci adalah istilah payung PostgreSQL yang mencakup fungsi dan prosedur, memungkinkan pemetaan relasional objek (ORM) dan alat migrasi yang memancarkan sintaks untuk bekerja tanpa modifikasi. `ROUTINE`

April 21, 2026

Fitur - AWS Backup menambahkan dukungan Aurora DSQL untuk kebijakan cadangan AWS Organizations - sekarang AWS Backup mendukung cluster Aurora DSQL sebagai tipe sumber daya dalam kebijakan cadangan Organizations. AWS Administrator organisasi dapat menentukan aturan kebijakan cadangan yang secara langsung menargetkan klaster Aurora DSQL di seluruh akun anggota, memberikan kontrol yang lebih tepat atas sumber daya mana yang disertakan dalam rencana cadangan. Organization-wide

April 13, 2026

Alat Pengembang - Konektor Aurora DSQL untuk PHP (**`PDO_PGSQL`**) - Menambahkan Konektor Aurora DSQL untuk PHP yang menangani pembuatan token IAM, konfigurasi Secure Sockets Layer (SSL), dan penyatuan koneksi untuk aplikasi PHP. Konektor menyediakan percobaan ulang opt-in optimistic concurrency control (OCC) dengan backoff eksponensial. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor PHP.

Maret 2026

Maret 31, 2026

Alat Pengembang - Konektor Aurora DSQL untuk .NET (**`Npgsql`**) dan Rust (**`SQLx`**) - Menambahkan konektor untuk.NET dan Rust yang menangani pembuatan token IAM, konfigurasi SSL, dan penyatuan koneksi. Kedua konektor menyediakan percobaan ulang OCC opt-in dengan backoff eksponensial, penyedia kredensi IAM khusus, dan dukungan profil. AWS Untuk informasi lebih lanjut, lihat konektor Aurora DSQL.

Maret 30, 2026

Kinerja - Pernyataan INSERT ON CONFLICT yang lebih cepat - Aurora DSQL meningkatkan kinerja penulisan untuk pernyataan. `INSERT ON CONFLICT` Peningkatan skala dengan jumlah baris yang diusulkan untuk penyisipan per transaksi, sehingga beban kerja yang mengelompokkan banyak `INSERT ON CONFLICT` pernyataan ke dalam satu transaksi melihat keuntungan terbesar.

Maret 30, 2026

Kinerja - Validasi keunikan yang lebih cepat untuk beban kerja sisipan besar - Aurora DSQL sekarang memvalidasi batasan indeks unik sekunder dalam batch daripada satu kunci pada satu waktu, mempercepat beban kerja penyisipan dan migrasi yang besar. Percepatan ini paling menonjol untuk transaksi yang menyentuh banyak baris atau tabel dengan beberapa indeks

sekunder unik — beban kerja di mana validasi keunikan sebelumnya merupakan hambatan. Misalnya, memasukkan 1.000 baris ke dalam tabel dengan satu indeks sekunder unik sekarang selesai dalam 577 ms, turun dari 1.041 ms (1,8x lebih cepat).

Maret 26, 2026

Alat Pengembang - Konektor Aurora DSQL untuk Ruby (**pg**permata) - Menambahkan Konektor Aurora DSQL untuk Ruby yang menangani pembuatan token IAM, konfigurasi SSL, dan penyatuan koneksi. Konektor menyediakan percobaan ulang OCC opt-in dengan backoff eksponensial dan penyedia kredensi IAM khusus. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DSQL Aurora dengan konektor Ruby pg.

Februari 2026

Februari 25, 2026

Alat Pengembang - Integrasi untuk Kode Visual Studio SQLTools dan DBeaver - Menambahkan Driver Aurora DSQL untuk SQLTools dan Plugin Aurora DSQL untuk Edisi Komunitas DBeaver. Kedua integrasi menyederhanakan konektivitas database dengan secara otomatis menangani otentikasi IAM dan mengelola token akses secara transparan. Driver SQLTools juga tersedia di Open VSX Registry untuk digunakan dengan Code-compatible editor VS seperti Cursor dan Kiro.

Februari 25, 2026

Fitur - Aurora DSQL Playground - AWS meluncurkan taman bermain berbasis browser yang memungkinkan pengembang untuk berinteraksi dengan database Aurora DSQL tanpa memerlukan akun. AWS Dengan pengaturan nol atau konfigurasi infrastruktur, pengembang dapat membuat skema, memuat data, dan mengeksekusi kueri SQL langsung dari browser mereka dengan kumpulan data sampel bawaan. Untuk mulai menjelajah, kunjungi Aurora DSQL Playground.

Februari 25, 2026

Alat Pengembang - Dukungan untuk Kura-kura, Flyway, dan Prisma - Menambahkan integrasi untuk ORM populer dan alat migrasi database: adaptor untuk Kura-kura (Python ORM), dialek untuk Flyway (alat manajemen skema), dan alat CLI untuk Prisma (ORM). Node.js Integrasi ini menangani otentikasi IAM dan persyaratan kompatibilitas Aurora DSQL-specific secara otomatis.

Februari 19, 2026

Alat Pengembang - Go, Python, dan Node.js konektor dengan WebSocket dukungan - Menambahkan Konektor Aurora DSQL untuk Go (`pgx`), Python (), dan Node.js (untuk`asyncpg`)

yang menyederhanakan otentikasi IAM. WebSocket `Postgres.js` Setiap konektor secara otomatis menghasilkan token untuk setiap koneksi. `Postgres.js`Konektor juga mendukung WebSocket protokol untuk lingkungan di mana koneksi Transmission Control Protocol (TCP) tidak tersedia. Untuk informasi lebih lanjut, lihat konektor Aurora DSQL.

Februari 18, 2026

Integrasi AI - Aurora DSQL terintegrasi dengan kekuatan Kiro dan keterampilan agen AI - Integrasi Aurora DSQL dengan kekuatan Kiro dan keterampilan agen AI memungkinkan pengembang untuk membangun aplikasi Aurora lebih cepat dengan pengembangan. DSQL-backed AI-assisted Kekuatan Kiro menggabungkan server Aurora DSQL Model Context Protocol (MCP) dengan praktik terbaik pengembangan untuk instalasi satu klik. Keterampilan Aurora DSQL memperluas kemampuan yang sama ke agen pengkodean AI tambahan melalui Skills CLI. Untuk informasi lebih lanjut, lihat Aurora DSQL Steering: Skills and Powers.

Februari 13, 2026

Kompatibilitas PostgreSQL - Kolom identitas dan objek urutan - Aurora DSQL sekarang mendukung kolom identitas dan objek urutan, memungkinkan pengembang untuk menghasilkan peningkatan otomatis, ID berbasis bilangan bulat langsung di database menggunakan pola SQL yang sudah dikenal. Untuk informasi selengkapnya, lihat Urutan dan kolom identitas.

Februari 11, 2026

Perluasan Wilayah - Aurora DSQL tersedia di Wilayah Kanada dan Australia - Aurora DSQL sekarang tersedia dengan kluster Single-region di Asia Pasifik (Melbourne), Asia Pasifik (Sydney), Kanada (Tengah), dan Kanada Barat (Calgary). Untuk daftar lengkap Wilayah yang didukung, lihat Ketersediaan wilayah untuk Aurora DSQL.

Februari 5, 2026

Alat Pengembang - Konektor Aurora DSQL untuk Go - Menambahkan Konektor Aurora DSQL untuk Go, yang dibungkus `pgx` dengan otentikasi IAM otomatis untuk pembuatan token, konfigurasi SSL, dan manajemen koneksi. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor Go.

Februari 3, 2026

Kompatibilitas PostgreSQL - Dukungan indeks untuk tipe data NUMERIK - Aurora DSQL sekarang mendukung indeks pada `NUMERIC` tipe data, memungkinkan pelanggan untuk `NUMERIC` menggunakan kolom sebagai kunci utama dan dalam indeks sekunder untuk meningkatkan kinerja kueri pada data desimal dan keuangan.

Januari 2026

Januari 15, 2026

Integrasi - Aurora DSQL tersedia dalam v0 oleh Vercel - AWS database termasuk Aurora DSQL sekarang tersedia dalam v0 oleh Vercel, memungkinkan pengembang untuk dengan cepat berputar dan terhubung ke database Aurora DSQL langsung dari platform Vercel.

2025

Desember 2025

Desember 19, 2025

Integrasi AI - Pembaruan AWS Labs Aurora DSQL MCP Server - Server MCP yang diperbarui dengan pendekatan instalasi untuk Claude Code dan Codex, termasuk penyiapan. CLI-based Untuk informasi selengkapnya, lihat AWS Labs Aurora DSQL MCP Server.

Desember 17, 2025

Integrasi - Aurora DSQL tersedia di Vercel Marketplace - database AWS termasuk Aurora DSQL sekarang tersedia di Vercel Marketplace, memungkinkan pengembang untuk menyediakan cluster Aurora DSQL dalam hitungan detik langsung dari platform pengembang Vercel.

Desember 11, 2025

Fitur - Pembuatan cluster cepat - Aurora DSQL sekarang mendukung pembuatan cluster dalam hitungan detik, memungkinkan pengembang untuk secara instan menyediakan database Aurora DSQL untuk membuat prototipe ide baru dengan cepat dan memanfaatkan editor kueri terintegrasi di Konsol Manajemen untuk segera mulai membangun. AWS

Desember 11, 2025

Fitur - AWS PrivateLink dengan Direct Connect dukungan - Menambahkan pengaturan Sistem Nama Domain pribadi (DNS) dan opsi koneksi ID cluster untuk AWS PrivateLink dengan Direct Connect atau peering VPC Amazon. Untuk informasi selengkapnya, lihat Mengelola dan menghubungkan ke klaster Aurora DSQL menggunakan. AWS PrivateLink

November 2025

November 21, 2025

Alat Pengembang - Konektor Python, Node.js, dan JDBC - Aurora DSQL meluncurkan konektor Python,, dan Java Database Connectivity (JDBC) baru yang menyederhanakan otorisasi IAM Node.js, memungkinkan pengembang untuk terhubung ke cluster Aurora DSQL menggunakan driver PostgreSQL standar dengan manajemen token IAM otomatis. Untuk informasi lebih lanjut, lihat konektor Aurora DSQL.

November 21, 2025

Fitur - Volume penyimpanan meningkat menjadi 256 TiB - Cluster database Aurora DSQL sekarang mendukung hingga 256 terabyte volume penyimpanan, menggandakan batas 128 terabyte sebelumnya dan memungkinkan pelanggan untuk menyimpan dan mengelola database yang lebih besar dalam satu cluster database.

November 21, 2025

Fitur - Editor Kueri Terpadu di Konsol AWS Manajemen - Aurora DSQL sekarang menyediakan editor kueri terintegrasi untuk akses SQL berbasis browser di Konsol Manajemen. AWS Pelanggan dapat terhubung dengan aman ke cluster Aurora DSQL mereka dan menjalankan kueri SQL langsung dari konsol tanpa menginstal atau mengonfigurasi klien eksternal. Untuk informasi selengkapnya, lihat Memulai dengan Editor Kueri Aurora DSQL.

November 20, 2025

Fitur — perkiraan Statement-level biaya dalam paket kueri — Aurora DSQL sekarang menyediakan perkiraan biaya tingkat pernyataan dan rencana kueri, memunculkan rincian per kategori termasuk biaya komputasi, baca, tulis, dan Multi-wilayah dengan total perkiraan DPU untuk membantu pengembang segera mendapatkan visibilitas ke profil perkiraan biaya mereka. Untuk informasi selengkapnya, lihat Bekerja dengan Aurora DSQL menjelaskan rencana.

November 20, 2025

Integrasi - Menggunakan JupyterLab dengan Aurora DSQL - Menerbitkan panduan langkah demi langkah untuk menghubungkan Aurora DSQL dengan JupyterLab lingkungan AI lokal dan Amazon. SageMaker Untuk informasi selengkapnya, lihat Menggunakan JupyterLab dengan Aurora DSQL.

Oktober 2025

Oktober 31, 2025

Fitur - Titik akhir yang sesuai dengan FIPS 140-3 - Aurora DSQL sekarang mendukung titik akhir yang sesuai dengan Standar Pemrosesan Informasi Federal (FIPS) 140-3, memungkinkan pelanggan di industri yang diatur dan sektor pemerintah untuk memenuhi persyaratan keamanan federal saat menghubungkan ke cluster Aurora DSQL.

Oktober 24, 2025

Fitur - dukungan Resource-based kebijakan - Aurora DSQL sekarang mendukung kebijakan berbasis sumber daya, memungkinkan pelanggan untuk melampirkan kebijakan inline langsung ke cluster Aurora DSQL untuk kontrol akses berbutir halus dengan izin baru:,, dan. `PutClusterPolicy` `GetClusterPolicy` `DeleteClusterPolicy` Untuk informasi selengkapnya, lihat Resource-based kebijakan untuk Aurora DSQL.

Oktober 23, 2025

Perluasan Wilayah - Aurora DSQL tersedia di Eropa (Frankfurt) - Aurora DSQL sekarang tersedia di Wilayah Eropa (Frankfurt), memperluas ketersediaan bagi pelanggan Eropa yang perlu menyebarkan database SQL terdistribusi dengan residensi data di Uni Eropa (UE).

Agustus 2025

Agustus 26, 2025

Fitur - AWS Fault Injection Service integrasi - Aurora DSQL sekarang terintegrasi dengan AWS Fault Injection Service (AWS FIS), memungkinkan pelanggan untuk menyuntikkan kegagalan ke dalam klaster DSQL Aurora wilayah tunggal dan Multi-wilayah untuk menguji toleransi kesalahan dan memvalidasi penanganan kesalahan aplikasi dan mekanisme pemulihan.

Juli 2025

Juli 29, 2025

Fitur - AWS Backup meningkatkan alur kerja pemulihan Aurora DSQL Multi-wilayah - AWS Backup meningkatkan alur kerja pemulihan Aurora DSQL Multi-wilayah, menyederhanakan proses pemulihan cadangan Aurora DSQL untuk pemulihan bencana dan skenario migrasi data. Wilayah AWS Untuk informasi selengkapnya, lihat Backup dan restore untuk Aurora DSQL.

Juli 3, 2025

Perluasan Wilayah - Aurora DSQL tersedia di Asia Pasifik (Seoul) - Aurora DSQL sekarang tersedia di Wilayah Asia Pasifik (Seoul), memperluas ketersediaan untuk pelanggan di Korea Selatan dan kawasan Asia Pasifik yang lebih luas.

Mei 2025

27 Mei 2025

Rilis Utama - Ketersediaan umum Aurora DSQL - Aurora DSQL sekarang tersedia secara umum (GA). Aurora DSQL adalah database SQL terdistribusi tanpa server dengan kompatibilitas PostgreSQL, ketersediaan tinggi aktif-aktif, dan skala hampir tidak terbatas. Rilis GA mencakup CloudWatch pemantauan, AWS Backup integrasi, dan enkripsi data dengan kunci KMS yang dikelola pelanggan. Untuk informasi lebih lanjut, lihat Apa itu Aurora DSQL?

8 Mei 2025

Fitur — AWS PrivateLink untuk Aurora DSQL — Aurora DSQL sekarang AWS PrivateLink mendukung, memungkinkan pelanggan untuk menyederhanakan konektivitas jaringan pribadi antara virtual private cloud (VPC), Aurora DSQL, dan pusat data lokal menggunakan antarmuka titik akhir VPC Amazon dan alamat IP pribadi. Untuk informasi selengkapnya, lihat Mengelola dan menghubungkan ke klaster Aurora DSQL menggunakan. AWS PrivateLink

2024

Desember 2024

Desember 3, 2024

Rilis Utama - Peluncuran pratinjau publik Aurora DSQL - Aurora DSQL diluncurkan dalam pratinjau publik sebagai database SQL terdistribusi tanpa server baru. Aurora DSQL menawarkan skala yang hampir tidak terbatas, kompatibilitas PostgreSQL v16, dukungan aktif-aktif multi-wilayah dengan perjanjian tingkat layanan ketersediaan 99,999% (SLA), otentikasi IAM, dan kontrol konkurensi optimis. Untuk informasi lebih lanjut, lihat Apa itu Aurora DSQL?

Riwayat dokumen untuk Panduan Pengguna Amazon Aurora DSQL

Tabel berikut menjelaskan rilis dokumentasi untuk Aurora DSQL.

Perubahan	Deskripsi	Tanggal
Konten baru: tipe data JSONB	Menambahkan dokumentasi untuk tipe `jsonb` data di Aurora DSQL, termasuk perilaku kompresi otomatis, batas ukuran terkompresi 1 MiB, dan operator JSONB yang didukung. Memperbarui bagian tipe data runtime kueri untuk mencerminkan bahwa sekarang `jsonb` tersedia sebagai tipe penyimpanan asli. Untuk informasi selengkapnya, lihat Tipe data yang didukung di Aurora DSQL.	Juni 5, 2026
Konten baru: Ubah aliran pengambilan data (CDC)	Menambahkan dokumentasi untuk aliran Aurora DSQL change data capture (CDC), yang menangkap perubahan tingkat baris yang berkomitmen dan mengirimkannya ke Amazon Kinesis Data Streams. Untuk informasi selengkapnya, lihat Mengubah aliran pengambilan data (CDC).	13 Mei 2026

Konten baru: Dukungan kata kunci STORAGE di CREATE TABLE dan ALTER TABLE	Menambahkan dokumentasi untuk `STORAGE` klausa di `CREATE TABLE` dan`ALTER TABLE`, yang menetapkan mode penyimpanan (`PLAIN`,`EXTERNAL`,`EXTENDED`,`MAIN`, atau`DEFAULT`) untuk kolom. Ini dapat digunakan untuk mengontrol perilaku kompresi untuk tipe data panjang variabel seperti. `json` Untuk informasi selengkapnya, lihat MEMBUAT TABEL dan MENGUBAH TABEL.	1 Mei 2026
Konten baru: tipe data JSON	Menambahkan dokumentasi untuk tipe `json` data di Aurora DSQL, termasuk perilaku kompresi otomatis, batas ukuran terkompresi 1 MiB, dan fungsi dan operator JSON yang didukung. Memperbarui panduan tipe runtime JSONB untuk mencerminkan ketersediaan penyimpanan asli. `json` Untuk informasi selengkapnya, lihat Tipe data yang didukung di Aurora DSQL.	1 Mei 2026

Konten baru: ALTER TABLE ADD CONSTRAINT MENGGUNAKAN INDEX	Menambahkan dokumentasi untuk `ALTER TABLE ... ADD table_constraint_using_index`, yang menambahkan `UNIQUE` kendala ke tabel berdasarkan indeks unik yang ada. Untuk informasi lebih lanjut, lihat ALTER TABLE.	April 20, 2026
Konten baru: Aurora DSQL Connector untuk PHP `PDO_PGSQL`	Menambahkan dokumentasi untuk Konektor Aurora DSQL untuk PHP `PDO_PGSQL`, yang dibungkus dengan otentikasi `PDO_PGSQL` IAM otomatis. Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi untuk aplikasi PHP. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor PHP.	April 10, 2026

Konten baru: Aurora DSQL Connector untuk Rust SQLx	Menambahkan dokumentasi untuk Konektor Aurora DSQL untuk RustSQLx, yang dibungkus dengan otentikasi IAM otomatisSQLx. Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi untuk aplikasi Rust. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor Rust.	Maret 27, 2026
Konten baru: Aurora DSQL Connector untuk .NET Npgsql	Menambahkan dokumentasi untuk Konektor Aurora DSQL untuk .NET Npgsql, yang membungkus Npgsql dengan otentikasi IAM otomatis. Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi untuk aplikasi.NET. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor.NET Npgsql.	Maret 20, 2026

Konten yang diperbarui: referensi perintah SQL dan kueri sistem	Menambahkan MULAI TRANSAKSI dan ROLLBACK ke referensi perintah kontrol transaksi, dengan END dan ABORT sebagai alias. Menambahkan kueri sistem yang berguna untuk mengambil informasi versi Aurora DSQL dan PostgreSQL. Untuk informasi selengkapnya, lihat referensi kompatibilitas PostgreSQL.	Maret 13, 2026
Konten yang diperbarui: Memuat data ke Aurora DSQL	Memperbarui panduan pemuatan data dengan `\copy` penggunaan sisi klien, praktik terbaik INSERT, dan panduan untuk membuat tabel sebelum memuat. Untuk informasi selengkapnya, lihat Memuat data ke Aurora DSQL.	Maret 13, 2026
Konten baru: Aurora DSQL Connector untuk Ruby pg	Menambahkan dokumentasi untuk Konektor Aurora DSQL untuk Ruby pg, yang membungkus permata pg dengan otentikasi IAM otomatis. Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi untuk aplikasi Ruby. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DSQL Aurora dengan konektor Ruby pg.	Maret 12, 2026

Konten yang diperbarui: Pekerjaan DDL asinkron	Memperbarui `sys.jobs` dokumentasi dengan detail yang diperluas tentang pemantauan dan pengelola an operasi DDL asinkron. Untuk informasi selengkapnya, lihat referensi kompatibilitas PostgreSQL.	6 Maret 2026
Konten yang diperbarui: Pembuatan token otentikasi PHP	Menambahkan tab PHP SDK ke halaman pembuatan token otentikasi. Untuk informasi selengkapnya, lihat Menghasil kan token autentikasi.	Maret 5, 2026
Konten baru: Memuat data ke Aurora DSQL	Menambahkan panduan untuk memuat data ke cluster Aurora DSQL, termasuk penggunaa n utilitas loader Aurora DSQL. Untuk informasi selengkapnya, lihat Memuat data ke Aurora DSQL.	Maret 5, 2026

Konten baru: Urutan dan kolom identitas	Ditambahkan dukungan untuk urutan dan kolom identitas. Halaman referensi perintah SQL baru untuk CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE, dan fungsi manipulasi urutan. Diperbarui CREATE TABLE dan ALTER TABLE untuk menyertak an sintaks kolom identitas. Menambahkan panduan baru untuk memilih jenis pengenal dan ukuran cache. Untuk informasi selengkapnya, lihat Urutan dan kolom identitas.	Februari 11, 2026
Konten baru: Aurora DSQL Connector for Go	Menambahkan dokumentasi untuk Konektor Aurora DSQL untuk Go, yang membungku s pgx dengan otentikasi IAM otomatis. Konektor menangani pembuatan token, konfigurasi SSL, dan manajemen koneksi untuk aplikasi Go. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor Go.	Februari 5, 2026

Konten yang diperbarui: Alat konektivitas cluster Amazon Aurora DSQL	Menata ulang dokumentasi alat konektivitas cluster untuk memperjelas perbedaan antara konektor AWS yang disediakan, adaptor, dan alat pihak ketiga. Menambahkan tautan yang hilang ke sampel kode. Untuk informasi selengkapnya, lihat Alat konektivitas kluster Amazon Aurora DSQL.	Januari 26, 2026
Konten baru: Plugin Aurora DSQL untuk Edisi Komunitas DBeaver	Menambahkan dokumentasi untuk Plugin Aurora DSQL untuk DBeaver Community Edition, yang memungkinkan otentikasi IAM dan pengaturan koneksi yang disederhanakan untuk cluster Aurora DSQL. Termasuk instruksi instalasi, konfigurasi koneksi, dan panduan pemecahan masalah. Untuk informasi selengkapnya, lihat Menggunakan DBeaver untuk mengakses Aurora DSQL.	Januari 26, 2026

Konten baru: Aurora DSQL Driver untuk SQLTools	Menambahkan dokumentasi untuk Driver Aurora DSQL untuk SQLTools, ekstensi Visual Studio Code yang memungkinkan pengembang untuk terhubung ke dan menanyakan database Aurora DSQL langsung dari VS Code dengan otentikasi IAM otomatis. Untuk informasi selengkapnya, lihat Menggunakan driver Aurora DSQL untuk SQLTools.	Januari 26, 2026
Konten yang diperbarui: Aurora DSQL Steering: Keterampilan dan Kekuatan	Menambahkan dokumentasi untuk mendukung instalasi menggunakan CLI Keterampilan untuk dukungan agnostik agen. Skills CLI menyediakan metode pengaturan yang disederhanakan yang bekerja di beberapa asisten pengkodean AI termasuk Claude Code, Cursor, Copilot, Gemini, dan lainnya. Untuk informasi lebih lanjut, lihat Aurora DSQL Steering: Skills and Powers.	Januari 23, 2026

Konten baru: Aurora DSQL Adapter untuk Tortoise ORM	Menambahkan dukungan untuk Tortoise ORM, kerangka ORM async Python. Adaptor Aurora DSQL untuk Tortoise ORM memungkinkan pengembang untuk menggunakan Tortoise ORM dengan cluster Aurora DSQL. Untuk informasi selengkapnya, lihat Adaptor dan dialek Aurora DSQL.	Januari 23, 2026
Konten baru: Aurora DSQL Steering: Keterampilan dan Kekuatan	Menambahkan dokumentasi baru untuk mengonfigurasi kemudi AI dengan Aurora DSQL menggunakan keterampilan dan kekuatan. Termasuk instruksi pengaturan untuk Kiro Powers, Claude Skills, Gemini Skills, dan Codex Skills. Untuk informasi lebih lanjut, lihat Aurora DSQL Steering: Skills and Powers.	Januari 16, 2026
Dukungan indeks tipe data numerik	Ditambahkan dukungan indeks untuk tipe `numeric` data di Aurora DSQL. Anda sekarang dapat menggunakan `numeric` kolom sebagai kunci utama dan dalam indeks sekunder. Untuk informasi selengkapnya, lihat Tipe data yang didukung di Aurora DSQL.	Januari 13, 2026

Konten yang diperbarui: Subset perintah SQL yang didukung	Dokumentasi perintah SQL direorganisasi ke halaman terpisah untuk meningkatkan navigasi dan kejelasan. Setiap perintah (CREATE TABLE, ALTER TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW) sekarang memiliki halaman khusus sendiri. Untuk informasi selengkapnya, lihat Subset perintah SQL yang didukung.	Januari 6, 2026
Konten yang diperbarui: AWS Labs Aurora DSQL MCP Server	Memperbarui dokumenta si MCP Server dengan pendekatan instalasi terperinc i untuk Claude Code dan Codex, termasuk contoh file CLI-based setup dan konfigurasi. Menambahkan panduan komprehensif untuk menemukan file konfigura si klien MCP di berbagai alat pengembangan. Untuk informasi selengkapnya, lihat AWS Labs Aurora DSQL MCP Server.	Desember 19, 2025

Konten yang diperbarui: Migrasi dari PostgreSQL ke Aurora DSQL	Menyusun ulang bagian kompatibilitas PostgreSQL sebagai panduan migrasi komprehensif. Termasuk informasi kompatibilitas kerangka kerja, pola migrasi umum, perbedaan arsitektur, dan panduan AI-assisted migrasi. Ditambahkan chapter baru untuk memberikan umpan balik pada Aurora DSQL. Untuk informasi selengkapnya, lihat Migrasi dari PostgreSQL ke Aurora DSQL.	Desember 16, 2025
Konten yang diperbarui: Menghubungkan ke Aurora DSQL menggunakan AWS PrivateLink	Menambahkan dokumentasi untuk penyiapan DNS pribadi dan opsi koneksi ID cluster untuk mendukung pelanggan yang menggunakan AWS PrivateLink dengan Direct Connect atau peering VPC Amazon. Termasuk panduan untuk menghubungkan tanpa DNS pribadi menggunakan opsi `amzn-cluster-id` koneksi. Untuk informasi selengkapnya, lihat Mengelola dan menghubungkan ke klaster Aurora DSQL menggunakan. AWS PrivateLink	Desember 11, 2025

Konten yang diperbarui: Siklus hidup cluster Aurora DSQL	Dokumentasi yang diperbarui untuk manajemen siklus hidup klaster Aurora DSQL. Menjelaskan definisi status klaster, transisi status, dan operasi yang tersedia selama status Idle dan Tidak Aktif. Untuk informasi selengkapnya, lihat Siklus hidup klaster Aurora DSQL.	Desember 4, 2025
Konten baru: Konektor Aurora DSQL untuk Python dan Node.js	Menambahkan dokumentasi untuk konektor Aurora DSQL untuk Python (psycopg, psycopg2, asyncpg) dan (node-postgres,). Node.js Postgres.js Konektor ini mengintegrasikan otentikasi IAM untuk menghubungkan aplikasi ke cluster Aurora DSQL. Untuk informasi lebih lanjut, lihat konektor Aurora DSQL.	November 21, 2025
Konten baru: Menggunakan JupyterLab dengan Aurora DSQL	Menambahkan panduan langkah demi langkah untuk menghubungkan dan menanyakan Aurora DSQL JupyterLab menggunakan dengan Python. Termasuk instruksi untuk JupyterLab instalasi lokal dan lingkungan Amazon SageMaker AI. Untuk informasi selengkapnya, lihat Menggunakan JupyterLab dengan Aurora DSQL.	November 20, 2025

konten diperbarui: Kuota untuk Aurora DSQL	Kuota penyimpanan cluster maksimum yang diperbaru i dari 128 TiB menjadi 256 TiB. Untuk informasi lebih lanjut, lihat Kuota untuk Aurora DSQL.	November 19, 2025
Konten baru: Memulai dengan Aurora DSQL Query Editor	Menambahkan dokumenta si untuk menggunakan Aurora DSQL Query Editor di Management Console. AWS Termasuk prasyarat , pengaturan koneksi, dan instruksi eksekusi kueri. Untuk informasi selengkapnya, lihat Memulai dengan Editor Kueri Aurora DSQL.	November 18, 2025

Resource-based dukungan kebijakan untuk Amazon Aurora DSQL	Menambahkan dukungan kebijakan berbasis sumber daya (RBP) dengan izin baru:,, dan. `PutClusterPolicy` `GetClusterPolicy` `DeleteClusterPolicy` Izin ini memungkinkan pengelolaan kebijakan sebaris yang dilampirkan ke cluster Aurora DSQL untuk kontrol akses berbutir halus. Kebijakan terkelola yang diperbarui AmazonAuroraDSQLFullAccess AmazonAuroraDSQLReadOnlyAccess,, dan AmazonAuroraDSQLConsoleFullAccess untuk menyertakan kemampuan RBP. Untuk informasi selengkapnya, lihat kebijakan AWS terkelola untuk Amazon Aurora DSQL.	Oktober 15, 2025
Konektor Aurora DSQL JDBC	Menambahkan dokumentasi untuk Konektor Aurora DSQL JDBC, konektor PgJDBC yang mengintegrasikan Autentikasi IAM untuk menghubungkan aplikasi Java ke kluster Amazon Aurora DSQL. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster Aurora DSQL dengan konektor JDBC.	September 2, 2025

AWS pembaruan kebijakan terkelola untuk AWS FIS integrasi	Diperbarui `AmazonAuroraDSQLFullAccess` dan `AmazonAuroraDSQLConsoleFullAccess` kebijakan untuk mendukung AWS Fault Injection Service integrasi dengan Aurora DSQL. Ini memungkinkan Anda untuk menyuntikkan kegagalan ke dalam kluster DSQL Aurora wilayah tunggal dan Multi-wilayah untuk menguji toleransi kesalahan aplikasi Anda. Untuk selengkapnya tentang kebijakan ini, lihat pembaruan kebijakan AWS terkelola.	Agustus 19, 2025
Ketersediaan umum (GA) Amazon Aurora DSQL	Amazon Aurora DSQL sekarang tersedia secara umum dengan dukungan tambahan untuk CloudWatch pemantauan, fitur perlindungan data yang ditingkatkan, dan integrasi. AWS Backup Untuk informasi selengkapnya, lihat Memantau Aurora DSQL dengan, CloudWatchMencadangkan dan memulihkan Amazon Aurora DSQL, dan Enkripsi data untuk Amazon Aurora DSQL.	27 Mei 2025

AmazonAuroraDSQLFullAccess memperbarui	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga menambahkan kemampuan untuk menggunakan kunci KMS yang dikelola pelanggan untuk enkripsi cluster. Untuk informasi selengkapnya, lihat AmazonAuroraDSQLFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	21 Mei 2025
AmazonAuroraDSQLConsoleFullAccess memperbarui	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL melalui. AWS Console Home Ini termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga mendukung penggunaan kunci KMS yang dikelola pelanggan untuk enkripsi dan peluncuran cluster. AWS CloudShell Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	21 Mei 2025

AmazonAuroraDSQLReadOnlyAccess memperbarui	Termasuk kemampuan untuk menentukan nama layanan titik akhir VPC yang benar saat menghubungkan ke cluster Aurora DSQL Anda melalui Aurora DSQL menciptakan titik akhir unik per sel, sehingga API ini membantu memastikan Anda dapat mengidentifikasi titik akhir yang benar untuk cluster Anda dan menghindari kesalahan koneksi. AWS PrivateLink Untuk informasi selengkapnya, lihat AmazonAuroraDSQLReadOnlyAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

AmazonAuroraDSQLFullAccess memperbarui	Kebijakan ini menambahkan empat izin baru untuk membuat dan mengelola kluster database di beberapa Wilayah AWS:`PutMultiRegionProperties` ,, `PutWitnessRegion` `AddPeerCluster` , dan. `RemovePeerCluster` Izin ini mencakup kontrol tingkat sumber daya dan kunci kondisi sehingga Anda dapat mengontrol pengguna kluster mana yang dapat Anda modifikasi. Kebijakan ini juga menambahkan `GetVpcEndpointServiceName` izin untuk membantu Anda terhubung ke klaster Aurora DSQL Anda. AWS PrivateLink Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

AmazonAuroraDSQLConsoleFullAccess memperbarui	Menambahkan izin baru ke Aurora DSQL untuk mendukung manajemen klaster Multi-wilayah dan koneksi titik akhir VPC. Izin baru meliputi: `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` . Lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025
AuroraDsqlServiceLinkedRolePolicy memperbarui	Menambahkan kemampuan untuk memublikasikan metrik ke `AWS/Usage` `CloudWatch` ruang nama `AWS/AuroraDSQL` dan ruang nama ke kebijakan. Hal ini memungkinkan layanan atau peran terkait untuk memancarkan data penggunaan dan kinerja yang lebih komprehensif ke CloudWatch lingkungan Anda. Untuk informasi selengkapnya, lihat AuroraDsqlServiceLinkedRolePolicydan Menggunakan peran terkait layanan di Aurora DSQL.	8 Mei 2025

AWS PrivateLink untuk Amazon Aurora DSQL	Aurora DSQL sekarang mendukung. AWS PrivateLink Dengan AWS PrivateLink, Anda dapat menyederhanakan konektivitas jaringan pribadi antara virtual private cloud (VPC), Aurora DSQL, dan pusat data lokal Anda menggunakan antarmuka titik akhir VPC Amazon dan alamat IP pribadi. Untuk informasi selengkapnya, lihat Mengelola dan menghubungkan ke kluster DSQL Amazon Aurora menggunakan. AWS PrivateLink	8 Mei 2025
Rilis awal	Rilis awal Panduan Pengguna Amazon Aurora DSQL.	Desember 3, 2024

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.